ORACLE
PEOPLESOFT ENTERPRISE

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode API Reference

March 2007

ORACLE

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode API Reference
SKU PT849PCR-B 0307
Copyright © 1988-2007, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Open Source Disclosure

Oracle takes no responsibility for its use or distribution of any open source or shareware software or documentation and disclaims any and all liability or damages resulting from use of said software or documentation. The following open source software may be used in Oracle's PeopleSoft products and the following disclaimers are provided.

Apache Software Foundation

This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 2000-2003. The Apache Software Foundation. All rights reserved. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

OpenSSL

Copyright © 1998-2005 The OpenSSL Project. All rights reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Loki Library

Copyright © 2001 by Andrei Alexandrescu. This code accompanies the book: Alexandrescu, Andrei. "Modern C++ Design: Generic Programming and Design Patterns Applied". Copyright © 2001 Addison-Wesley. Permission to use, copy, modify, distribute and sell this software for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

Helma Project

Copyright © 1999-2004 Helma Project. All rights reserved. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE HELMA PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Helma includes third party software released under different specific license terms. See the licenses directory in the Helma distribution for a list of these license.

Sarissa

Copyright © 2004 Manos Batsis.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

ICU

ICU License - ICU 1.8.1 and later COPYRIGHT AND PERMISSION NOTICE Copyright © 1995-2003 International Business Machines Corporation and others. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation. THE SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE. Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

Sun's JAXB Implementation – JDSDK 1.5 relaxngDatatype.jar 1.0 License

Copyright © 2001, Thai Open Source Software Center Ltd, Sun Microsystems. All rights reserved.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

W3C IPR SOFTWARE NOTICE

Copyright © 2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

Note: The original version of the W3C Software Copyright Notice and License could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720.

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

Contents

General Preface

About This PeopleBook cxiii

PeopleSoft Enterprise Application Prerequisites cxiii

Application Fundamentals cxiii

Documentation Updates and Printed Documentation cxiv

Obtaining Documentation Updates cxiv

Downloading and Ordering Printed Documentation cxiv

Additional Resources cxv

Typographical Conventions and Visual Cues cxvi

Typographical Conventions cxvi

Visual Cues cxvii

Country, Region, and Industry Identifiers cxvii

Currency Codes cxviii

Comments and Suggestions cxviii

Common Elements Used in PeopleBooks cxviii

Preface

PeopleCode API Reference Preface cxxi

PeopleCode API Reference cxxi

PeopleCode Typographical Conventions cxxi

Chapter 1

AESection Class 1

Understanding AESection Class 1

How an AESection is Accessed 2

AESection Example 3

Data Type of an AESection Object 4

Scope of an AESection Object 5

AESection Class Built-in Function 5

AESection Class Methods 5

AddStep 5

Close 6

Open 6

Copyright © 1988-2007, Oracle. All rights reserved.

Save....8
SetSQL....8
SetTemplate....9
AESection Class Property....10
IsOpen....10

Chapter 2
Analytic Calculation Engine Classes....11
Understanding the Analytic Calculation Engine Classes....11
Using the Analytic Calculation Engine Classes with PeopleSoft Application Engine....12
Running Synchronously....12
Using Trees....12
Error Handling....13
Data Types for Analytic Calculation Engine Classes....13
Scope of Analytic Calculation Engine Classes....13
Analytic Calculation Engine Classes Built-in Functions....14
AnalyticInstance Class Methods....14
CheckAsyncStatus....14
CheckStatus....15
Copy....15
Delete....16
GetAnalyticModel....17
Load....17
RunAsync....19
RunSync....19
Terminate....20
Unload....20
AnalyticInstance Class Properties....21
AnalyticType....21
ID....21
Messages....21
AnalyticModel Class Methods....21
AddMember....21
AttachTree....22
CalculateCube....24
DetachTree....24
GetCubeCollection....25
GetCellProperties....25
GetMembers....26

Copyright © 1988-2007, Oracle. All rights reserved.

GetTree....27
Recalculate....28
RenameMember....29
AnalyticModel Class Property....29
Messages....29
CubeCollection Class....30
CubeCollection Class Methods....30
CollapseNode....30
DrillIntoNode....31
DrillOutOfNode....31
ExpandNode....32
GetData....33
GetDimFilter....34
GetDimSort....35
GetLayout....35
GetRowCount....36
GetSlice....36
SetData....37
SetDimensionOrder....37
SetDimFilter....38
SetDimSort....38
SetSlice....39
ShowHierarchy....40
UnsetDimFilter....41
UnsetDimSort....41
CubeCollection Class Property....42
Messages....42

Chapter 3
Analytic Calculation Engine Metadata Classes....43
Understanding the PeopleSoft Analytic Calculation Engine Metadata Classes....43
Using the Analytic Calculation Engine Metadata Classes....44
Inserting and Deleting Objects....44
Building a Rule as an Object....44
Error Handling....45
Data Types of the Analytic Calculation Engine Metadata Objects....45
Scope of Analytic Calculation Engine Metadata Objects....45
How to Import the Analytic Calculation Engine Metadata Classes....46
How to Create an Analytic Calculation Engine Metadata Class Object....47

Copyright © 1988-2007, Oracle. All rights reserved.

Analytic Calculation Engine Metadata Classes Constructor....47
AnalyticModelDefn....47
AnalyticModelDefn Class....48
AnalyticModelDefn Class Methods....48
AddCube....48
AddCubeCollection....49
AddDimension....49
AddExplicitDimensionSet....50
AddOrganizer....50
AddUserFunction....51
CopyCube....52
CopyCubeCollection....52
CopyDimension....53
CopyExplicitDimensionSet....53
CopyTo....54
CopyUserFunction....54
Create....55
Delete....55
DeleteCube....56
DeleteCubeCollection....57
DeleteDimension....57
DeleteExplicitDimensionSet....58
DeleteOrganizer....58
DeleteUserFunction....59
Get....60
GetCube....60
GetCubeCollection....61
GetCubeCollectionNames....61
GetCubeNames....62
GetDimension....62
GetDimensionNames....63
GetExplicitDimensionSet....63
GetExplicitDimensionSetNames....64
GetOrganizer....64
GetOrganizerNames....65
GetUserFunction....65
GetUserFunctionNames....66
Rename....67
RenameCube....67
RenameCubeCollection....68

Copyright © 1988-2007, Oracle. All rights reserved.

RenameDimension....68
RenameExplicitDimensionSet....69
RenameOrganizer....70
RenameUserFunction....70
Save....71
Validate....72
AnalyticModelDefn Properties....72
CircularFormulaWarn....72
Description....72
IsValid....73
LongDescription....73
MaxDelta....73
MaxIterations....73
Messages....74
Name....74
ResolveCircularDeps....74
DimensionDefn Class....74
DimensionDefn Class Properties....75
AggregateSequence....75
AggregationUserFunction....75
Comments....75
Name....75
TotalMemberName....75
ExplicitDimensionSet Class....76
ExplicitDimensionSet Methods....76
AttachDimension....76
DetachDimension....76
GetDimensionNames....77
ExplicitDimensionSet Properties....77
Name....77
SequenceNumber....78
CubeDefn Class....78
CubeDefn Class Methods....78
AttachDimension....78
DetachDimension....79
GetCauses....79
GetCircularDeps....80
GetDimensionAggregate....81
GetEffects....81
GetDimensionNames....82

Copyright © 1988-2007, Oracle. All rights reserved.

GetRule....82
SetDimensionAggregate....83
SetRule....83
UsesDimension....84
CubeDefn Class Properties....84
CalcAggregates....84
Comments....84
DimensionCount....85
FormatType....85
IsVirtual....85
Name....85
ValueDimensionName....85
CubeCollectionDefn Class....86
CubeCollectionDefn Class Methods....86
AttachCube....86
DetachCube....87
GetAggregateMapping....87
GetCubeNames....88
GetDimensionNames....88
GetDimSort....89
GetFieldMapping....89
GetFilter....90
GetPersistAggregate....90
SetAggregateMapping....91
SetDimSort....92
SetFieldMapping....93
SetFilter....93
SetPersistAggregate....94
UsesCube....95
UsesDimension....95
CubeCollectionDefn Class Properties....96
AggregateRecName....96
Comments....96
CubeCount....96
DimensionCount....96
Name....96
RecordName....97
UserFunctionDefn Class....97
UserFunctionDefn Class Methods....97
GetRule....97

Copyright © 1988-2007, Oracle. All rights reserved.

SetRule....97
UserFunctionDefn Class Properties....98
Comments....98
Name....98
OrganizerDefn Class....98
OrganizerDefn Class Methods....99
AttachPart....99
DetachPart....99
GetPartNames....100
UsesPart....100
OrganizerDefn Class Properties....101
Comments....101
Name....101
RuleDefn Class....102
RuleDefn Class Methods....102
AddRuleExpression....102
GenerateRule....102
RuleDefn Class Property....103
RuleString....103
RuleExpressions Classes....103
Using the Constants Class....104
Assignment Class....104
Assignment Class Method....104
GenerateRule....104
Assignment Class Properties....105
Expression....105
Variable....105
Comparison Class....105
Comparison Class Method....106
GenerateRule....106
Comparison Class Properties....107
Operand1....107
Operand2....107
Type....107
Constant Class....108
Constant Class Method....109
GenerateRule....109
Constant Class Properties....109
Type....109
Value....109

Copyright © 1988-2007, Oracle. All rights reserved.

Constants Class....110
Cube Class....110
Cube Class Methods....110
AddIndex....110
GenerateRule....111
GetIndexes....111
Cube Class Property....112
Name....112
ExpressionBlock Class....112
ExpressionBlock Methods....112
AddRuleExpression....112
GetRuleExpressions....113
FunctionCall Class....113
FunctionCall Class Methods....117
AddArgument....117
GenerateRule....117
GetArguments....118
FunctionCall Class Properties....118
Name....118
Type....118
MemberReference Class....118
MemberReference Class Method....119
GenerateRule....119
MemberReference Class Properties....119
Dimension....119
Member....119
Operation Class....120
Operation Class Method....120
GenerateRule....120
Operation Class Properties....121
Operand1....121
Operand2....121
Type....121
Variable Class....122
Variable Class Method....122
GenerateRule....122
Variable Class Properties....123
Name....123
Type....123
Analytic Model Metadata Classes Examples....123

Copyright © 1988-2007, Oracle. All rights reserved.

Creating an Analytic Model Code Example....124

Chapter 4
Analytic Grid Classes....129
Understanding the Analytic Calculation Engine Classes....129
Using the Analytic Grid in PeopleCode....130
Using Freeze Column Mode....131
Error Handling....132
Data Types of the Analytic Grid Objects....132
Scope of Analytic Grid Objects....132
Analytic Grid Built-in Function....132
AnalyticGrid Class Methods....132
GetColumn....133
GetCubeCollection....133
LoadData....134
SetAnalyticInstance....134
SetLayout....135
AnalyticGrid Class Properties....135
InActive....135
Label....136
ShowGridLines....136
SlicerVisible....136

Chapter 5
Analytic Type Classes....137
Understanding the Analytic Calculation Engine Classes....137
Using the Analytic Type Classes....138
Error Handling....138
Data Types of the Analytic Type Objects....138
Scope of Analytic Type Objects....138
How to Import the Analytic Type Classes....139
How to Create an Analytic Type Class Object....139
Analytic Type Classes Constructor....140
AnalyticTypeDefn....140
AnalyticTypeDefn Class Methods....140
AddModel....140
AddRecord....141
Create....142

Copyright © 1988-2007, Oracle. All rights reserved.

CopyTo....142
Delete....143
DeleteModel....143
DeleteRecord....144
Get....144
GetModel....144
GetModelNames....145
GetRecord....145
GetRecordNames....146
Rename....146
Save....147
AnalyticTypeDefn Class Properties....147
AppClassPath....147
Comments....147
Description....148
Name....148
OwnerID....148
AnalyticTypeModelDefn Class Properties....148
Name....148
Type....148
AnalyticTypeRecordDefn Class Methods....149
GetSelectedField....149
SetSelectedField....149
UnsetSelectedField....150
AnalyticTypeRecordDefn Class Properties....150
Callback....150
Description....150
Name....151
Readable....151
ReadOnce....151
ScenarioManaged....151
SyncOrder....151
Writeable....151
AnalyticType Classes Example....152

Chapter 6
Application Classes....153
Understanding Application Classes....153
When Would You Use Application Classes?....154

Copyright © 1988-2007, Oracle. All rights reserved.

Application Classes General Structure....154
Class Name....155
Class Extension....156
Declaration of Public External Interface....156
Access Control and the Declaration of Protected Properties and Methods....157
Declaration of Private Instance Variables and Methods....160
Definition of Methods....162
Declaration of Abstract Methods and Properties....164
Interfaces....165
Constructors....166
External Function Declarations....168
Naming Standards....170
Naming Packages....170
Naming Classes....170
Naming Methods....170
Naming Properties....170
Import Declarations....170
Self-Reference....171
Using %This with Constructors....173
Properties and Constants....174
Differentiating Between Properties and Methods....174
Overriding Properties....175
Using Get and Set with Properties....176
Using Read-Only Properties....176
Using Methods and Properties for Collections....177
Declaring Constants....177
Using Variables in Application Classes....177
Placement of Variable Declarations....177
Declaring Private Instance Variables....178
Global Variables....178
Overriding Variables and Properties....178
Superclass Reference....178
Downcasting....179
Exception Handling....180
Commenting and Documenting Application Classes....181
Understanding Comments and Documentation....182
Method Header Comments....182
Class Header Comments....183
Designing Base Classes....183
Base Classes....184

Copyright © 1988-2007, Oracle. All rights reserved.

Abstract Base Classes.....184
Generic Base Classes.....184
Utility Classes.....184
Declaration of Application Classes.....185
Declaring Application Classes.....185
Importing Class Names.....185
Referencing Superclasses.....185
Declaring Private Methods.....185
Scope of an Application Class.....185
Application Classes Built-in Functions and Language Constructs.....186
Class.....186
Get.....187
Import.....188
Interface.....189
Method.....190
Set.....190

Chapter 7
API Repository.....193
Understanding the PeopleSoft API Repository.....193
The PeopleSoft API Repository.....194
Example of Accessing the Repository Using PeopleCode.....195
Example of Accessing the Repository by Using Visual Basic.....199
Repository Properties.....201
Bindings.....201
Namespaces.....201
Bindings Collection Properties.....201
Count.....201
Bindings Collection Methods.....202
Item.....202
Bindings Properties.....202
Name.....202
Bindings Methods.....202
Generate.....203
Namespaces Collection Properties.....203
Count.....203
Namespaces Collection Methods.....203
Item.....203
ItemByName.....204

Copyright © 1988-2007, Oracle. All rights reserved.

Namespaces Properties....204
Classes....204
Name....205
Namespaces Methods....205
CreateObject....205
ClassInfo Collection Properties....205
Count....205
ClassInfo Collection Methods....206
Item....206
ItemByName....206
ClassInfo Properties....207
Documentation....207
Methods....207
Name....207
Properties....208
MethodInfo Collection Properties....208
Count....208
MethodInfo Collection Methods....208
Item....208
ItemByName....209
MethodInfo Properties....209
Arguments....210
Documentation....210
Name....210
Type....210
PropertyInfo Collection Properties....210
Count....211
PropertyInfo Collection Methods....211
Item....211
ItemByName....212
PropertyInfo Properties....212
Documentation....212
Name....212
Type....212
Usage....213

Chapter 8
Array Class....215
Understanding Arrays....215

Copyright © 1988-2007, Oracle. All rights reserved.

Creating Arrays....215
Populating an Array....217
Removing Items From an Array....218
Creating Empty Arrays....218
Creating and Populating Multi-Dimensional Arrays....219
Using Flattening and Promotion....221
Declaring Array Objects....222
Understanding the Scope of an Array Object....222
Array Class Built-in Functions....222
Array Class Methods....223
Clone....223
Find....223
Get....224
Join....225
Next....227
Pop....228
Push....229
Replace....231
Reverse....232
Set....233
Shift....234
Sort....234
Subarray....236
Substitute....237
Unshift....238
Array Class Properties....238
Dimension....238
Len....239

Chapter 9
BPEL Classes....241
Understanding the BPEL Classes....241
Scope of the BPEL Classes....241
Data Types of the BPEL Classes....242
How to Import the BPEL Classes....242
How to Create a BPEL Object....242
BPEL Classes Constructors....243
AsyncFFSend....243
BPELUtil....243

Copyright © 1988-2007, Oracle. All rights reserved.

IBUtil....244
AsyncFFSend Class....244
AsyncFFSend Class Method....244
OnRequestSend....244
BPELUtil Class....245
BPELUtil Class Methods....245
GetASyncBPELProcessInstanceUrl....245
GetBPELProcessBrowserUrl....246
GetOperationType....246
GetSyncBPELProcessInstanceUrl....246
LaunchASyncBPELProcess....247
LaunchSyncBPELProcess....248
IBUtil Class....248
IBUtil Class Methods....248
GetBPELConsoleUrl....249
GetBPELDomain....249
GetMessageId....249
GetNode....250
GetNumberOfRoutings....250
GetStatus....251
GetTransactionId....251

Chapter 10
Business Interlink Class....253
Understanding Business Interlink Class....253
Using the Interlink Object....254
Deciding Which Methods to Use....254
Executing the Business Interlink Object....254
Supporting Batch Input and Output....255
Supporting Rowsets....255
Using the Flat Table Methods....255
Supporting Dynamic Output....255
Using Hierarchical Data (BIDocs)....256
Using the Incoming Business Interlink Methods and Properties....259
Understanding the State of an Interlink Object....260
Using Business Interlink with Application Engine....260
Using the Data Type of a Business Interlink Object....260
Understanding the Scope of a Business Interlink Object....260
Business Interlink Class Built-in Functions....261

Copyright © 1988-2007, Oracle. All rights reserved.

Business Interlink Class Methods....261
AddDoc....261
AddInputRow....262
AddNextDoc....264
AddValue....266
BulkExecute....268
Clear....272
Execute....273
FetchIntoRecord....275
FetchIntoRowset....277
FetchNextRow....280
GetCount....281
GetDoc....282
GetFieldCount....285
GetFieldType....286
GetFieldValue....287
GetInputDocs....288
GetNextDoc....289
GetOutputDocs....291
GetPreviousDoc....292
GetStatus....294
GetValue....295
InputRowset....297
MoveFirst....299
MoveNext....300
MoveToDoc....301
ResetCursor....302
Incoming Business Interlink Methods....303
AddAttribute....303
AddComment....304
AddProcessInstruction....305
AddText....306
CreateElement....307
GenXMLString....308
GetAttributeName....309
GetAttributeValue....310
GetNode....311
ParseXMLString....312
Incoming Business Interlink Properties....313
AttributeCount....313

Copyright © 1988-2007, Oracle. All rights reserved.

ChildNodeCount....313
NodeName....314
NodeType....315
NodeValue....316
Business Interlink Class Property....316
StopAtError....316
Configuration Parameters....317
Interlink Plug-in Configuration Parameters....317
URL Configuration Parameter....318
BIDocValidate Configuration Parameter....318

Chapter 11
Chart Class....319
Understanding the Chart Class....319
Chart Terms....322
Chart Types....322
2D Bar....323
2D Histogram....323
2D Horizontal Bar....324
2D Horizontal Stacked Bar....324
2D Horizontal Percent Bar....325
2D Line....325
2D Percent Bar....326
2D Pie....327
2D Scatter....327
2D Stacked Bar....328
3D Bar....328
3D Percent Bar....329
3D Pie....329
3D Stacked Bar....330
Gantt....330
Style Sheets and Charts....331
Considerations Using Style Sheets....331
Default Style Sheet Classes....332
Font Considerations for Extended Languages....332
Component Processor Considerations....333
Translation Considerations....333
Label Considerations....333
Bar Chart Considerations....333

Copyright © 1988-2007, Oracle. All rights reserved.

iScripts and Charts....334
Tick Mark Considerations....334
Bubble Chart Considerations....335
Line Chart Considerations....335
Data Series....336
Gantt Charts....338
Understanding Gantt Chart Terms....338
Using Gantt Charts in PeopleSoft Pure Internet Architecture....339
Creating Gantt Charts....340
Specifying DateTime Axis Formats....341
Working with Start and End Dates....342
Using Gantt Glyphs....342
Error Handling....343
Data Type of a Chart Object....343
Scope of a Chart Object....344
Chart Class Reference....344
Chart Class Built-in Functions....344
Chart Data Methods....345
Refresh....345
Reset....345
SetData....346
SetDataHints....347
SetDataSeries....350
SetDataURLs....351
SetDataXAxis....351
SetDataYAxis....352
Chart Data Properties....353
DataStartRow....353
DataWidth....353
ImageMap....354
IsTrueXY....354
Chart Appearance Properties....354
GridLines....355
GridLineType....355
Height....356
IsDrillable....356
IsPlainImage....357
LineType....357
RotationAngle....358
Style....358

Copyright © 1988-2007, Oracle. All rights reserved.

StyleSheet....358
Type....359
Width....359
Chart Colors....360
Chart Color Methods....361
SetColorArray....361
SetDataColor....362
Chart Axes Methods....362
SetXAxisLabels....362
SetYAxisLabels....363
Chart Axes Properties....364
XAxisCross....364
XAxisLabelOrient....365
XAxisStyle....365
XAxisTicks....366
YAxisLabelOrient....366
YAxisMax....366
YAxisMin....366
YAxisStyle....367
YAxisTicks....367
Chart Title Properties....367
MainTitle....367
MainTitleOrient....368
MainTitleStyle....368
XAxisTitle....368
XAxisTitleOrient....368
XAxisTitleStyle....369
YAxisTitle....369
YAxisTitleOrient....369
YAxisTitleStyle....369
Chart Legend Methods....369
SetLegend....370
Chart Legend Properties....370
HasLegend....371
LegendMaxEntries....371
LegendPosition....371
LegendStyle....371
Chart Overlay Methods....372
SetOLData....372
SetOLDataSeries....372

Copyright © 1988-2007, Oracle. All rights reserved.

SetOLDataXAxis....373
SetOLDataYAxis....373
Chart Overlay Properties....374
OLLineType....374
OLType....374
Scatter Chart Methods....374
SetDataAnnotations....375
SetDataGlyphScale....375
SetOLDataAnnotations....375
SetOLDataGlyphScale....376
Scatter Chart Property....376
ShowCrossHair....376
TrueXY Line Chart Properties....377
XAxisCrossPoint....377
XAxisMax....377
XAxisMin....377
XAxisPrecision....377
XAxisScaleResolution....378
YAxisCrossPoint....378
YAxisMax....379
YAxisMin....379
YAxisPrecision....379
YAxisScaleResolution....379
Gantt Chart Methods....380
SetActualEndDate....380
SetActualStartDate....380
SetActualTaskBarColor....381
SetChartArea....381
SetPlannedEndDate....382
SetPlannedStartDate....382
SetPlannedTaskBarColor....383
SetTableXScrollbar....383
SetTaskAppData....384
SetTaskAppDataTitles....384
SetTaskBarURL....385
SetTaskData....385
SetTaskDependencyChildID....386
SetTaskDependencyData....386
SetTaskDependencyType....387
SetTaskDependencyURL....388

Copyright © 1988-2007, Oracle. All rights reserved.

SetTaskDependencyParentID....388
SetTaskExpanded....389
SetTaskHints....389
SetTaskID....390
SetTaskLabel....390
SetTaskLevel....390
SetTaskMilestone....391
SetTaskName....392
SetTaskProgress....392
SetTaskProgressBarColor....393
SetWBSNumbering....393
Gantt Chart Properties....394
AxisEndDateTime....394
AxisStartDateTime....394
DataEndDateTime....395
DataStartDateTime....395
PixelsPerRow....395
ShowTaskLabels....395
TaskDependencyLineType....395
TaskMilestoneGlyph....396
TaskTitle....396
XAxisPosition....396
YAxisPosition....397
Gantt Chart DateTimeAxis Specific Methods....397
SetSecondFormat....397
SetMinuteFormat....398
SetHourFormat....398
SetDayFormat....399
SetMonthFormat....400
SetYearFormat....400
Chart Examples....401
Creating a Chart....401
Creating a Gantt Chart....403
Creating a Chart Using an iScript....407

Chapter 12
Component Interface Classes....411
Understanding Component Interface Class....411
Life Cycle of a Component Interface....412

Copyright © 1988-2007, Oracle. All rights reserved.

Setting Component Interface Keys....414
Standard and User-Defined Component Interface Methods....415
Accessing Component Interface Standard Properties....416
Accessing User-Defined Component Interface Properties....417
Data Type of a Component Interface Object....417
Scope of a Component Interface Object....418
Implementing a Component Interface....418
Component Interface Methods and Timeouts....419
Traversing a Component Interface and Using Data Collections....419
Working With Effective-Dated Data....421
How a Developer Uses GetEffectiveItem....421
Reusing Existing Code....422
Differences in Search Dialog Processing....422
Differences in PeopleCode Event and Function Behavior....422
Using %Menu Conditions....423
Session Class Methods....423
FindCompIntfcs....423
GetCompIntfc....424
Component Interface Collection Methods....424
First....424
Item....425
Next....425
Component Interface Collection Property....425
Count....426
Component Interface Class Methods....426
Cancel....426
CopyRowset....426
CopyRowsetDelta....428
CopySetupRowset....431
CopySetupRowsetDelta....433
Create....435
Find....436
Get....437
GetPropertyByName....437
Save....439
SetPropertyByName....439
Component Interface Class Properties....440
ComponentName....440
CreateKeyInfoCollection....441
EditHistoryItems....441

Copyright © 1988-2007, Oracle. All rights reserved.

FindKeyInfoCollection....441
GetDummyRows....442
GetHistoryItems....442
GetKeyInfoCollection....442
InteractiveMode....443
PropertyInfoCollection....443
StopOnFirstError....443
Data Collection Methods....443
CurrentItem....444
DeleteItem....444
GetEffectiveItem....446
GetEffectiveItemNum....446
InsertItem....447
Item....448
ItemByKeys....448
Data Collection Properties....449
Count....450
CurrentItemNumber....450
Data Item Class Property....450
ItemNum....450
Accessing the Structure of a Component Interface....451
CompIntfPropInfoCollection Collection....451
CompIntfPropInfoCollection Collection Methods....453
First....453
Item....453
Next....454
CompIntfPropInfoCollection Collection Properties....454
Count....454
CompIntfPropInfoCollection Object Properties....454
Format....454
IsCollection....455
IsReadOnly....455
Key....455
LabelLong....456
LabelShort....456
Length....456
Name....456
Prompt....457
PropertyInfoCollection....457
Required....457

Copyright © 1988-2007, Oracle. All rights reserved.

Type....457
Xlat....458
YesNo....458
Component Interface Examples....458
Create a New Instance of Data Example....458
Getting an Existing Instance of Data Example....461
Retrieving a List of Instance of Data Example....463
Inserting Effective-Dated Data Example....465
Inserting Effective-Dated Data Example Using Visual Basic....468
Inserting or Deleting a Row of Data Example....469

Chapter 13
Crypt Class....475
Understanding the Crypt Class....475
Creating a Crypt Object....475
Declaring Crypt Objects....475
Scope of a Crypt Object....476
Crypt Class Methods....476
FirstStep....476
GoToStep....476
LoadLibrary....477
NextStep....477
Open....478
SetParameter....478
UpdateData....479
Crypt Class Properties....479
Result....479
Verified....480

Chapter 14
Exception Class....481
Understanding Exception Class....481
How Exceptions Are Thrown....482
When to Use Exceptions....482
Try-Catch Sections....483
Data Type for Exception Class Objects....484
Scope of Exception Class Objects....484
Exception Class Built-in Functions....484

Copyright © 1988-2007, Oracle. All rights reserved.

Exception Class Methods.....485
GetSubstitution.....485
Output.....485
SetSubstitution.....486
ToString.....486
Exception Class Properties.....487
Context.....487
DefaultText.....487
MessageNumber.....488
MessageSetNumber.....488
MessageSeverity.....488
StackTrace.....488
SubstitutionCount.....489

Chapter 15
Field Class.....491
Understanding the Field Class.....491
Considerations Using User Interface Properties.....491
Shortcut Considerations.....491
Data Type for a Field Object.....492
Scope of a Field Object.....492
Field Class Built-in Function.....492
Field Class Methods.....492
AddDropDownItem.....493
ClearDropDownList.....494
GetAuxFlag.....494
GetLongLabel.....495
GetRelated.....496
GetShortLabel.....497
SearchClear.....498
SetCursorPos.....498
SetDefault.....499
Field Class Properties.....500
DataAreaCollapsed.....500
DecimalPosition.....501
DisplayFormat.....501
DisplayOnly.....502
DisplayZero.....503
DisplayZeroChanged.....503

Copyright © 1988-2007, Oracle. All rights reserved.

EditError....504
Enabled....504
FieldLength....504
FormatLength....505
FormattedValue....505
IsAltKey....506
IsAuditFieldAdd....506
IsAuditFieldChg....506
IsAuditFieldDel....506
IsAutoUpdate....506
IsChanged....507
IsDateRangeEdit....507
IsDescKey....507
IsDuplKey....507
IsEditTable....507
IsEditXlat....508
IsFromSearchField....508
IsInBuf....508
IsKey....508
IsListItem....509
IsNotUsed....509
IsRequired....509
IsSearchItem....509
IsSystem....510
IsThroughSearchField....510
IsUseDefaultLabel....510
IsYesNo....510
Label....510
LabelImage....511
LongTranslateValue....512
MessageNumber....512
MessageSetNumber....513
Name....513
OriginalValue....513
ParentRecord....514
PromptTableName....514
SearchDefault....514
SearchEdit....515
SetComponentChanged....515
ShortTranslateValue....516

Copyright © 1988-2007, Oracle. All rights reserved.

ShowRequiredFieldCue……517
SmartZero……517
SqlText……517
StoredFormat……518
Style……518
SystemIDFieldName……519
TimeStampFieldName……520
Type……520
Value……521
Visible……522

Chapter 16
File Class……523
Understanding File Layout……523
Data Type of a File Object……524
Scope of a File Object……524
File Security Considerations……525
File Access Interruption Recovery……525
Plain Text Files……526
Automatic PeopleCode Generation……527
End Of Line Considerations……528
File Layout Error Processing……528
File Class Built-in Functions……529
File Class Methods……529
Close……530
CreateRowset……530
Delete……531
GetPosition……532
GetString……533
Open……534
ReadLine……537
ReadRowset……538
SetFileId……540
SetFileLayout……542
SetPosition……543
SetRecTerminator……544
WriteLine……545
WriteRaw……546
WriteRecord……546

Copyright © 1988-2007, Oracle. All rights reserved.

WriteRowset....548
WriteString....550
File Class Properties....551
CurrentRecord....551
IgnoreInvalidId....551
IsError....552
IsNewFileId....552
IsOpen....553
Name....553
UseSpaceForNull....554
File Layout Examples....557
WriteRecord Example....558
ReadRecord Example....560
WriteRowset Example....561
ReadRowset Example....565
File Rowset Considerations....565
Application Engine Example....566
Multiple File Layouts....569
Reading Multiple File Layouts....569
Writing Multiple File Layouts....571

Chapter 17
Grid Classes....575
Understanding Grid and GridColumn Classes....575
Shortcut Considerations....576
The Grid Class in PeopleCode....576
Data Type for a Grid or Grid Column Object....577
Scope of a Grid or Grid Column Object....577
Grid Class Built-in Function....578
Grid Class Method....578
GetColumn....578
Grid Class Properties....580
gridcolumn....580
Label....580
GridColumn Class....580
GridColumn Class Properties....581
Enabled....581
Label....581
Name....582

Copyright © 1988-2007, Oracle. All rights reserved.

Visible....582

Chapter 18
Internet Script Classes (iScript)....585
Understanding Internet Script Classes....585
URL vs. URI....587
Web Libraries....587
iScript Security....589
When to Use an iScript....589
Style Sheets and Styles....590
Other Considerations....590
Details of an iScript URL....590
The Generate Functions....591
Error Handling....593
Scope of the Internet Script Classes....593
Data Types of the iScript Classes....594
Internet Script Classes....594
Request Class....594
Response....595
Cookies....595
Internet Script Classes Built-in Functions....595
Request Class Methods....596
GetContentBody....596
GetCookieNames....597
GetCookieValue....597
GetHeader....597
GetHeaderNames....598
GetHelpURL....598
GetParameter....598
GetParameterNames....599
GetParameterValues....599
Request Class Properties....599
AuthTokenDomain....599
AuthType....600
ByPassSignOn....600
BrowserPlatform....600
BrowserType....600
BrowserVersion....600
ContentURI....600

Copyright © 1988-2007, Oracle. All rights reserved.

ExpireMeta....601
FullURI....601
HTTPMethod....602
LogoutURL....602
PathInfo....602
Protocol....602
QueryString....602
RelativeURL....603
RemoteAddr....603
RemoteHost....603
RequestURI....603
RemoteUser....604
Scheme....604
ServerName....604
ServerPort....605
ServletPath....605
Timeout....605
Response Class....605
Response Class Methods....605
Clear....606
CreateCookie....606
GetCookie....606
GetCookieNames....606
GetHeader....607
GetHeaderNames....607
GetImageURL....607
GetJavaScriptURL....608
GetStyleSheetURL....608
RedirectURL....609
SetContentType....609
SetHeader....609
UseSimpleURL....610
Write....610
WriteLine....611
Response Class Properties....611
Charset....611
DefaultStyleSheetName....611
Cookie Class....612
Cookie Class Properties....612
Domain....612

Copyright © 1988-2007, Oracle. All rights reserved.

MaxAge....612
Name....612
Path....613
Secure....613
Value....613

Chapter 19
Java Class....615
Understanding Java Class....615
Supported Versions of Java....616
Naming Standards for Classes and Packages....616
System Setup for Java Classes....616
From PeopleCode to Java....617
State Management Concerns....618
CreateJavaObject Example....619
CreateJavaArray Example....619
GetJavaClass Example....620
From Java to PeopleCode....620
SysVar Java Class....620
SysCon Java Class....620
Func Java Class....620
Name Java Class....621
Accessing PeopleCode Objects....621
Using Application Classes From Java to PeopleCode....623
PeopleCode and Java Data Types Mapping....624
Considerations When Using the PeopleCode Java Functions....625
Copying Arrays of Data Between PeopleCode and Java....626
Considerations Working with the Java Garbage Collector....626
Error Handling and the PeopleCode Java Functions....627
The Java Debugging Environment....627
Data Type of a Java Object....628
Scope of a Java Object....628
PeopleCode Java Built-in Functions....628

Chapter 20
Mail Classes....629
Understanding PeopleSoft MultiChannel Framework Mail Classes....629
PeopleSoft MultiChannel Framework Mail Classes....629

Copyright © 1988-2007, Oracle. All rights reserved.

Scope of the Mail Classes....630
Data Types of the Mail Classes....630
Importing Mail Classes....630
Creating a Mail Object....631
Using the Mail Classes....631
General Lifecycle of an Outbound Email....631
Delivery Status Notification and Return Receipts....632
Priority....633
MCFBodyPart and MCFEmail Considerations....633
Retrieving Email From a Mail Server With the MCFGetMail Class....635
Structure of Rowset Used for Email Information....635
Error Messages Returned by MCFGetMail Class Methods....642
Mail Classes Built-in Functions....643
Mail Classes Constructors....643
MCFBodyPart....643
MCFGetMail....643
MCFOutboundEmail....643
MCFMailStore....644
SMTPSession....644
MCFBodyPart Class....644
MCFBodyPart Class Methods....644
AddHeader....644
GetHeader....645
GetHeaderCount....646
GetHeaderName....646
GetHeaderNames....647
GetHeaderValues....647
GetUnparsedHeaders....648
SetAttachmentContent....648
MCFBodyPart Class Properties....650
AttachmentURL....650
Charset....650
ContentType....650
Description....650
Disposition....651
Filename....651
FilePath....651
FilePathType....652
IsAttachment....652
MultiPart....652

Copyright © 1988-2007, Oracle. All rights reserved.

Text....653
MCFEmail Class....653
MCFEmail Class Properties....653
BCC....653
BounceTo....653
CC....654
From....654
Importance....654
Priority....654
Recipients....655
RefIDs....655
ReplyIDs....655
ReplyTo....655
Sender....656
Sensitivity....656
Subject....656
Text....656
MCFGetMail Class Methods....656
CreateQuarantineFolder....656
GetCount....657
GetEmailCount....657
ReadAllEmailHeadersWithAttach....658
ReadEmails....659
ReadEmailsWithAttach....660
ReadEmailsWithUID....661
ReadEmailWithAttach....661
ReadHeaders....662
RemoveEmail....663
RemoveEmails....664
SetMCFEmail....664
MCFGetMail Class Properties....665
AttachmentRoot....665
ContentTypes....666
ErrorCount....666
IBNode....666
MailServer....666
Password....667
QuarantineCount....667
QuarantineFolder....667
Status....668

Copyright © 1988-2007, Oracle. All rights reserved.

UserID....668
MCFInboundEmail Class Methods....668
DumpToFile....668
GetAttachments....670
GetFrom....670
GetParts....670
GetSender....671
ReadFromDatabase....671
SaveToDatabase....672
MCFInboundEmail Class Properties....672
AttachList....672
AttachSizes....672
DttmReceived....673
DttmSaved....673
DttmSent....673
IBNode....674
Language....674
MessageID....674
NotifyCC....674
NotifyTo....674
OffsetReceived....675
OffsetSent....675
Server....675
Size....675
Status....675
UID....676
User....676
MCFMailStore Class....676
MCFMailStore Import Statements....676
MCFMailStore Methods....676
AuthorizeEmailAttach....676
DeleteEmail....677
RetrieveEmail....678
StoreEmail....679
MCFMailUtil Class....680
MCFMailUtil Class Methods....680
DecodeText....680
DecodeWord....681
EncodeText....682
EncodeWord....683

Copyright © 1988-2007, Oracle. All rights reserved.

IsEmailServerAvailable....684
ValidateAddress....685
MCFMultiPart Class....686
MCFMultiPart Class Methods....686
AddBodyPart....686
GetBodyPart....687
GetContentType....687
GetCount....687
MCFMultiPart Class Property....688
SubType....688
MCFOutboundEmail Class....688
MCFOutboundEmail Class Methods....688
AddAttachment....688
AddHeader....689
GetHeader....690
GetHeaderCount....691
GetHeaderName....691
GetHeaderNames....692
GetHeaderValues....692
Send....693
SetSMTPParam....693
MCFOutboundEmail Class Properties....694
BackupSMTPPort....694
BackupSMTPServer....695
BackupSMTPUserName....695
BackupSMTPUserPassword....695
BCC....695
BounceTo....695
CC....696
Charset....696
ContentLanguage....696
ContentType....696
DefaultCharSet....697
Description....697
Disposition....697
ErrorDescription....697
ErrorDetails....698
Filename....698
FilePath....698
FilePathType....698

Copyright © 1988-2007, Oracle. All rights reserved.

From....699
Importance....699
InvalidAddresses....699
IsAuthenticationReqd....700
IsOkToSendPartial....700
IsReturnReceiptReqd....700
MessageNumber....700
MessageSetNumber....701
MultiPart....701
Priority....701
Recipients....702
RefIDs....702
ReplyIDs....702
ReplyTo....702
ResultOfSend....702
Sender....703
Sensitivity....703
SMTPPort....703
SMTPServer....703
SMTPUserName....704
SMTPUserPassword....704
StatusNotifyOptions....704
StatusNotifyReturn....704
Subject....705
Text....705
TimeToWaitForResult....705
UsedDefaultConfig....705
UsedPrimaryServer....706
ValidSentAddresses....706
ValidUnsentAddresses....706
SMTPSession Class....706
SMTPSession Class Methods....706
Send....707
SendAll....707
SetSMTPParam....708
SMTPSession Class Properties....709
BackupPort....709
BackupServer....709
BackupUserName....709
BackupUserPassword....709

Copyright © 1988-2007, Oracle. All rights reserved.

IsAuthenticationReqd....709
Port....710
Server....710
UsedDefaultConfig....710
UsedPrimaryServer....710
UserName....711
UserPassword....711
Mail Classes Examples....711
Creating Text Email....711
Creating Email and Overriding SMTP Settings....712
Creating HTML Email....714
Creating Multi-Part Email With Both Text and HTML Parts....715
Creating HTML Email with Images....716
Creating Email with Attachments....718
Creating Email Attachments Specifying a URL....719
Creating Multiple Emails....720
Authenticating Email While Sending....721

Chapter 21
MCFIMInfo Class....723
Understanding the MCFIMInfo Class....723
Using the MCFIMInfo Class....723
Data Type for MCFIMInfo Objects....724
Scope of MCFIMInfo Objects....724
MCFIMInfo Class Built-in Function....724
MCFIMInfo Class Methods....724
AddUser....724
CheckAll....725
GetAdditionalUserInfo....725
GetErrorImageName....725
GetOffLineImageName....726
GetOnlineImageName....726
GetUnknownImageName....726
GetLaunchURL....727
GetStatus....727
RemoveUser....728

Copyright © 1988-2007, Oracle. All rights reserved.

Chapter 22
Message Classes.....729
Understanding Message Classes.....729
Messages, Service Operations and Handlers.....730
Integration Broker Application Classes.....730
Data Type of Message Objects.....731
Scope of Message Objects.....731
Message Object Population.....731
Considerations When Populating a Rowset From a Message.....733
Considerations for Publishing and Subscribing to Partial Records.....735
Considerations When Subscribing to Character Fields.....735
Message Segments.....735
Error Handling.....738
Message Class Built-In Functions.....739
Message Class Methods.....739
Clone.....739
CopyPartRowset.....740
CopyRowset.....741
CopyRowsetDelta.....742
CopyRowsetDeltaOriginal.....744
CreateNextSegment.....747
DeleteSegment.....748
ExecuteEdits.....748
GenXMLPartString.....751
GenXMLString.....751
GetContentString.....752
GetPartAliasName.....752
GetPartName.....753
GetPartRowset.....753
GetPartVersion.....754
GetPartXMLDoc.....754
GetQueryString.....755
GetRowset.....755
GetSegment.....757
GetXmlDoc.....757
InBoundPublish.....758
LoadXMLPartString.....758
LoadXMLString.....759
Publish.....760
SegmentRestart.....762

Copyright © 1988-2007, Oracle. All rights reserved.

SetEditTable....762
SetQueryString....763
SetStatus....764
SetXmlDoc....765
SyncRequest....766
Update....767
UpdateSegment....768
Message Class Properties....769
ActionName....769
AliasName....769
ChannelName....769
Cookies....770
CurrentSegment....771
DefaultMessageVersion....771
DoNotPubToNodeName....772
GUID....772
IBException....772
IBInfo....773
IsActive....773
IsDelta....774
IsEditError....774
IsEmpty....775
IsLocal....775
IsOperationActive....776
IsParts....776
IsPartsStructured....776
IsRequest....777
IsSourceNodeExternal....777
IsStructure....777
MessageDetail....777
Name....778
NRId....778
OperationName....778
OperationVersion....778
ParentTransactionId....778
PartCount....779
PubID....779
PubNodeName....779
QueueName....780
QueueSeqId....780

Copyright © 1988-2007, Oracle. All rights reserved.

ResponseStatus....780
SegmentCount....780
SegmentsByDatabase....780
Size....781
SubName....782
SubQueueName....782
SubscriptionProcessId....782
TransactionId....783
Version....783
IntBroker Class....783
IntBroker Class Methods....784
Cancel....784
ConnectorRequest....785
ConnectorRequestUrl....786
DeleteOrphanedSegments....787
GetArchData....787
GetArchIBInfoData....788
GetIBInfoData....788
GetMessage....789
GetMessageErrors....790
GetMsgSchema....791
GetSyncIBInfoData....791
GetSyncLogData....792
InBoundPublish....793
IsOperationActive....794
MsgSchemaExists....794
Publish....795
Resubmit....796
SetErrorMessage....797
SetStatus....798
SyncRequest....799
Update....801
UpdateXmlDoc....801
IBInfo Class....802
IBInfo Class Methods....802
AddAttachment....802
ClearAttachments....803
DeleteAttachment....804
GetAttachmentContentID....804
GetAttachmentProperty....805

Copyright © 1988-2007, Oracle. All rights reserved.

LoadConnectorProp……807
LoadConnectorPropFromNode……807
LoadConnectorPropFromRouting……808
LoadConnectorPropFromTrx……808
SetAttachmentProperty……809
IBInfo Class Properties……810
AppServerDomain……810
IBConnectorInfo……810
ConnectorOverride……811
ConversationID……811
Cookies……811
DestinationNode……812
ExternalMessageID……812
ExternalOperationName……812
ExternalUserName……812
ExternalUserPassword……813
FinalDestinationNode……813
HTTPSessionId……813
InReplyToID……813
MessageChannel……813
MessageName……814
MessageQueue……814
MessageType……814
MessageVersion……814
NodeDN……815
NonRepudiationID……815
NumberOfAttachments……815
OperationType……815
OperationVersion……815
OrigNode……816
OrigProcess……816
OrigTimeStamp……816
OrigUser……816
PublicationID……816
RequestingNodeName……817
RequestingNodeDescription……817
ResponseAsAttachment……817
SegmentsUnOrder……817
SourceNode……817
SyncServiceTimeout……818

Copyright © 1988-2007, Oracle. All rights reserved.

TransactionID....818
UserName....818
VisitedNodes....819
WSA_MessageID....819
WSA_ReplyTo....819
ConnectorInfo Collection....819
ConnectorInfo Collection Methods....819
AddConnectorProperties....819
AddQueryStringArg....820
ClearConnectorProperties....821
ClearQueryStringArgs....821
DeleteConnectorProperties....822
DeleteQueryStringArg....822
GetConnectorPropertiesName....823
GetConnectorPropertiesType....823
GetConnectorPropertiesValue....823
GetNumberOfConnectorProperties....824
GetNumberOfQueryStringArgs....824
GetQueryStringArgName....824
GetQueryStringArgValue....825
ConnectorInfo Collection Properties....825
ConnectorClassName....825
ConnectorName....826
Cookies....826
PathInfo....826
RemoteFrameworkURL....826

Chapter 23
Mobile Classes....827
Mobile Classes Overview....827
Understanding the Mobile Device and the Component Interface....828
Mobile Class Hierarchy....831
Debugging in Mobile....835
Debug Settings....836
Trace Settings....837
Error Handling....837
Data Types for Mobile....838
Scope of Mobile Objects....838
Variables in Mobile PeopleCode....838

Copyright © 1988-2007, Oracle. All rights reserved.

Mobile Functions....838
Session Class Method....839
GetCompIntfc....839
Mobile Class....840
Mobile Class Methods....840
Create....840
Find....841
Get....842
GetName....842
GetParent....843
GetPeerDefaultAttributesByName....843
GetPropertyAttrsByName....844
GetPropertyByName....845
GetPropertyInfoByName....845
GetTopParent....846
IsNew....846
IsNewAndNeverSaved....847
IsSameAs....847
ReadBlobFromFile....848
Save....849
SetModified....850
SetPropertyByName....851
ValidateEnum....851
ValuesDifferFromLastSync....852
WasSaved....852
WriteBlobToFile....853
Mobile Class Property....854
PropertyInfoCollection....854
PropertyInfoCollection....854
PropertyInfoCollection Method....854
Item....854
PropertyInfoCollection Property....855
Count....855
PropertyInfo Class....856
PropertyInfo Class Properties....856
DefaultValue....856
HasDefaultValue....856
IsCollection....857
IsRefToParent....857
IsPeerReference....857

Copyright © 1988-2007, Oracle. All rights reserved.

IsSimpleApplicationProperty....858
IsSystemProperty....858
Name....858
PropertyInfoCollection....859
CI Collection Class....859
CI Collection Method....860
Item....860
CI Collection Properties....860
Count....860
DataCollection Class....861
Data Collection Methods....861
GetListViewAttrs....861
Item....862
DeleteItem....863
InsertItem....864
Data Collection Properties....865
Count....865
ListViewAttrs Class....866
ListViewAttrs Method....866
IsSet....866
ListViewAttrs Properties....867
DetailViewLabel....867
Label....867
ListViewLabel....868
Visible....868
PropertyAttrs Class....869
PropertyAttrs Method....869
IsSet....869
PropertyAttrs Properties....870
DisplayOnly....870
InError....871
Label....871
ShowRequiredCue....871
Visible....872
PeerDefaultAttributes Class....873
PeerDefaultAttributes Class Method....873
IsSet....873
PeerDefaultAttributes Properties....874
DisplayOnly....874
Label....874

Copyright © 1988-2007, Oracle. All rights reserved.

Visible....875
Classes and Functions and System Variables Used in Mobile....875
PeopleCode Classes Used in Mobile....875
System Variables Used in Mobile....876
Built-In Functions Used in Mobile....876

Chapter 24
Notification Classes....879
Understanding Notification Classes....879
Scope of the Notification Classes....879
Data Types of the Notification Classes....880
How to Import the Notification Classes....880
How to Create a Notification Object....881
Notification Classes Constructors....881
Notification....881
NotificationAddress....882
NotificationTemplate....883
Worklist....883
WorklistEntry....884
WSWorklistEntry....885
Notification Class....885
Notification Class Import Statements....885
Notification Class Method....886
Send....886
Notification Class Properties....886
ContentType....886
dttmCreated....886
EmailReplyTo....887
FileNames....887
FileTitles....887
language_cd....887
Message....887
NotifyBCC....888
NotifyCC....888
NotifyFrom....888
NotifyGuid....888
NotifyTo....888
SourceComponent....889
SourceMarket....889

Copyright © 1988-2007, Oracle. All rights reserved.

SourceMenu....889
Subject....889
Template....889
NotificationAddress Class....890
NotificationAddress Class Properties....890
Channel....890
Description....890
EmailId....890
Language....890
Oprid....891
NotificationTemplate Class....891
NotificationTemplate Class Methods....891
GetAndExpandTemplate....891
SetupCompVarsAndRcpts....892
SetupGenericVars....893
NotificationTemplate Class Properties....893
ComponentId....894
Instruction....894
Language....894
Market....894
Priority....894
Responses....894
Subject....895
TemplateId....895
TemplateType....895
Text....895
Worklist Class....895
Worklist Class Method....896
Reassign....896
WorklistEntry Class....896
WorklistEntry Class Methods....896
Create....897
GetResponseStatus....898
Reassign....898
Save....899
SaveWithCustomData....900
SelectByKey....901
SelectByMessageId....902
Update....902
WorklistEntry Class Properties....903

Copyright © 1988-2007, Oracle. All rights reserved.

actiondttm....903
busactivity....903
buseventname....903
busprocname....903
commentshort....903
do_replicate_flag....904
instanceid....904
instselecteddttm....904
inststatus....904
insttimeoutddttm....905
instworkeddttm....905
IsCreatedViaWebService....905
oprid....905
originatorid....905
prevoprid....905
requestmessageid....906
ResponseStatus....906
syncid....906
timedout....906
transactionid....907
url....907
wl_priority....907
wldaystoselect....907
wldaystowork....907
worklistname....908
WSWorklistEntry Class....908
WSWorklistEntry Class Methods....908
OnError....908
OnNotify....909
WSWorklistEntry Class Properties....909
InstanceID....909
TransactionID....909
Notification Classes Examples....910
Creating a WorklistEntry....910
Updating a WorklistEntry....911

Chapter 25
Optimization PeopleCode....915
Using Optimization PeopleCode on the Application Server....915

Copyright © 1988-2007, Oracle. All rights reserved.

Using Optimization PeopleCode in an Application Engine Program....916
Performing Optimization in PeopleCode....916
Creating New Analytic Instances....916
Loading Analytic Instances Into an Analytic Server....917
Running Optimization Transactions....918
Invoking the Optimization PeopleCode Plug-In....919
Shutting Down Optimization Engines....919
Deleting Existing Analytic Instances....919
Programming for Database Updates....920
Using Lights-Out Mode with Optimization....920
Understanding Lights-out Mode....921
Creating a Request Message....922
Creating a Response Message....926
Editing the Request PeopleCode....927
Editing the Response PeopleCode....931
Optimization Built-in Functions....933
CreateOptEngine....933
CreateOptInterface....935
DeleteOptProbInst....936
GetOptEngine....937
GetOptProbInstList....938
InsertOptProbInst....939
IsValidOptProbInst....941
OptEngine Class Methods....942
CheckOptEngineStatus....942
FillRowset....943
GetDate....945
GetDateArray....945
GetDateTime....946
GetDateTimeArray....947
GetNumber....948
GetNumberArray....949
GetString....950
GetStringArray....950
GetTime....952
GetTimeArray....952
GetTraceLevel....953
RunAsynch....954
RunSynch....956
SetTraceLevel....957

Copyright © 1988-2007, Oracle. All rights reserved.

ShutDown....959
OptEngine Class Properties....960
DetailMsgs....960
DetailedStatus....961
OptBase Application Class....962
OptBase Class Methods....963
GetParmDate....963
GetParmDateArray....964
GetParmDateTime....964
GetParmDateTimeArray....964
GetParmNumber....965
GetParmNumberArray....965
GetParmInt....966
GetParmIntArray....966
GetParmString....967
GetParmStringArray....967
GetParmTime....968
GetParmTimeArray....968
Init....969
OptDeleteCallback....969
OptInsertCallback....970
OptPostUpdateCallback....970
OptPreUpdateCallback....971
OptRefreshCallback....972
SetOutputParmDate....972
SetOutputParmDateArray....973
SetOutputParmDateTime....973
SetOutputParmDateTimeArray....974
SetOutputParmNumber....974
SetOutputParmNumberArray....975
SetOutputParmInt....975
SetOutputParmIntArray....976
SetOutputParmString....976
SetOutputParmStringArray....977
SetOutputParmTime....977
SetOutputParmTimeArray....978
OptInterface Class Methods....978
ActivateModel....979
ActivateObjective....979
DeactivateModel....980

Copyright © 1988-2007, Oracle. All rights reserved.

DumpMsgToLog....981
FindRowNum....981
GetSolution....982
GetSolutionDetail....984
IsModelActive....985
RestoreBounds....985
SetVariableBounds....986
SetVariableType....988
Solve....989

Chapter 26
Page Class....991
Understanding Page Class....991
Shortcut Considerations....991
Data Type of a Page Object....992
Scope of a Page Object....992
Page Class Built-in Function....992
Page Class Properties....992
CopyURLLink....992
CustomizePageLink....993
DisplayOnly....993
HelpLink....994
Name....994
NewWindowLink....994
Visible....995

Chapter 27
PortalRegistry Classes....997
Understanding Portal Registry....997
Folders....998
Content References....998
Nodes....999
Security....999
Attributes....1000
Additional Portal Objects....1000
Using the PortalRegistry API....1000
Using Security....1001
Using Object Properties....1001

Copyright © 1988-2007, Oracle. All rights reserved.

Accessing Objects....1001
Using PermissionValue, RolePermissionValue, Cascading Permissions and CascadingRolePermissions....1002
Working With ValidFrom and ValidTo....1004
Doing Error Handling....1004
Understanding the Life-Cycle of a PortalRegistry Object....1005
Viewing the PortalRegistry Class Hierarchy....1008
Using Content References....1013
UsageType....1013
TemplateType....1015
URLType....1015
Nodes and URL....1016
Naming Conventions....1017
Deleting Content Considerations....1017
Saving Content Considerations....1018
Data Type of a PortalRegistry Object....1018
Scope of a PortalRegistry Object....1019
PortalRegistry Reference....1019
Session Class Methods....1019
FindPortalRegistries....1020
GetLocalNode....1020
GetNodes....1021
GetPortalRegistry....1021
GetRemoteNodes....1022
PortalRegistry Class....1022
PortalRegistry Class Methods....1023
Close....1023
CopyObject....1023
Create....1024
CreateContentRefLink....1025
CreateRemote....1026
Delete....1027
DeleteHomepage....1028
FindCRefByName....1028
FindCRefByURL....1029
FindCRefForURL....1030
FindCRefLinkByName....1031
FindFolderByName....1032
FindPgItByName....1033
GetAbsoluteContentURL....1033
GetDefaultHPTabOID....1034

Copyright © 1988-2007, Oracle. All rights reserved.

GetQualifiedURL....1034
GrantPermissionForComponent....1035
GrantPermissionForScript....1036
Open....1037
PermissionListDelete....1037
PermissionListSaveAs....1038
RevokePermissionForComponent....1038
RevokePermissionForScript....1039
Save....1040
PortalRegistry Class Properties....1041
DefaultTemplate....1041
Description....1041
Favorites....1042
FolderNavObject....1042
Homepage....1042
IsFolderNavigation....1042
Name....1042
NodeTemplates....1043
OwnerId....1043
PageletCategories....1043
Portals....1043
RootFolder....1043
TabDefinitions....1044
TemplateObject....1044
PortalRegistry Collection....1044
PortalRegistry Collection Methods....1044
First....1044
Item....1045
Next....1045
PortalRegistry Collection Property....1045
Count....1045
Node Class....1046
Node Class Properties....1046
ActiveNode....1046
AppsRelease....1046
ContentURI....1047
DefaultPortalName....1047
Description....1047
IsDefault....1047
IsLocal....1047

Copyright © 1988-2007, Oracle. All rights reserved.

Name....1048
NodePassword....1048
NodeType....1048
PortalURI....1048
ToolsRelease....1049
Node Collection....1049
Node Collection Methods....1049
First....1049
ItemByName....1049
Next....1050
Node Collection Property....1050
Count....1050
RemoteNode Collection....1051
RemoteNode Collection Methods....1051
First....1051
ItemByName....1051
Next....1052
RemoteNode Collection Property....1052
Count....1052
Portal Class....1052
Portal Class Method....1053
Save....1053
Portal Class Properties....1053
HostNodeName....1053
IsLocal....1053
Name....1054
Portal Collection....1054
Portal Collection Methods....1054
First....1054
ItemByName....1054
Next....1055
Portal Collection Properties....1055
Count....1055
NodeTemplate Class....1056
NodeTemplate Class Properties....1056
DefaultTemplate....1056
Name....1056
TemplateObject....1056
NodeTemplate Collection....1057
NodeTemplate Collection Methods....1057

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteItem....1057
First....1057
InsertItem....1058
ItemByName....1058
Next....1059
NodeTemplate Collection Properties....1059
Count....1059
Folder Class....1059
Folder Class Method....1060
Save....1060
Folder Class Properties....1061
Attributes....1061
Author....1061
AuthorAccess....1061
Authorized....1061
CascadedPermissions....1062
CascadedRolePermissions....1062
ContentRefs....1062
CreationDate....1063
Description....1063
Folders....1063
IsMobile....1063
IsVisible....1063
Label....1064
Name....1064
OwnerId....1064
ParentName....1064
Path....1065
Permissions....1065
Product....1065
RolePermissions....1065
SequenceNumber....1066
PublicAccess....1066
ValidFrom....1066
ValidTo....1066
Folder Collection....1066
Folder Collection Methods....1067
DeleteItem....1067
First....1067
InsertItem....1068

Copyright © 1988-2007, Oracle. All rights reserved.

ItemByName....1068
Next....1069
Folder Collection Property....1070
Count....1070
Content Reference Class....1070
Content Reference Class Methods....1070
CreateLink....1070
Save....1071
Content Reference Class Properties....1072
AbsoluteContentURL....1072
AbsolutePortalURL....1072
AssignedPagelets....1072
Attributes....1073
Author....1073
AuthorAccess....1073
Authorized....1073
CascadedPermissions....1074
CascadedRolePermissions....1074
ContentProvider....1074
CreationDate....1075
Data....1075
Description....1075
HtmlText....1075
IsMobile....1075
IsVisible....1076
Label....1076
Links....1076
Name....1076
OwnerId....1077
ParentName....1077
Path....1077
Permissions....1077
Product....1078
PublicAccess....1078
QualifiedURL....1078
RelativeURL....1078
RolePermissions....1079
SequenceNumber....1079
StorageType....1079
Template....1080

Copyright © 1988-2007, Oracle. All rights reserved.

TemplateObject....1081
TemplateType....1081
URL....1081
URLType....1082
UsageType....1083
ValidFrom....1083
ValidTo....1083
Content Reference Collection....1084
Content Reference Collection Methods....1084
DeleteItem....1084
First....1085
InsertItem....1085
ItemByName....1086
Next....1086
Content Reference Collection Property....1087
Count....1087
AttributeValue Class....1087
AttributeValue Class Properties....1087
Label....1087
Name....1088
Translatable....1088
Value....1089
Attribute Collection....1089
Attribute Collection Methods....1089
DeleteItem....1089
First....1090
InsertItem....1090
ItemByName....1091
Next....1092
Attribute Collection Property....1092
Count....1092
PermissionValue Class....1092
PermissionValue Class Properties....1093
Cascade....1093
Name....1093
PermType....1093
PermissionValue Collection....1094
PermissionValue Collection Methods....1094
DeleteItem....1094
First....1095

Copyright © 1988-2007, Oracle. All rights reserved.

InsertItem....1095
ItemByName....1096
Next....1096
PermissionValue Collection Property....1097
Count....1097
RolePermissionValue Collection....1097
RolePermissionValue Collection Methods....1098
DeleteItem....1098
First....1098
InsertItem....1099
ItemByName....1099
Next....1100
RolePermissionValue Collection Property....1100
Count....1100
Content Reference Links....1101
Content Reference Link Methods....1101
Delete....1101
Save....1102
Content Reference Link Properties....1102
AbsoluteContentURL....1102
AbsolutePortalURL....1103
Attributes....1103
Author....1103
AuthorAccess....1103
Authorized....1103
CascadedPermissions....1104
CascadedRolePermissions....1104
ContentProvider....1105
CreationDate....1105
Data....1105
Description....1105
IsMobile....1105
IsVisible....1106
Label....1106
Name....1106
OwnerId....1106
ParentName....1107
Path....1107
Permissions....1107
Product....1107

Copyright © 1988-2007, Oracle. All rights reserved.

PublicAccess....1108
RelativeURL....1108
RolePermissions....1108
SequenceNumber....1108
Template....1109
TemplateObject....1109
TemplateType....1109
URL....1110
URLType....1110
UsageType....1110
ValidFrom....1111
ValidTo....1111
Link Collection....1111
Link Collection Methods....1111
First....1111
Next....1112
Link Collection Property....1112
Count....1112
Link Class....1112
Link Properties....1112
LinksObjectName....1113
LinksObjectType....1113
LinksPortalName....1113
TabDefinition Class....1113
TabDefinition Class Methods....1113
Save....1113
TabDefinition Class Properties....1114
AssignedPagelets....1114
AvailableCategories....1114
AvailablePagelets....1115
Attributes....1115
Author....1115
AuthorAccess....1115
ColumnLayout....1115
CreationDate....1116
Description....1116
DynamicCategories....1116
HelpID....1116
HtmlText....1116
IsHideActionBar....1116

Copyright © 1988-2007, Oracle. All rights reserved.

IsLayoutLocked....1117
IsRenameable....1117
Label....1117
LayoutBehavior....1117
Name....1118
OwnerId....1118
Product....1118
PublicAccess....1118
QualifiedURL....1118
SequenceNumber....1119
StyleSheet....1119
ValidFrom....1119
ValidTo....1119
TabDefinition Collection....1119
TabDefinition Collection Methods....1120
DeleteItem....1120
First....1120
InsertItem....1121
ItemByName....1121
Next....1122
TabDefinition Collection Property....1122
Count....1122
AssignedPagelet Class....1123
AssignedPagelet Class Properties....1123
Column....1123
LayoutBehavior....1123
PageletName....1124
Row....1124
AssignedPagelet Collection....1124
AssignedPagelet Collection Methods....1124
DeleteItem....1124
First....1125
InsertItem....1125
ItemByName....1126
Next....1127
AssignedPagelet Collection Property....1127
Count....1127
AvailableCategory Class....1128
AvailableCategory Class Properties....1128
AvailablePagelets....1128

Copyright © 1988-2007, Oracle. All rights reserved.

CategoryName....1128
AvailableCategory Collection....1128
AvailableCategory Collection Methods....1129
First....1129
ItemByName....1129
Next....1130
AvailableCategory Collection Property....1130
Count....1130
AvailablePagelet Class....1130
AvailablePagelet Class Properties....1131
CategoryLabel....1131
CategoryName....1131
Column....1131
LayoutBehavior....1131
PageletLabel....1132
PageletName....1132
Row....1132
AvailablePagelet Collection....1132
AvailablePagelet Collection Methods....1132
First....1133
ItemByName....1133
Next....1133
AvailablePagelet Collection Property....1134
Count....1134
DynamicCategory Collection....1134
DynamicCategory Collection Methods....1134
DeleteItem....1135
First....1135
InsertItem....1136
ItemByName....1136
Next....1136
DynamicCategory Collection Property....1137
Count....1137
PageletCategory Class....1137
PageletCategory Class Method....1138
Save....1138
PageletCategory Class Properties....1138
Attributes....1138
Author....1139
AuthorAccess....1139

Copyright © 1988-2007, Oracle. All rights reserved.

Authorized....1139
CascadedPermissions....1139
CreationDate....1140
Description....1140
Label....1140
Name....1140
OwnerId....1140
Pagelets....1141
Permissions....1141
Product....1141
PublicAccess....1141
SequenceNumber....1142
PageletCategory Collection....1142
PageletCategory Collection Methods....1142
DeleteItem....1142
First....1143
InsertItem....1143
ItemByName....1144
Next....1144
PageletCategory Collection Property....1145
Count....1145
Pagelet Class....1145
Pagelet Class Method....1145
Save....1145
Pagelet Class Properties....1146
Attributes....1146
Author....1146
AuthorAccess....1146
Authorized....1147
CascadedPermissions....1147
ContentProvider....1147
CreationDate....1147
DefaultColumn....1147
Description....1148
EditPageContentProvider....1148
EditPageQueryString....1148
HelpID....1148
IsHideMinimize....1148
Label....1148
Name....1149

Copyright © 1988-2007, Oracle. All rights reserved.

OwnerId....1149
ParentName....1149
Permissions....1149
Product....1149
PublicAccess....1150
QualifiedURL....1150
SequenceNumber....1150
URL....1150
URLType....1151
Pagelet Collection....1151
Pagelet Collection Methods....1151
DeleteItem....1151
First....1152
InsertItem....1152
ItemByName....1153
Next....1154
Pagelet Collection Property....1154
Count....1154
UserHomepage Class....1154
UserHomepage Class Method....1155
Save....1155
UserHomepage Class Properties....1155
Greeting....1155
UserId....1156
UserTab....1156
UserTab Class....1156
UserTab Class Properties....1156
ColumnLayout....1156
Label....1157
QualifiedURL....1157
SelectedPagelets....1157
SequenceNumber....1157
TabName....1157
UserTab Collection....1158
UserTab Collection Methods....1158
DeleteItem....1158
First....1158
InsertItem....1159
ItemByName....1159
Next....1160

Copyright © 1988-2007, Oracle. All rights reserved.

UserTab Collection Property....1160
Count....1160
SelectedPagelet Class....1160
SelectedPagelet Class Properties....1161
CategoryName....1161
Column....1161
IsMinimized....1161
PageletName....1161
Row....1161
SelectedPagelet Collection....1162
SelectedPagelet Collection Methods....1162
DeleteItem....1162
First....1162
InsertItem....1163
ItemByName....1163
Next....1164
SelectedPagelet Collection Property....1164
Count....1164
Favorite Class....1165
Favorite Class Properties....1165
CRefName....1165
Label....1165
QualifiedURL....1165
SequenceNumber....1165
URL....1166
Favorite Collection....1166
Favorite Collection Methods....1166
DeleteItem....1166
First....1167
InsertItem....1167
ItemByLabel....1168
Next....1168
Save....1169
Favorite Collection Property....1169
Count....1169
PortalRegistry Classes Example....1169
Changing PortalRegistry Properties....1169
Adding a ContentProvider....1171
Adding a Folder....1172
Adding a Content Reference....1175

Copyright © 1988-2007, Oracle. All rights reserved.

Setting Permissions Using the PermissionValue Object....1177
Using Attributes....1179
Visual Basic Example....1180
C/C++ Example....1181

Chapter 28
PostReport Class....1185
PostReport Class Overview....1185
Determining the Distribution List....1185
Data Type of a PostReport Object....1186
Scope of a PostReport Object....1186
PostReport Class Built-in Function....1186
PostReport Class Methods....1186
AddDistributionOption....1186
Put....1187
PostReport Class Properties....1188
ExpirationDate....1188
OutDestFormat....1188
ProcessInstance....1189
ProcessName....1189
ProcessType....1189
ReportDescr....1190
ReportFolder....1190
ReportId....1190
ServerName....1191
SourceReportPath....1191
PostReport Class Examples....1191

Chapter 29
ProcessRequest Class....1193
Understanding ProcessRequest Class....1193
Data Type of a ProcessRequest Object....1194
Scope of a ProcessRequest Object....1194
Options for Items In Jobs and Jobsets....1194
Values for Output Type and Format....1198
Alternative Option To Specify Email or Web Attributes....1201
File Dependant Processing....1202
ProcessRequest Class Built-in Functions....1203

Copyright © 1988-2007, Oracle. All rights reserved.

ProcessRequest Class Methods.....1203
AddDistributionOption.....1203
PrintJobHTMLRpt.....1204
PrintJobRqstRpt.....1205
PrintSchdlHTMLRpt.....1208
RunJobSetNow.....1211
Schedule.....1211
SetEmailOption.....1212
SetItemFolder.....1213
SetOutputOption.....1214
UpdateRunStatus.....1215
ProcessRequest Class Properties.....1216
EmailAttachLog.....1216
EmailSubject.....1216
EmailText.....1216
EmailWebReport.....1217
FileName.....1217
JobName.....1217
LanguageCd.....1218
NotifyTextMsgNum.....1218
NotifyTextMsgSet.....1219
OutDest.....1219
OutDestFormat.....1220
OutDestType.....1220
PortalFolder.....1221
ProcessInstance.....1221
ProcessName.....1221
ProcessType.....1222
RunControlID.....1223
RunDateTime.....1223
RunLocation.....1223
RunRecurrence.....1224
RunStatus.....1224
Status.....1225
TimeZone.....1226
ProcessRequest Class Examples.....1226
Scheduling a Single Process.....1226
Scheduling a Job Where Job Item Changes Are Not Made.....1227
Scheduling a Job Where Job Item Changes Are Made.....1227
PrcsApi Class.....1228

Copyright © 1988-2007, Oracle. All rights reserved.

Scope of a PrcsApi Object....1228
Data Type of a PrcsApi Object....1229
How to Import the PrcsApi Class....1229
How to Create a PrcsApi Object....1229
PrcsApi Class Constructor....1230
PrcsApi....1230
PrcsApi Class Methods....1230
getAllFileNames....1230
notifyToWindow....1231

Chapter 30
Query Classes....1233
Understanding Query Classes....1233
Collections in the Query Classes....1233
Life-Cycle of a Query....1234
Query Class Hierarchy....1236
Query Classes Overview....1239
Query....1239
QuerySelect....1239
QueryRecords....1239
QueryOutputFields....1240
QuerySelectedFields....1240
QueryCriteria....1240
QueryDBRecords and QueryDBRecordFields....1240
Working With Criteria and Expressions....1241
Setting the Type....1241
Considerations Adding New Expressions....1241
Operator and Expression 2 Dependencies....1241
Query Monitor....1243
Using Query Metadata....1243
Running a Query....1245
Specifying User's Language....1246
Query Security....1246
Error Handling With Query Classes....1246
Understanding QueryOutputFields and QuerySelectedFields....1247
Understanding Having Criteria....1248
Data Type of Query Objects....1248
Scope of the Query Objects....1248
Session Class Methods....1249

Copyright © 1988-2007, Oracle. All rights reserved.

AdvancedSearchQueries....1249
AdvancedSearchRecords....1252
FindQueryDBRecords....1253
FindQueries....1254
FindQueriesDateRange....1255
GetQuery....1255
GetQuerySecurityProfile....1256
SearchQueryDBRecords....1256
SearchPrivateQueries....1258
SearchPublicQueries....1259
Query Collection....1260
Query Collection Methods....1260
First....1261
Item....1261
ItemByName....1262
Next....1262
Query Collection Property....1262
Count....1263
Query Class....1263
Query Class Methods....1263
AddQuerySelect....1263
AddPrompt....1264
Close....1264
Create....1265
Delete....1266
DeletePrompt....1266
FindExpression....1267
FormatResultString....1267
Open....1268
Rename....1269
RunToFile....1270
RunToRowset....1272
Save....1275
Query Class Properties....1275
Approved....1276
ApproveOprId....1276
ApproveUserId....1276
ApproveDtTm....1276
CreateOprId....1277
CreateDtTm....1277

Copyright © 1988-2007, Oracle. All rights reserved.

CreateUserId....1277
Description....1277
Disabled....1277
ExecAppName....1278
ExecLogging....1278
Folder....1278
LastSQLErrorCode....1278
LastUpdDttm....1279
LastUpdOprId....1279
LongDescription....1279
Metadata....1279
MetaSQL....1279
Name....1280
PDFFont....1280
Prompts....1280
PromptRecord....1281
PublicPrivate....1281
QuerySelect....1281
QueryStatistics....1282
RunTimePrompts....1282
SQL....1282
Type....1282
QuerySelect Class....1283
QuerySelect Methods....1284
AddAllFields....1284
AddCriteria....1284
AddExpression....1285
AddHavingCriteria....1285
AddQueryOutputField....1286
AddQueryRecord....1286
AddQuerySelectedField....1287
DeleteCriteria....1288
DeleteExpression....1288
DeleteField....1288
DeleteHavingCriteria....1289
DeleteRecord....1289
QuerySelect Properties....1290
Criteria....1290
Distinct....1290
Expressions....1290

Copyright © 1988-2007, Oracle. All rights reserved.

HavingCriteria....1291
ParentSelectNum....1291
QueryOutputFields....1291
QueryRecords....1291
QuerySelectedFields....1292
QuerySelects....1292
SelectNum....1292
SelectType....1292
QuerySelect Collection....1293
QuerySelect Collection Methods....1293
AddUnion....1293
DeleteQuerySelect....1293
First....1294
Item....1294
ItemBySelNum....1295
Next....1295
QuerySelect Collection Property....1296
Count....1296
QueryRecord Collection....1296
QueryRecord Collection Methods....1296
First....1296
Item....1297
ItemByAlias....1297
Next....1298
QueryRecord Collection Property....1298
Count....1298
QueryRecord Class....1299
QueryRecord Class Method....1299
GetField....1299
QueryRecord Class Properties....1299
Description....1300
JoinAlias....1300
JoinFieldName....1300
JoinType....1300
Name....1301
QueryFields....1301
RecordAlias....1301
QueryField Collection....1301
QueryField Collection Methods....1302
First....1302

Copyright © 1988-2007, Oracle. All rights reserved.

Item....1302
ItemByNameAndAlias....1303
ItemByExpNum....1303
Next....1303
QueryField Collection Property....1304
Count....1304
QueryField Class....1304
QueryField Methods....1305
AddTranslateExpression....1305
AddTranslateField....1305
QueryField Class Properties....1306
Aggregate....1306
ColumnNumber....1306
Description....1306
Decimal....1307
ExpNum....1307
Flag....1307
Format....1307
HeadingText....1308
HeadingType....1308
HeadingUniqueFieldName....1308
Length....1308
LongName....1309
Name....1309
OrderByDirection....1309
OrderByNumber....1309
QueryRecord....1310
RecordAlias....1310
ShortName....1310
TranslateEffDtLogic....1311
TranslateExpression....1311
TranslateField....1311
TranslateOption....1311
Type....1312
QueryCriteria Collection....1312
QueryCriteria Collection Methods....1312
First....1312
Item....1313
ItemByName....1313
Next....1314

Copyright © 1988-2007, Oracle. All rights reserved.

QueryCriteria Collection Property....1314
Count....1314
QueryCriteria Class....1314
QueryCriteria Class Methods....1315
AddExpr1Expression....1315
AddExpr1Field....1315
AddExpr2Expression....1316
AddExpr2Field1....1316
AddExpr2Field2....1317
AddExpr2List....1318
AddExpr2Subquery....1318
QueryCriteria Class Properties....1318
Expr1Expression....1319
Expr1Field....1319
Expr1Type....1319
Expr2Constant1....1320
Expr2Constant2....1320
Expr2Expression1....1320
Expr2Expression2....1321
Expr2Field1....1321
Expr2Field2....1321
Expr2List....1322
Expr2Subquery....1322
Expr2Type....1322
Logical....1325
LParenLvl....1325
Name....1325
Negation....1326
Operator....1326
R1ExprNum....1327
R2ExprNum....1328
R1ExprType....1328
R2ExprType....1328
RParenLvl....1328
QueryExpression Collection....1328
QueryExpression Collection Methods....1329
First....1329
Item....1329
ItemByName....1330
Next....1330

Copyright © 1988-2007, Oracle. All rights reserved.

QueryExpression Collection Property....1330
Count....1331
QueryExpression Class....1331
QueryExpression Class Properties....1331
Aggregate....1331
BindFlag....1331
Decimal....1332
ExpNum....1332
IsXlatExpression....1332
Length....1332
Name....1332
OutputField....1333
RightExprFlag....1333
SelectedField....1333
Text....1333
Type....1334
QueryList Class....1334
QueryList Class Methods....1334
AddListValue....1334
First....1335
Item....1335
Next....1336
QueryList Class Property....1336
Count....1336
QueryListValue Class....1337
QueryListValue Class Properties....1337
IsPrompt....1337
Value....1337
QueryRecordHierarchy Collection....1337
QueryRecordHierarchy Collection Methods....1337
First....1338
Item....1338
ItemByName....1338
Next....1339
QueryRecordHierarchy Collection Property....1339
Count....1340
QueryRecordHierarchy Class....1340
QueryRecordHierarchy Class Properties....1340
Description....1340
Level....1340

Copyright © 1988-2007, Oracle. All rights reserved.

Name....1341
ParentFlag....1341
Query Metadata Collection....1341
Query Metadata Collection Methods....1341
First....1341
Item....1342
ItemByName....1342
Next....1343
Query Metadata Collection Property....1343
Count....1343
Query Metadata Class....1344
Query Metadata Class Properties....1344
Name....1344
Value....1344
QueryStatistics Class....1345
QueryStatistics Class Properties....1345
AvgExecTime....1345
AvgFetchTime....1345
AvgNumRows....1345
ExecCount....1345
LastExecDtTm....1346
QuerySecurityProfile Class....1346
QuerySecurityProfile Class Properties....1346
AllowAnyJoin....1346
AllowDistinct....1346
AllowExpressions....1346
AllowSubqueries....1347
AllowUnions....1347
ApprovePrivateQuery....1347
ApprovePublicQuery....1347
CanCreatePublic....1347
CanCreateWorkFlow....1347
CanModifyQuery....1348
CanRunQuery....1348
CanRunToCrystal....1348
CanRunToExcel....1348
LimitUnapproved....1348
MaxInTreeCriteria....1349
MaxJoins....1349
MaxRowsToFetch....1349

Copyright © 1988-2007, Oracle. All rights reserved.

MaxUnapprovedRows....1349
QueryDBRecord Collection....1349
QueryDBRecord Collection Methods....1350
First....1350
Item....1350
ItemByName....1351
Next....1351
QueryDBRecord Collection Property....1351
Count....1352
QueryDBRecord Class....1352
QueryDBRecord Class Methods....1352
QueryDBRecordFieldByIndex....1352
QueryDBRecordFieldByName....1353
QueryDBRecord Class Properties....1353
Description....1353
Name....1353
QueryDBRecordFields....1353
RecordHierachy....1354
QueryDBRecordField Collection....1354
QueryDBRecordField Collection Methods....1354
First....1354
Item....1355
ItemByName....1355
Next....1356
Sort....1356
QueryDBRecordField Collection Property....1357
Count....1357
QueryDBRecordField Class....1357
QueryDBRecordField Class Properties....1357
Decimal....1357
Description....1358
Flag....1358
Format....1359
Length....1359
LongName....1360
LookupTableName....1360
LookupTableRecord....1360
Name....1360
Shortname....1360
Type....1360

Copyright © 1988-2007, Oracle. All rights reserved.

QueryPrompt Collection.....1361
QueryPrompt Collection Methods.....1361
First.....1361
Item.....1361
ItemByName.....1362
Next.....1362
QueryPrompt Collection Property.....1363
Count.....1363
QueryPrompt Class.....1363
QueryPrompt Properties.....1364
EditType.....1364
FieldDecimal.....1364
FieldFormat.....1364
FieldLength.....1365
FieldName.....1365
FieldType.....1365
HeadingText.....1366
HeadingType.....1366
LangCount.....1366
Name.....1366
PromptRecordFieldName.....1366
PromptTable.....1367
UniquePromptName.....1367
UseCount.....1367
Query Classes Examples.....1367
Creating a New Query.....1367
Adding Criteria.....1369
Using Outer Joins.....1375

Chapter 31
Record Class.....1377
Understanding Record Class.....1377
Shortcut Considerations.....1377
Record Methods Used to Create SQL Statements.....1378
Data Type of a Record Object.....1379
Scope of a Record Object.....1379
Record Class Built-in Functions.....1379
Record Class Methods.....1379
CompareFields.....1379

Copyright © 1988-2007, Oracle. All rights reserved.

CopyChangedFieldsTo....1380
CopyFieldsTo....1381
DBPatternMatch....1384
Delete....1385
ExecuteEdits....1386
GetField....1388
Insert....1390
Save....1391
SearchClear....1394
SelectByKey....1394
SelectByKeyEffDt....1396
SetDefault....1398
SetEditTable....1398
Update....1400
Record Class Properties....1402
FieldCount....1402
fieldname....1402
IsChanged....1402
IsDeleted....1403
IsEditError....1403
Name....1404
ParentRow....1404
RelLangRecName....1404

Chapter 32
Row Class....1407
Understanding Row Class....1407
Shortcut Considerations....1408
Data Type of a Row Object....1408
Scope of a Row Object....1408
Row Class Built-in Function....1408
Row Class Methods....1408
CopyTo....1409
GetNextEffRow....1409
GetPriorEffRow....1410
GetRecord....1410
GetRowset....1412
scrollname....1413
Row Class Properties....1413

Copyright © 1988-2007, Oracle. All rights reserved.

ChildCount....1413
DeleteEnabled....1414
IsChanged....1415
IsDeleted....1415
IsEditError....1416
IsNew....1416
ParentRowset....1416
recname....1417
RecordCount....1417
RowNumber....1417
Selected....1418
Style....1418
Visible....1419

Chapter 33
Rowset Class....1421
Understanding Rowset Class....1421
Shortcut Considerations....1422
Data Type of a Rowset Object....1422
Scope of a Rowset Object....1422
Rowset Class Built-In Functions....1422
Rowset Class Methods....1423
ClearDeletesChanges....1423
CopyTo....1424
DeleteRow....1426
Fill....1428
FillAppend....1430
Flush....1432
FlushRow....1433
GetCurrEffRow....1434
GetNewEffRow....1434
GetRow....1435
HideAllRows....1436
InsertRow....1437
Refresh....1438
Select....1439
SelectNew....1441
SetDefault....1442
ShowAllRows....1443

Copyright © 1988-2007, Oracle. All rights reserved.

Sort....1443
Rowset Class Properties....1444
ActiveRowCount....1445
ChangeOnInit....1445
DataAreaCollapsed....1445
DBRecordName....1446
DeleteEnabled....1446
EffDt....1447
EffSeq....1448
InsertEnabled....1448
IsEditError....1449
Level....1449
Name....1450
ParentRow....1450
ParentRowset....1451
RowCount....1451
SetComponentChanged....1451
TopRowNumber....1451

Chapter 34
RowsetCache Class....1453
Understanding a Rowset Cache....1453
Using the RowsetCache Class....1453
Error Handling....1454
Data Type of a RowsetCache Object....1455
Scope of a RowsetCache Object....1455
RowsetCache Class Built-in Functions....1455
RowsetCache Class Methods....1455
Delete....1455
Get....1456
Save....1457
RowsetCache Class Properties....1458
Content....1458
Description....1458

Chapter 35
Search Classes....1459
Understanding Search Class....1459

Copyright © 1988-2007, Oracle. All rights reserved.

Using the SearchResults Collection....1460
Understanding SearchResults....1460
Search Classes Hierarchy....1462
Error Handling....1464
Data Type of Search Objects....1465
Scope of the Search Objects....1465
Session Class Methods....1465
GetSearchIndexes....1466
GetSearchQuery....1466
PortalRegistry Class Methods....1466
BuildSearchIndex....1467
GetSearchQuery....1467
SearchQuery Class....1468
SearchQuery Class Methods....1468
Execute....1468
Parse....1469
SearchQuery Class Properties....1470
HitCount....1470
Indexes....1470
IndexName....1471
KnowledgeBase....1471
Language....1472
ProcessedCount....1472
QueryText....1472
RequestedFields....1472
ScorePrecision....1472
SortSpecifications....1473
ParseResult Collection....1473
ParseResult Collection Methods....1473
First....1473
Next....1474
ParseResult Collection Properties....1474
ErrorCount....1474
WarningCount....1475
ParseResult Class....1475
ParseResult Class Properties....1475
Message....1475
Severity....1475
SearchResult Collection....1476
SearchResult Collection Methods....1476

Copyright © 1988-2007, Oracle. All rights reserved.

First....1476
Item....1476
Next....1477
SearchResult Collection Property....1477
Count....1477
SearchResult Class....1477
SearchResult Class Properties....1477
Key....1478
Score....1478
ScoreAsNumber....1478
SearchFields....1478
SearchField Collection....1479
SearchField Collection Methods....1479
First....1479
ItemByName....1479
Next....1480
SearchField Collection Property....1480
Count....1480
SearchField Class....1480
SearchField Class Properties....1480
Name....1480
Value....1481
SearchIndexes Collection....1481
SearchIndexes Collection Methods....1481
First....1481
ItemByName....1481
Next....1482
SearchIndex Collection Property....1482
Count....1482
SearchIndex Class....1482
SearchIndex Class Method....1483
Save....1483
SearchIndex Class Properties....1483
ExtraOptions....1483
FSOpts....1484
HTTPOpts....1484
Languages....1484
Location....1484
Name....1484
RecOpts....1484

Copyright © 1988-2007, Oracle. All rights reserved.

Schedules....1485
Type....1485
SearchRecord Options Class....1485
SearchRecord Options Class Properties....1486
Fields....1486
Filter....1486
IncrementalView....1486
RecName....1486
VeggieKey....1487
ZoneOptions....1488
SearchRecordField Collection....1488
SearchRecordField Collection Methods....1488
DeleteItem....1488
First....1489
InsertItem....1489
ItemByName....1490
Next....1490
SearchRecordField Collection Property....1490
Count....1491
SearchRecordField Class....1491
SearchRecordField Class Properties....1491
FieldName....1491
IsAttachment....1491
IsVerityField....1492
IsWordIndex....1492
RecordName....1492
Search Language Collection....1492
Search Language Collection Methods....1492
DeleteItem....1492
First....1493
InsertItem....1493
ItemByName....1494
Next....1494
Search Language Collection Property....1494
Count....1494
Search Language Class....1495
Search Language Class Properties....1495
LanguageCd....1495
MapLanguageCd....1495
Search Schedule Collection....1495

Copyright © 1988-2007, Oracle. All rights reserved.

Search Schedule Collection Methods....1495
DeleteItem....1496
First....1496
InsertItem....1496
ItemByName....1497
Next....1497
Search Schedule Collection Property....1497
Count....1497
Search Schedule Class....1498
Search Schedule Class Properties....1498
BuildType....1498
RunCntlId....1498
RunRecurrence....1499
ServerName....1499
Search HTTP Options Class....1499
Search HTTP Options Class Properties....1499
AllowHTTPS....1499
DomainLimit....1500
GlobList....1500
GlobListType....1500
LinkDepth....1500
MIMEList....1501
MIMEListType....1501
ProxyHost....1501
ProxyPort....1501
StartOpts....1502
Search FS Options Class....1502
Search FS Options Class Properties....1502
GlobList....1502
GlobListType....1502
MIMEList....1503
MIMEListType....1503
StartOpts....1503
Search Start Options Collection....1504
Search Start Options Collection Methods....1504
DeleteItem....1504
First....1504
InsertItem....1505
ItemByName....1505
Next....1506

Copyright © 1988-2007, Oracle. All rights reserved.

Search Start Options Collection Property.....1506
Count.....1506
Search Start Options Class.....1506
Search Start Options Class Properties.....1506
IsDomainRestricted.....1507
IsHostRestricted.....1507
Value.....1507
Search API Examples.....1507
General Purpose Search API Example.....1507
Portal Search API Example.....1508

Chapter 36
Session Class.....1511
Understanding Session Class.....1511
Security and Access to the PeopleSoft System.....1512
Error Handling.....1512
Displaying Error Messages.....1513
Regional Settings.....1513
Trace Settings.....1514
Session Object Declaration.....1514
State Considerations.....1515
Considerations for Declaring Collections.....1515
Scope of a Session Object.....1515
Session Class Built-in Function.....1516
Session Class Methods.....1516
Connect.....1516
Disconnect.....1518
API Instantiation Methods.....1518
Session Class Properties.....1519
ErrorPending.....1519
PSMessages.....1519
PSMessagesMode.....1520
RegionalSettings.....1521
Repository.....1521
SuspendFormatting.....1521
TraceSettings.....1521
WarningPending.....1521
Accessing a PSMessages Collection.....1522
PSMessages Collection Methods.....1522

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteAll....1522
DeleteItem....1523
First....1523
Item....1523
Next....1524
PSMessages Collection Property....1524
Count....1524
PSMessage Class Properties....1525
ExplainText....1525
MessageNumber....1525
MessageSetNumber....1525
MessageType....1525
Source....1526
Text....1528
Regional Settings Class Properties....1528
ClientTimeZone....1528
CurrencyFormat....1529
CurrencySymbol....1529
DateFormat....1529
DateSeparator....1530
DecimalSymbol....1530
DigitGroupingSymbol....1530
LanguageCD....1530
UseLocalTime....1531
1159Separator....1531
2359Separator....1532
Trace Setting Class Properties....1532
API....1533
COBOLStmtTimings....1533
ConnDisRollbackCommit....1533
DBSpecificCalls....1533
ManagerInfo....1533
NetworkServices....1534
NonSSBs....1534
OutputUNICODE....1534
PCExtFcnCalls....1534
PCFcnReturnValues....1534
PCFetchedValues....1535
PCIntFcnCalls....1535
PCListProgram....1535

Copyright © 1988-2007, Oracle. All rights reserved.

PCParameterValues....1535
PCProgramStatements....1535
PCStack....1535
PCStartOfPrograms....1536
PCTraceProgram....1536
PCVariableAssignments....1536
RowFetch....1536
SQLStatement....1536
SQLStatementVariables....1537
SSBs....1537
SybBindInfo....1537
SybFetchInfo....1537
TraceFile....1537

Chapter 37
SOAPDoc Class....1539
Understanding SOAPDoc Class....1539
SOAP Message Exchange Model....1540
SOAP Message Document....1540
The SOAPDoc Class....1542
SOAPDoc Object Creation....1542
SOAP Header Section....1542
SOAP Envelope Section....1543
SOAP Body Section....1543
SOAP Method Section....1543
SOAPDoc Section Access....1544
PeopleCode to Create a SOAPDoc....1544
SOAPDoc Fault Section....1545
The ValidateSOAPDoc Method....1545
SOAPDoc Objects and Messages....1545
Data Type of a SOAPDoc Object....1546
Scope of a SOAPDoc Object....1546
SOAPDoc Built-In Functions....1546
SOAPDoc Class Methods....1546
AddBody....1546
AddEnvelope....1547
AddFault....1550
AddHeader....1551
AddMethod....1552

Copyright © 1988-2007, Oracle. All rights reserved.

AddParm....1554
GetParmName....1554
GetParmValue....1555
ValidateSOAPDoc....1556
SOAPDoc Class Properties....1556
BodyNode....1557
EnvelopeNode....1557
FaultCode....1557
FaultCodeS....1557
FaultString....1558
HeaderNode....1558
MethodName....1559
MethodNode....1559
ParmCount....1559
XmlDoc....1560
SOAPDoc Class Examples....1560
Creating a SOAP Document....1560
Reading an Existing SOAP Document....1561
Using SOAP XML Text....1561

Chapter 38
SQL Class....1563
Understanding SQL Class....1563
Using Record Class SQL....1564
Creating a SQL Definition....1565
Binding and Executing of SQL Statements....1565
Fetching From a Select....1566
Using Array of Any for Bind Values or Fetch Results....1566
Understanding Different "Styles" of SELECT....1567
Reusing a Cursor....1567
Using the ReuseCursor Property....1568
Reusing a Cursor Without the ReuseCursor Property....1568
Understanding SQL Objects and PeopleSoft Application Engine Programs....1570
Declaring a SQL Object....1571
Scope of an SQL Object....1571
SQL Class Built-in Functions....1571
SQL Class Methods....1572
Close....1572
Execute....1572

Copyright © 1988-2007, Oracle. All rights reserved.

Fetch....1574
Open....1576
SQL Class Properties....1577
BulkMode....1577
IsOpen....1579
LTrim....1579
ReuseCursor....1580
RowsAffected....1580
Status....1581
TraceName....1581
Value....1582

Chapter 39
SyncServer Class....1585
Understanding Synchronization Server Events and the SyncServer Classes....1585
Understanding the Synchronization Events....1585
Understanding the SyncServer Class....1587
Considerations Using Synchronization Events and the SyncServer Class....1588
Using the Synchronization Server Events....1589
Using OnConflict....1589
Using OnSelect....1590
Using OnValidate....1592
Using Attachment Events....1593
Using Filtering....1594
Scope of SyncServer Objects....1594
SyncServer Class....1595
SyncServer Class Methods....1595
AddReference....1595
AddReferenceType....1595
CheckForChanges....1596
Notify....1598
SelectAll....1599
SyncServer Class Properties....1599
ClientPlatform....1599
ConflictAlgorithm....1600
ConflictStatus....1600
ExcludeProperty....1600
ExportLocation....1601
IsReferenceUnresolved....1601

Copyright © 1988-2007, Oracle. All rights reserved.

LastSyncDateTime....1601
LastSyncRowCount....1602
PropertyValue....1602
SessionID....1602
Synchronizing....1603
SyncIDs....1603
TypeID....1603
validateID....1604
validateRowCount....1604
validateVersion....1604
Advanced Considerations for Synchronization....1605
Using OnValidateSet....1605
Using OnGetDefinition....1606

Chapter 40
Tree Classes....1609
Understanding Tree Classes....1609
Relationships Between Different Tree Classes....1610
Collections in the Tree Classes....1612
Error Handling With Trees....1612
Leaves and Nodes Insert Verification....1613
Data Types for Tree Objects....1614
Scope of the Tree Objects....1614
Tree Classes Implementation....1615
Session Class Methods....1617
GetTree....1617
GetTreeStructure....1618
Branch Collection....1618
Branch Collection Property....1618
Count....1619
Leaf Class....1619
Leaf Class Methods....1619
Cut....1619
Delete....1620
DeleteByRange....1620
InsertDynSib....1620
InsertSib....1621
MoveAsChild....1622
MoveAsChildByName....1623

Copyright © 1988-2007, Oracle. All rights reserved.

MoveAsSib....1623
MoveAsSibByRange....1624
PasteSib....1625
UpdateRanges....1625
Leaf Class Properties....1625
DisplayLevelNumber....1626
Dynamic....1626
HasNextSib....1626
HasPrevSib....1626
ImageName....1626
IsChanged....1627
IsCut....1627
IsDeleted....1627
IsInserted....1627
NextSib....1627
Parent....1628
PrevSib....1628
RangeFrom....1628
RangeTo....1628
TreeBranchName....1629
TreeEffDt....1629
TreeName....1629
TreeSetId....1629
TreeUserKeyValue....1629
Level Collection....1629
Level Collection Methods....1630
Add....1630
Item....1630
Remove....1631
Level Collection Properties....1631
Count....1631
First....1631
Last....1631
Next....1631
Level Class....1632
Level Class Methods....1632
Create....1632
Level Class Properties....1632
AllValuesAudit....1632
Description....1633

Copyright © 1988-2007, Oracle. All rights reserved.

Name....1633
Number....1633
TreeBranchName....1633
TreeEffDt....1633
TreeName....1633
TreeSetId....1634
TreeUserKeyValue....1634
Node Class....1634
Node Class Methods....1634
Branch....1634
Cut....1635
Delete....1635
DeleteByName....1636
Expand....1636
InsertChildLeaf....1637
InsertChildNode....1637
InsertChildRecord....1638
InsertDynChildLeaf....1639
InsertSib....1640
InsertSibRecord....1640
MoveAsChild....1641
MoveAsChildByName....1641
MoveAsSib....1642
MoveAsSibByName....1643
PasteChild....1643
PasteSib....1644
RefreshDescription....1644
Rename....1645
SwitchLevel....1645
Unbranch....1645
Node Class Properties....1646
AllChildCount....1646
AllChildNodeCount....1646
ChildLeafCount....1647
ChildNodeCount....1647
ColImageName....1647
Description....1647
DisplayLevelNumber....1647
ExpImageName....1647
FirstChildLeaf....1648

Copyright © 1988-2007, Oracle. All rights reserved.

FirstChildNode....1648
HasChildLeaves....1648
HasChildNodes....1648
HasChildren....1648
HasNextSib....1649
HasPrevSib....1649
IsBranched....1649
IsChanged....1649
IsCut....1649
IsDeleted....1649
IsInserted....1650
IsRoot....1650
LastChildLeaf....1650
LastChildNode....1650
LevelNumber....1651
Name....1651
NextSib....1651
Parent....1651
PrevSib....1652
State....1652
TreeBranchName....1652
TreeEffDt....1653
TreeName....1653
TreeSetId....1653
TreeUserKeyValue....1653
Type....1653
Tree Class....1653
Tree Class Methods....1654
Audit....1654
AuditByName....1655
Close....1656
Copy....1656
Create....1658
Delete....1659
Exists....1660
FindLeaf....1661
FindNode....1661
FindRoot....1662
InsertRoot....1663
LeafExists....1663

Copyright © 1988-2007, Oracle. All rights reserved.

LockTree....1664
Open....1665
OpenForExport....1666
OpenWholeTree....1667
NodeExists....1668
Rename....1668
Save....1669
SaveAs....1670
SaveAsDraft....1671
SaveDraft....1672
SetImportMode....1673
TreeLocksNumber....1673
UnlockTree....1674
UpdateLock....1675
Tree Class Properties....1676
AllValues....1676
AuditDetails....1676
Branches....1676
BranchImageName....1676
BranchLevel....1677
BranchName....1677
Category....1677
Description....1677
DuplicateLeaves....1678
EffDt....1678
HasDetailRanges....1678
HasLockedBranches....1679
IsBranched....1679
IsChanged....1679
IsOpen....1680
IsQueryTree....1680
IsValid....1680
IsVersionChanged....1680
IsWholeTree....1681
KeyBranchName....1681
KeyEffDt....1681
KeyName....1681
KeySetId....1682
KeyUserKeyValue....1682
LeafCount....1682

Copyright © 1988-2007, Oracle. All rights reserved.

LeafImageName....1682
LeafOnClipboard....1682
LevelCount....1683
Levels....1683
LevelUse....1683
LockOwner....1684
LockStatus....1684
Name....1684
NodeColImageName....1685
NodeCount....1685
NodeExpImageName....1685
NodeOnClipboard....1685
ParentLevel....1686
ParentName....1686
PerformanceMethod....1686
PerformanceSelector....1687
PerformanceSelectorOption....1687
SetID....1688
Status....1688
Structure....1688
StructureName....1688
TreeImageName....1689
UserKeyValue....1689
UseUpdateReservation....1689
Tree Structure Class....1690
Tree Structure Class Methods....1690
Close....1690
Copy....1690
Create....1691
Delete....1691
Open....1692
Rename....1692
Save....1693
Tree Structure Class Properties....1693
Description....1693
DetailComponent....1694
DetailField....1694
DetailMenu....1694
DetailMenuBar....1695
DetailMenuItem....1695

Copyright © 1988-2007, Oracle. All rights reserved.

DetailMultiNavigate....1695
DetailPage....1695
DetailRecord....1696
IndirectionMethod....1696
KeyName....1698
LevelComponent....1698
LevelMenu....1698
LevelMenuBar....1698
LevelMenuItem....1698
LevelPage....1699
LevelRecord....1699
Name....1699
NodeComponent....1700
NodeField....1700
NodeMenu....1700
NodeMenuBar....1700
NodeMenuItem....1701
NodeMultiNavigate....1701
NodePage....1701
NodeRecord....1701
NodeUserKeyField....1702
SummarySetId....1702
SummaryLevelNumber....1702
SummaryTreeName....1703
SummaryUserKeyValue....1703
Type....1703
Traverse Tree Hierarchy Example....1703

Chapter 41
Universal Queue Classes....1707
Understanding Universal Queue Classes....1707
MCFFactory Class Hierarchy....1707
Scope of the Universal Queue Classes....1708
Data Types of the Universal Queue Classes....1709
How to Import Universal Queue Classes....1709
How to Create a Universal Queue Object....1710
Universal Queue Classes Constructors....1710
Agent....1710
AgentPhysQueueProps....1711

Copyright © 1988-2007, Oracle. All rights reserved.

AgentPhysQueueTasks....1711
Broadcast....1712
LogicalQueue....1712
MCFFactory....1713
PhysicalQueue....1713
Task....1714
TaskList....1715
Util....1716
Agent Class....1717
Agent Methods....1717
Delete....1717
Refresh....1718
RefreshQTaskList....1719
Agent Properties....1719
AgentID....1719
AgentProps....1720
AgentTasks....1720
Buddy....1721
Language....1721
Name....1721
NickName....1721
PhysicalQueueID....1721
TotalPhysicalQueues....1722
AgentPhysQueueProps Class....1722
AgentPhysQueueProps Properties....1722
AgentID....1722
PhysicalQueueID....1722
SkillLevel....1723
WorkLoad....1723
AgentPhysQueueTasks Class....1723
AgentPhysQueueTasks Method....1723
Refresh....1723
AgentPhysQueueTasks Properties....1724
AcceptedTaskList....1724
AssignedTaskList....1724
AgentID....1725
PhysicalQueueID....1725
Broadcast Class....1725
Broadcast Class Method....1725
Broadcast....1725

Copyright © 1988-2007, Oracle. All rights reserved.

LogicalQueue Class....1727
LogicalQueue Properties....1727
LogicalQueueID....1727
PhysicalQueueID....1727
MCFFactory Class....1728
MCFFactory Property....1728
LogicalQueue....1728
PhysicalQueue Class....1728
PhysicalQueue Methods....1729
Refresh....1729
RefreshTaskList....1729
PhysicalQueue Properties....1730
AcceptedTaskList....1730
AssignedTaskList....1730
Agent....1730
BrowserURL....1730
EnqueuedTaskList....1731
EscalatedTaskList....1731
InternalURL....1731
IsActive....1731
LogicalQueueID....1732
OverflowedTaskList....1732
PhysicalQueueID....1732
RENURLID....1733
TotalAgents....1733
Task Class....1733
Task Methods....1733
Close....1733
Enqueue....1734
Refresh....1735
RefreshStatus....1736
Task Properties....1737
AgentID....1737
ApplicationData....1737
Comments....1737
EnqueueTime....1737
EscalationTime....1737
Language....1737
OriginalTime....1738
OverFlowTime....1738

Copyright © 1988-2007, Oracle. All rights reserved.

PhysicalQueueID....1738
TiedAgentID....1738
TaskList Class....1738
TaskList Method....1738
Refresh....1739
TaskList Properties....1739
AgentID....1739
PhysicalQueueID....1739
Task....1739
TaskType....1740
Total....1740
Util Class....1740
Util Methods....1740
GetLogicalQueue....1741
GetTimeDiff....1741
MCFFactory Example....1742

Chapter 42
XmlDoc Class....1745
Understanding XmlDoc Class....1745
When to Use an XmlDoc Object....1746
XmlDoc Object Creation....1746
Considerations Using a Unique Namespace....1747
Considerations Using Rowsets....1748
XmlNode Class Considerations....1749
Accessing and Traversing an XmlNode Object....1749
Error Handling....1749
SOAPDoc Object Considerations....1750
Scope of XmlDoc and XmlNode Objects....1750
Data Type of an XmlDoc or XmlNode Object....1750
XmlDoc Class Built-in Functions....1751
XmlDoc Methods....1751
CopyRowset....1751
CopyToPSFTMessage....1752
CopyToRowset....1754
CreateDocumentElement....1756
CreateDocumentType....1757
GenFormattedXmlString....1758
GenXmlString....1759

Copyright © 1988-2007, Oracle. All rights reserved.

GetElementsByTagName....1759
LoadIBContent....1760
ParseXmlFromURL....1762
ParseXmlString....1763
XmlDoc Properties....1764
DocumentElement....1764
IsNull....1765
XmlNode....1765
XmlNode Methods....1765
AddAttribute....1765
AddAttributeNS....1766
AddCDataSection....1767
AddComment....1768
AddElement....1769
AddElementNS....1770
AddEntityReference....1771
AddNode....1772
AddProcessInstruction....1773
AddText....1774
CopyNode....1774
FindNode....1776
FindNodes....1777
GenXmlString....1778
GetAttributeName....1778
GetAttributeValue....1779
GetCDataValue....1779
GetCDataValues....1780
GetChildNode....1781
GetElement....1781
GetElements....1782
GetElementsByTagName....1783
GetElementsByTagNameNS....1784
InsertCDataSection....1785
InsertComment....1786
InsertElement....1787
InsertElementNS....1789
InsertEntityReference....1790
InsertNode....1791
InsertProcessInstruction....1792
InsertText....1794

Copyright © 1988-2007, Oracle. All rights reserved.

RemoveAllChildNode....1795
RemoveChildNode....1796
XmlNode Properties....1797
AttributesCount....1797
ChildNodeCount....1797
Index....1798
IsNull....1798
LocalName....1798
NamespaceURI....1798
NextSibling....1799
NodeName....1799
NodePath....1799
NodeType....1799
NodeValue....1800
ParentNode....1800
Prefix....1800
PreviousSibling....1800

Chapter 43
XML Publisher Classes....1801
Understanding the PeopleSoft XML Publisher and the XML Publisher Classes....1801
XML Publisher Terms....1802
Life-Cycles of the XML Publisher Classes....1803
XML Publisher Classes Hierarchies....1804
Report Manager Definition Classes Hierarchy....1804
XML Publisher Report Search Classes Structure....1805
XML Publisher Generation Classes Structure....1806
XML Publisher Engine Classes Hierarchy....1806
Error Handling....1808
Data Types of XML Publisher Objects....1808
Scope of XML Publisher Objects....1808
How to Import the XML Publisher Classes....1808
How to Create XML Publisher Objects....1809
Search Operator Values....1809
Report Manager Definition Classes Constructors....1810
DataSourceDefn....1810
ReportDefn....1811
TemplateDefn....1811
TemplateFile....1812

Copyright © 1988-2007, Oracle. All rights reserved.

TranslationFile....1813
DataSourceDefn Class Methods....1813
Get....1813
GetData....1814
GetSampleData....1814
GetSchema....1815
DataSourceDefn Class Properties....1815
ActiveFlag....1815
Description....1815
IsPublic....1816
LastUpdatedBy....1816
LastUpdateDTTM....1816
Name....1816
ObjectOwnerID....1816
RegisteredBy....1816
RegisteredDTTM....1817
Type....1817
ReportDefn Class Methods....1817
DisplayOutput....1817
Get....1818
GetActiveTemplatesByDistributionChannel....1818
GetDatasource....1818
GetDefaultTemplate....1819
GetOutDestFormatString....1819
GetPSQueryPromptRecord....1820
GetTemplate....1821
GetTemplateList....1821
PrintOutput....1821
ProcessEtextReport....1822
ProcessReport....1823
Publish....1824
SetPSQueryPromptRecord....1825
SetRuntimeDataRowset....1826
SetRuntimeDataXMLDoc....1826
SetRuntimeDataXMLFile....1827
SetRuntimeProperties....1828
ReportDefn Class Properties....1828
AllowRecipientEntry....1829
AllowViewerEntry....1829
ArchiveAfter....1829

Copyright © 1988-2007, Oracle. All rights reserved.

BurstFieldName....1829
CategoryId....1830
Description....1830
DestinationPrinter....1830
FolderName....1830
ID....1830
IsReadOnly....1831
LastUpdateDTTM....1831
LastUpdateOprID....1831
ObjectOwnerID....1831
OutDestination....1831
OutDestinationType....1831
OutputEditable....1832
RegisteredBy....1832
RegisteredDTTM....1832
SecurityFieldName....1832
SecurityIDType....1832
SecurityJoinTable....1833
Status....1833
TemplateControlFieldName....1833
UseBurstValueAsOutputFileName....1834
TemplateDefn Class Methods....1834
Get....1834
GetActiveTemplateFile....1834
GetTemplateFile....1835
GetTemplateFileList....1835
TemplateDefn Class Properties....1836
Description....1836
DistributionChannel....1836
ID....1836
IsReadOnly....1836
IsSubTemplate....1837
LanguageCode....1837
LastUpdateDTTM....1837
LastUpdateOprID....1837
ObjectOwnerID....1837
RegisteredBy....1837
RegisteredDTTM....1837
ReportCategoryID....1838
Type....1838

Copyright © 1988-2007, Oracle. All rights reserved.

TemplateFile Class Methods....1838
GetFile....1838
GetMapFile....1839
GetTranslationFile....1839
GetTranslationFileList....1840
TemplateFile Class Properties....1840
EffectiveDate....1840
FileName....1840
MapFileName....1840
Status....1841
TranslationFile Class Method....1841
GetFile....1841
TranslationFile Class Properties....1842
Description....1842
FileName....1842
LanguageCode....1842
Status....1842
XMLPManager Class Methods....1842
GetDatasourceDefnList....1843
GetReportDefnList....1844
GetTemplateDefnList....1846
Report Manager Search Classes Constructors....1848
Report....1848
ReportManager....1849
SearchAttribute....1850
Report Class....1850
Report Class Properties....1850
contentId....1850
CreatedDateTime....1851
DatabaseName....1851
Description....1851
ExpireDate....1851
FileURL....1851
FolderName....1851
ProcessInstanceID....1852
ReportInstanceID....1852
ReportName....1852
ReportURL....1852
ReportManager Class....1852
ReportManager Class Methods....1853

Copyright © 1988-2007, Oracle. All rights reserved.

AddSearchFieldCriteria....1853
GetReportList....1853
SetBurstFieldCriteria....1854
SetCaseSensitive....1854
SetDateCriteria....1855
SetFolderCriteria....1855
SetProcessInstanceCriteria....1856
SetReportIDCriteria....1856
SetUserIdCriteria....1857
SearchAttribute Class....1857
SearchAttribute Class Properties....1857
attrName....1857
attrValue....1857
compareOp....1858
XML Generation Class Constructors....1858
QueryDS....1858
RowsetDS....1858
QueryDS Class....1859
QueryDS Class Methods....1859
GetXMLData....1859
GetXMLSampleData....1860
GetXSDSchema....1861
RowsetDS Class....1861
RowsetDS Class Methods....1861
AddXMLData....1861
AddXSDSchema....1862
GetXMLData....1863
GetXSDSchema....1863
XML Publisher Engine Class Constructors....1864
EFTProcessor....1864
FOProcessor....1864
FormProcessor....1865
FOUtility....1865
PageNumber....1866
PDFMapTool....1866
PDFMerger....1866
Properties....1867
RTFProcessor....1867
Watermark....1868
EFTProcessor Class....1868

Copyright © 1988-2007, Oracle. All rights reserved.

EFTProcessor Class Methods....1868
GenerateOutput....1868
GenerateXSL....1869
EFTProcessor Class Properties....1870
ConfFile....1870
ConfProp....1871
FOProcessor Class....1871
FOProcessor Class Methods....1871
GenerateMergedOutput....1871
GenerateMergedOutputFO....1872
GenerateOutput....1873
GenerateOutputFO....1874
FOProcessor Class Properties....1875
ConfFile....1875
ConfProp....1876
Locale....1876
FormProcessor Class....1877
FormProcessor Class Methods....1877
ExtractFieldNames....1877
ExtractFieldValues....1877
FillForm....1878
FormProcessor Class Properties....1879
ConfFile....1879
ConfProp....1880
Locale....1880
FOUtility Class....1880
FOUtility Class Methods....1881
GenerateFO....1881
MergeFOs....1881
FOUtility Class Property....1882
Prop....1882
PageNumber Class....1882
PageNumber Class Properties....1883
BackgroundFile....1883
FontName....1883
FontSize....1884
PositionX....1884
PositionY....1884
StartFromPageNum....1884
StartNum....1885

Copyright © 1988-2007, Oracle. All rights reserved.

PDFMapTool Class.....1885
PDFMapTool Class Methods.....1885
EnableMap.....1885
extractMap.....1886
processForm.....1887
PDFMapTool Class Property.....1887
Locale.....1887
PDFMerger Class.....1888
PDFMerger Class Method.....1888
MergePDFs.....1888
PDFMerger Class Properties.....1889
ConfProp.....1889
Locale.....1889
PageNumber.....1889
Watermark.....1890
Properties Class.....1890
Properties Class Methods.....1891
GetProperty.....1891
SetProperty.....1891
RTFProcessor Class.....1892
RTFProcessor Class Methods.....1892
GenerateXSL.....1892
GenerateXSLXliff.....1893
RTFProcessor Class Properties.....1894
ConfFile.....1894
ConfProp.....1895
Locale.....1895
Watermark Class.....1895
Watermark Class Properties.....1896
ImageFile.....1896
ImageFileLowerLeftX.....1896
ImageFileLowerLeftY.....1896
ImageFileUpperRightX.....1896
ImageFileUpperRightY.....1897
PageIndex.....1897
Text.....1897
TextAngle.....1897
TextFontName.....1897
TextFontSize.....1897
TextStartPosX.....1898

Copyright © 1988-2007, Oracle. All rights reserved.

TextStartPosY....1898
XML Publisher Classes Example....1898
Generate and Publish a Report....1898

Appendix A
Quick Reference for PeopleCode Classes....1901
PeopleCode Classes....1901
Typographical Conventions and Visual Cues....1901
AESection....1901
Analytic Calculation Engine Classes....1902
Analytic Calculation Engine Metadata Classes....1905
Analytic Grid Classes....1913
Analytic Type Classes....1914
Application Classes....1916
Array....1917
BPEL....1919
Business Interlinks and BIDocs....1920
Chart....1922
Component Interface....1929
Exception....1929
Crypt....1931
Exception....1931
Field....1932
File....1936
Grid and GridColumn....1938
Internet Script....1939
Java....1943
Mail Classes....1943
Message....1952
Mobile....1960
Notification....1965
Optimization....1971
Page....1974
PortalRegistry....1975
PostReport....1975
ProcessRequest....1976
Query....1979
Record....1979
Row....1981

Copyright © 1988-2007, Oracle. All rights reserved.

Rowset....1982
RowsetCache....1984
Search....1985
Session....1985
SOAPDoc....1985
SQL....1987
SyncServer....1988
TransformData....1990
Tree....1990
Universal Queue....1991
XmlDoc....1994
XML Publisher....1999
Session Classes....2010
Session Classes Methods and Properties....2010
API Repository Classes Methods and Properties....2015
Component Interface Class Methods and Properties....2018
Search Classes Methods and Properties....2022
PortalRegistry Classes Methods and Properties....2030
Query Classes Methods and Properties....2056
Tree Classes Methods and Properties....2074
Mapping of Functions to Methods and Properties....2085
Mapping of Old Names to New Names....2088
Deprecated Functions....2090
Deprecated Methods and Properties....2091
Deprecated Messaging PeopleCode Functions, Methods and Properties....2092
Changed Messaging PeopleCode....2092
Deprecated Messaging PeopleCode Built-in Functions....2093
Deprecated Message Class Methods and Properties....2094
Additional Messaging Deprecated Methods and Properties....2095
Integration Broker Deprecated Methods....2095
PeopleCode Syntax Quick Reference....2096

Glossary of PeopleSoft Enterprise Terms....2101

Index....2127

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

About This PeopleBook

PeopleSoft Enterprise PeopleBooks provide you with the information that you need to implement and use PeopleSoft Enterprise applications from Oracle.

This preface discusses:

- PeopleSoft Enterprise application prerequisites.
- Application fundamentals.
- Documentation updates and printed documentation.
- Additional resources.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common elements in PeopleBooks.

Note. PeopleBooks document only elements, such as fields and check boxes, that require additional explanation. If an element is not documented with the process or task in which it is used, then either it requires no additional explanation or it is documented with common elements for the section, chapter, PeopleBook, or product line. Elements that are common to all PeopleSoft Enterprise applications are defined in this preface.

PeopleSoft Enterprise Application Prerequisites

To benefit fully from the information that is covered in these books, you should have a basic understanding of how to use PeopleSoft Enterprise applications.

You might also want to complete at least one introductory training course, if applicable.

You should be familiar with navigating the system and adding, updating, and deleting information by using PeopleSoft Enterprise menus, pages, or windows. You should also be comfortable using the World Wide Web and the Microsoft Windows or Windows NT graphical user interface.

These books do not review navigation and other basics. They present the information that you need to use the system and implement your PeopleSoft Enterprise applications most effectively.

Application Fundamentals

Each application PeopleBook provides implementation and processing information for your PeopleSoft Enterprise applications.

For some applications, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals PeopleBook. Most product lines have a version of the application fundamentals PeopleBook. The preface of each PeopleBook identifies the application fundamentals PeopleBooks that are associated with that PeopleBook.

Copyright © 1988-2007, Oracle. All rights reserved.

The application fundamentals PeopleBook consists of important topics that apply to many or all PeopleSoft Enterprise applications. Whether you are implementing a single application, some combination of applications within the product line, or the entire product line, you should be familiar with the contents of the appropriate application fundamentals PeopleBooks. They provide the starting points for fundamental implementation tasks.

Documentation Updates and Printed Documentation

This section discusses how to:

- Obtain documentation updates.
- Download and order printed documentation.

Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on Oracle's PeopleSoft Customer Connection website. Through the Documentation section of Oracle's PeopleSoft Customer Connection, you can download files to add to your PeopleBooks Library. You'll find a variety of useful and timely materials, including updates to the full line of PeopleSoft Enterprise documentation that is delivered on your PeopleBooks CD-ROM.

Important! Before you upgrade, you must check Oracle's PeopleSoft Customer Connection for updates to the upgrade instructions. Oracle continually posts updates as the upgrade process is refined.

See Also

Oracle's PeopleSoft Customer Connection, http://www.oracle.com/support/support_peoplesoft.html

Downloading and Ordering Printed Documentation

In addition to the complete line of documentation that is delivered on your PeopleBook CD-ROM, Oracle makes PeopleSoft Enterprise documentation available to you via Oracle's website. You can:

- Download PDF files.
- Order printed, bound volumes.

Downloading PDF Files

You can download PDF versions of PeopleSoft Enterprise documentation online via the Oracle Technology Network. Oracle makes these PDF files available online for each major release shortly after the software is shipped.

See Oracle Technology Network, http://www.oracle.com/technology/documentation/psftent.html.

Ordering Printed, Bound Volumes

You can order printed, bound volumes of selected documentation via the Oracle Store.

See Oracle Store, http://oraclestore.oracle.com/OA_HTML/ibeCCtpSctDspRte.jsp?section=14021

Copyright © 1988-2007, Oracle. All rights reserved.

Additional Resources

The following resources are located on Oracle's PeopleSoft Customer Connection website:

Resource	Navigation
Application maintenance information	Updates + Fixes
Business process diagrams	Support, Documentation, Business Process Maps
Interactive Services Repository	Support, Documentation, Interactive Services Repository
Hardware and software requirements	Implement, Optimize + Upgrade; Implementation Guide; Implementation Documentation and Software; Hardware and Software Requirements
Installation guides	Implement, Optimize + Upgrade; Implementation Guide; Implementation Documentation and Software; Installation Guides and Notes
Integration information	Implement, Optimize + Upgrade; Implementation Guide; Implementation Documentation and Software; Pre-Built Integrations for PeopleSoft Enterprise and JD Edwards EnterpriseOne Applications
Minimum technical requirements (MTRs)	Implement, Optimize + Upgrade; Implementation Guide; Supported Platforms
Documentation updates	Support, Documentation, Documentation Updates
PeopleBooks support policy	Support, Support Policy
Prerelease notes	Support, Documentation, Documentation Updates, Category, Release Notes
Product release roadmap	Support, Roadmaps + Schedules
Release notes	Support, Documentation, Documentation Updates, Category, Release Notes
Release value proposition	Support, Documentation, Documentation Updates, Category, Release Value Proposition
Statement of direction	Support, Documentation, Documentation Updates, Category, Statement of Direction
Troubleshooting information	Support, Troubleshooting
Upgrade documentation	Support, Documentation, Upgrade Documentation and Scripts

Copyright © 1988-2007, Oracle. All rights reserved.

Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions.
- Visual cues.
- Country, region, and industry identifiers.
- Currency codes.

Typographical Conventions

This table contains the typographical conventions that are used in PeopleBooks:

Typographical Convention or Visual Cue	Description
Bold	Indicates PeopleCode function names, business function names, event names, system function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Italics	Indicates field values, emphasis, and PeopleSoft Enterprise or other book-length publication titles. In PeopleCode syntax, italic items are placeholders for arguments that your program must supply. We also use italics when we refer to words as words or letters as letters, as in the following: Enter the letter *O*.
KEY+KEY	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press the W key.
`Monospace font`	Indicates a PeopleCode program or other code example.
“ ” (quotation marks)	Indicate chapter titles in cross-references and words that are used differently from their intended meanings.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).

Copyright © 1988-2007, Oracle. All rights reserved.

Typographical Convention or Visual Cue	Description
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.

Visual Cues

PeopleBooks contain the following visual cues.

Notes

Notes indicate information that you should pay particular attention to as you work with the PeopleSoft Enterprise system.

Note. Example of a note.

If the note is preceded by *Important!,* the note is crucial and includes information that concerns what you must do for the system to function properly.

Important! Example of an important note.

Warnings

Warnings indicate crucial configuration considerations. Pay close attention to warning messages.

Warning! Example of a warning.

Cross-References

PeopleBooks provide cross-references either under the heading "See Also" or on a separate line preceded by the word *See.* Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.

Country, Region, and Industry Identifiers

Information that applies only to a specific country, region, or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a country-specific heading: "(FRA) Hiring an Employee"

Example of a region-specific heading: "(Latin America) Setting Up Depreciation"

Country Identifiers

Countries are identified with the International Organization for Standardization (ISO) country code.

Copyright © 1988-2007, Oracle. All rights reserved.

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in PeopleBooks:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in PeopleBooks:

- USF (U.S. Federal)
- E&G (Education and Government)

Currency Codes

Monetary amounts are identified by the ISO currency code.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like to see changed about PeopleBooks and other Oracle reference and training materials. Please send your suggestions to your product line documentation manager at Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, U.S.A. Or email us at appsdoc@us.oracle.com.

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions.

Common Elements Used in PeopleBooks

As of Date	The last date for which a report or process includes data.
Business Unit	An ID that represents a high-level organization of business information. You can use a business unit to define regional or departmental units within a larger organization.
Description	Enter up to 30 characters of text.
Effective Date	The date on which a table row becomes effective; the date that an action begins. For example, to close out a ledger on June 30, the effective date for the ledger closing would be July 1. This date also determines when you can view and change the information. Pages or panels and batch processes that use the information use the current row.

Copyright © 1988-2007, Oracle. All rights reserved.

Once, Always, and **Don't Run**	Select Once to run the request the next time the batch process runs. After the batch process runs, the process frequency is automatically set to Don't Run. Select Always to run the request every time the batch process runs. Select Don't Run to ignore the request when the batch process runs.
Process Monitor	Click to access the Process List page, where you can view the status of submitted process requests.
Report Manager	Click to access the Report List page, where you can view report content, check the status of a report, and see content detail messages (which show you a description of the report and the distribution list).
Request ID	An ID that represents a set of selection criteria for a report or process.
Run	Click to access the Process Scheduler request page, where you can specify the location where a process or job runs and the process output format.
SetID	An ID that represents a set of control table information, or TableSets. TableSets enable you to share control table information and processing options among business units. The goal is to minimize redundant data and system maintenance tasks. When you assign a setID to a record group in a business unit, you indicate that all of the tables in the record group are shared between that business unit and any other business unit that also assigns that setID to that record group. For example, you can define a group of common job codes that are shared between several business units. Each business unit that shares the job codes is assigned the same setID for that record group.
Short Description	Enter up to 15 characters of text.
User ID	An ID that represents the person who generates a transaction.

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleCode API Reference Preface

This preface introduces the PeopleCode Reference PeopleBook.

PeopleCode API Reference

This PeopleBook is the reference book for the PeopleCode APIs. PeopleCode is the proprietary language used in the development of PeopleSoft applications. Its chapters describe the syntax and fundamental elements of the PeopleCode APIs.

The "About These PeopleBooks" preface contains general product line information, such as related documentation, common page elements, and typographical conventions. This preface also contains a glossary with useful terms that are used in PeopleBooks.

See also *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide* and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*.

PeopleCode Typographical Conventions

Throughout this book, we use typographical conventions to distinguish between different elements of the PeopleCode language, such as bold to indicate function names, italics for arguments, and so on.

Please take a moment to review the following typographical cues:

Font Type	Description
`monospace font`	Indicates a PeopleCode program or other example
Keyword	In PeopleCode syntax, items in **keyword** font indicate function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Variable	In PeopleCode syntax, items in *variable* font are placeholders for arguments that your program must supply.
...	In PeopleCode syntax, ellipses indicate that the preceding item or series can be repeated any number of times.

Copyright © 1988-2007, Oracle. All rights reserved.

Font Type	Description
{*Option1*|*Option2*}	In PeopleCode syntax, when there is a choice between two options, the options are enclosed in curly braces and separated by a pipe.
[ ]	In PeopleCode syntax optional items are enclosed in square brackets.
&Parameter	In PeopleCode syntax an ampersand before a parameter indicates that the parameter is an already instantiated object.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 1

AESection Class

This chapter provides an overview of AESection class and discusses the following topics:

- How an AESection is accessed.
- AESection example.
- Data type of an AESection object.
- Scope of an AESection object.
- AESection Class built-in function.
- AESection Class reference.

Understanding AESection Class

Before PeopleTools 8, users could perform SQL directly on the Application Engine tables, thereby changing their SQL and Application Engine "flow" in a dynamic manner, prior to running their applications. Some applications, for example, let the user input their "rules" in a user-friendly application, then convert these rules, at save time, into Application Engine constructs.

With PeopleTools 8, Application Engine programs should not perform SQL directly on the Application Engine tables, as they are system tables, and are cached. SQL on the Application Engine tables may not be accurately reflected when the applications are executed, because of the caching mechanism.

To overcome this problem, developers have two basic operations that let them modify their Application Engine programs from their online pages:

- Ability to modify SQL definitions, which are referenced within Application Engine SQL using the PeopleCode SQL class and the meta-SQL function %SQL.
- Ability to dynamically change the execution flow of a given Application Engine section.

Use the AESection class to do the latter. This section object is used to modify the steps and SQL associated with a given section by using PeopleCode.

Before describing the specifics of the AESection class, the following are some general terms to understand:

- *base* section

  The *base* section represents the Application Engine section you are working on. This is the section that you are changing in PeopleCode. In other words, it's the target section.

- *template* section

  The *template* section represents the Application Engine section that is the source, or "model" for the section you're building. You copy *from* the template section *to* the base section.

Copyright © 1988-2007, Oracle. All rights reserved.

The template must exist in the database before you can use it.

If the base section you specify doesn't exist, a *new* base section is created. This enables you to dynamically create sections as needed.

You can copy steps (and their attributes) from the template to the base. The only attribute of the step you can modify is the SQL statement that gets executed.

Warning! When you open or get an AESection object, (that is, the base section) any existing steps in the section are deleted. You must add a new step to the section before you can modify it.

The main assumption for this class is that your rules are dynamic primarily in the SQL that they execute, but that the structure of the rules are static, or at least defined well enough that a standard template can be applied.

Note that you can't update the Application Engine *actions* that are not SQL-related (that is, PeopleCode, Call Section, or Log Message). In other words, you can't change the PeopleCode associated with a PeopleCode action within a step. You can add a step containing a PeopleCode action to your new section, but you can't change the PeopleCode dynamically.

See Also

Chapter 38, "SQL Class," page 1563

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "Meta-SQL," %SQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Engine, "Creating Application Engine Programs," Adding Steps

How an AESection is Accessed

When an AESection is opened (or accessed), the system first checks if it exists with the given input parameters.

If the base section you specify doesn't exist, a *new* base section is created. This enables you to dynamically create sections as needed. In addition, if the target section doesn't exist, all section-level attributes are copied from the template to the target section. If the target exists, it retains its attribute settings.

If an effective date is specified (with EffDt), but there is no match using that effective date, the AESection is opened using the base effective date of '1900-01-01'.

The market defined for the current component is used for the section to be open. If no section with this market exists, the default GBL is used.

If you open an AESection object from within a running Application Engine program, the market value is set to the value of the current process.

The database platform you're currently running is used as the database platform for the section to be opened. If no section with this database platform exists, the default "none" is used.

To find the correct section to open, the precedence is:

1. Market
2. Database platform
3. Effective date

 Copyright © 1988-2007, Oracle. All rights reserved.

When a section is closed and its changes are saved, the market and database platform from the system that performed the changes is used as the market and database platform for the modified section.

Let's take an example. Assume that you're running on DB2 UDB for OS/390 and z/OS, and your current market is set to 'MKT'. Assume that you want to open a section called Sect1 in an Application Engine program called TestAppl, and that your data looks something like this:

Number	Application Engine Program	Application Engine Section	Market	Platform	Effective Date
1	TestAppl	Sect1	USA	DB2	1990-01-01
2	TestAppl	Sect1	USA	None	1900-01-01
3	TestAppl	Sect1	GBL	None	1900-01-01

If you use:

```
&SECTION.Open("TestAppl", "Sect1", "1998-12-14");
```

Then the third section in the previous list is used, because the market takes highest precedence. When the section is saved, the values for market and platform are updated to the current system values.

Warning! When you open or get an AESection object, (that is, any base section) any existing steps in the section are deleted. You must add a new step to the section before you can modify it.

The other attributes of the section, however, are retained, with the exceptions noted previously.

AESection Example

Assume that you have a template section called TEMPLATE in the Application Engine program called MY_APPL. The template looks like this:

Steps	Actions
NewStep1	DO When DO Select SQL
NewStep2	DO Select CallSection

Copyright © 1988-2007, Oracle. All rights reserved.

Also, assume that you have a base section called DYN_SECT in the Application Engine program called RULES. When you start, this section looks like this:

Steps	Actions
Step1	DO Select Call Section DO Unit
Step2	Call Section

Here's the PeopleCode:

```
Local AESection &Section;
&Section = GetAESection("RULES", "DYN_SECT");
/* Open the base section */
&Section.SetTemplate("MY_APPL", "TEMPLATE");
/* Set the template section */
&Section.AddStep("NewStep2");
/* Insert NewStep2     */
/* Do some SQL stuff here  */
&Section.SetSQL("DO_SELECT", &MySql);
/* Modify the SQL in the added step */
&Section.Save();
&Section.Close();
/* Save and close   */
```

The base section looks like this after execution:

Steps	Actions
NewStep2	DO Select Call Section

Note. The existing steps in the base section have been *overwritten* by the new step from the template section.

Data Type of an AESection Object

You should declare an AESection object as type AESection. For example:

```
Local AESection &SECTION;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Scope of an AESection Object

An AESection object can only be instantiated from PeopleCode.

The AESection Object is designed for use within online pages. Typically, dynamic sections should be constructed in response to an end-user action.

Note. Do not call an AESection object from an Application Engine PeopleCode Action. If you need to access another section, use CallSection Actions instead.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Engine, "Creating Application Engine Programs," Specifying Call Section Actions

AESection Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

AESection Class Methods

In this section, we discuss each AESection class method, in alphabetical order.

AddStep

Syntax

```
AddStep(ae_step_name [, NewStepName])
```

Description

The given step name from the template section is added as the next step into the base section, and named the existing step name.

Note. When you open or get a section, all the existing steps are deleted. The first time you execute AddStep, you add the first step. The second time you execute AddStep, you add the second step, and so on.

All attributes of the step are copied, including all of its actions. The only changeable attribute of a step are its SQL statements, which can be modified using the SetSQL method. SetSQL is run on the *current* step, that is, the step most recently added.

You can also change the name of the step by using the optional parameter *NewStepName*.

If the step named does not exist in the template, an error occurs. Likewise, if you have not opened or set the template for the section, you receive an error.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ae_step_name	Specify the step name from the template to be added as the *next* step in the base section. This parameter takes a string value.
NewStepName	Specify a new name for the step to be added. This parameter is optional, and takes a string value.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," SetSQL, page 8

Close

Syntax

```
Close()
```

Description

Close closes the AESection object. Any unsaved changes to the section are discarded.

Parameters

None.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," Save, page 8 and Chapter 1, "AESection Class," Open, page 6.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

Open

Syntax

```
Open(ae_applid, ae_section, [effdt])
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Open method associates the AESection object with the given Application Engine section, based on the *ae_applid* and *ae_section*. If the *effdt* is specified, this is also used to get the section object. In other words, the Open method sets the base section.

Warning! When you open or get an AESection object, (that is, the base section) any existing steps in the section are deleted.

Note. If the base section you specify doesn't exist, a *new* base section is created. This enables you to dynamically create sections as needed. In addition, if the target section doesn't exist, all section-level attributes are copied from the template to the target section. If the target exists, it retains its attribute settings.

If the AESection is still open when you issue the Open method, the previously opened object is discarded and none of the changes saved. To prevent accidentally discarding your changes, you can use the IsOpen property to verify if a section is already open.

The AESection is open based on the Market and database type of your current system.

Parameters

Parameter	Description
ae_applid	Specify the application ID of the section you want to access. This parameter takes a string value.
ae_section	Specify the section name of the section you want to access. This parameter takes a string value.
effdt	Specify the effective date of the section you want to access (optional). This parameter takes a string value.

Returns

An AESection object.

Example

```
Local AESection &SECTION;
&SECTION = GetAESection("RULES1", "DYN_SECT");
/* do some processing */
&SECTION.Close();
&SECTION.Open("RULES2", "DYN_SECT");
/* do additional processing */
```

See Also

Chapter 1, "AESection Class," IsOpen, page 10

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

Copyright © 1988-2007, Oracle. All rights reserved.

Save

Syntax

```
Save()
```

Description

The Save method saves the section to the database. The AESection object remains open after you use this method. To close the object, you must use the Close method.

You must commit all database changes prior to using this method. This is to avoid locking critical Tools tables and hence freezing all other users. You receive a runtime error message if you try to use this method when there are pending database updates, and your PeopleCode program terminates. You need to commit any database updates prior to using this method. Use the CommitWork function to commit database updates.

Parameters

None.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," Close, page 6 and Chapter 1, "AESection Class," Open, page 6.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CommitWork

SetSQL

Syntax

```
SetSQL(action_type_string, string)
```

Description

The SetSQL method replaces the SQL associated with the given action type in the current step in the base section with the SQL in *string*. The *current* step is the latest step that was added using AddStep. The action types are:

- DO_WHEN
- DO_WHILE
- DO_SELECT
- SQL
- DO_UNTIL

Note. All action types must be passed in as strings with quotation marks.

If the action specified does not exist in the current step, an error occurs.

 Copyright © 1988-2007, Oracle. All rights reserved.

You can use a SQL object as *string*.

```
&SECTION.SetSQL("SQL", &SQL);
```

Parameters

Parameter	Description
action_type	Specifies the action type of the current step that should be changed. This parameter takes a string value.
string	Specifies the SQL to be used to replace the SQL in the current step.

Returns

None.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," AddStep, page 5

Chapter 38, "SQL Class," page 1563

SetTemplate

Syntax

SetTemplate(*ae_applid, ae_section*)

Description

The SetTemplate method sets the template to be used with an AESection object, as identified by *ae_applid* and *ae_section*.

The rules for assigning a template section are similar to the rules for selecting a base section. The selection of market and database platform are done exactly the same. However, the effective date of the template is always set based on the effective date of '1900-01-01'.

You must set the template before you can use any of the other methods.

Parameters

Parameter	Description
ae_applid	Specify the name of the Application Engine program that contains the section to be used as the template.
ae_section	Specify the name of the Application Engine section in the program to be used as the section template.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

See Chapter 1, "AESection Class," AESection Example, page 3.

See Also

Chapter 1, "AESection Class," Open, page 6

Chapter 1, "AESection Class," How an AESection is Accessed, page 2

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAESection

AESection Class Property

In this section, we discuss the IsOpen AESection property.

IsOpen

Description

If this property is True, the section object is already open. If you try to open a section object that is already open, the open section object is closed and all the changes that haven't been saved are discarded before the object is re-opened.

Example

```
If Not(&MYSECTION.IsOpen) Then
   &MYSECTION.Open(MYAPPLID, SECTION2);
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 2

Analytic Calculation Engine Classes

This chapter provides an overview of the Analytic Calculation Engine classes, and discuses:

- Using the Analytic Calculation Engine classes with PeopleSoft Application Engine
- Running synchronously
- Error handling
- Data types for the Analytic Calculation Engine classes
- Scope of the Analytic Calculation Engine class objects
- Analytic Calculation Engine reference

Understanding the Analytic Calculation Engine Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the Analytic Grid classes to manipulate the display of analytic calculation data on a page.
- Use the Analytic Type classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” page 43

Chapter 4, “Analytic Grid Classes,” page 129

Chapter 5, “Analytic Type Classes,” page 137

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine

Copyright © 1988-2007, Oracle. All rights reserved.

Using the Analytic Calculation Engine Classes with PeopleSoft Application Engine

All PeopleSoft Application Engine programs run on the application engine server. This means your program runs on the application engine server, not the analytic server.

PeopleSoft recommends that you put each of the following methods into its own step when you're creating an analytic calculation using a PeopleSoft Application Engine program:

- AnalyticInstance Load method
- AnalyticInstance Unload method
- AnalyticInstance Copy method
- AnalyticInstance Delete method (in case of any readable/writable records in the analytic type)
- AnalyticModel Recalculate method
- CubeCollection SetData method

Also, with the Recalculate method, you should use the Commit after step option.

Running Synchronously

Some of the methods and functions associated with these classes have the option of being run asynchronously. When you run these asynchronously, and not from a PeopleSoft Application Engine program, these methods and functions have additional functionality, that is, if the RenServer is configured, a window displays the PeopleSoft Analytic Calculation Engine execution status.

Using Trees

PeopleSoft Analytic Calculation Engine uses trees to establish hierarchies of a dimension's parent-child relationships. PeopleSoft Analytic Calculation Engine uses these hierarchies to:

- Calculate and display aggregated data to end users.
- Enable end users to navigate through data by performing such actions as expanding and collapsing nodes.
- Enable end users to drill down and drill up through data.

It is important to understand that PeopleSoft trees and hierarchies differ in the following manner: You create one tree for each dimension that requires a hierarchy. The analytic model uses that tree to create one hierarchy for one dimension.

Use the AttachTree method to attach a tree to a dimension, and use the DetachTree method to detach the tree from the dimension.

You should be aware of the following restrictions:

- Because the AttachTree method attaches a specific tree to an analytic instance, the system throws an error if the tree's name, setID or effective date is incorrect.
- You can attach only one tree to one dimension at a time.

 Copyright © 1988-2007, Oracle. All rights reserved.

- If the analytic instance is already loaded into the analytic server, the tree isn't attached until the analytic model is loaded.
- No aggregate data is displayed to the user after you use the DetachTree method.
- If your application loads the analytic model after the tree has been detached, the analytic model doesn't create a hierarchy for the dimension.
- If the analytic instance is already loaded into the analytic server, the DetachTree method isn't applied to the tree until the next time your application loads the analytic instance.

You can only attach a tree to an analytic model before the analytic instance

No aggregate data displayed to user after detach tree.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Hierarchies"

Error Handling

All the Analytic Calculation Engine classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

Using the Messages Property

Use the Messages property to determine whether any errors occurred. The Messages property returns a multi-dimensional array. You can use the Len property of the array class to determine the length of the array. If the array has a length of 0, there are no errors.

Data Types for Analytic Calculation Engine Classes

Every PeopleSoft Analytic Calculation Engine object is declared as its own data type, that is, AnalyticModel objects are declared as type AnalyticModel, CubeCollection objects are declared as type CubeCollection, and so on.

The following are the data types for the PeopleSoft Analytic Calculation Engine classes:

- AnalyticInstance
- AnalyticModel
- CubeCollection

Scope of Analytic Calculation Engine Classes

The Analytic Calculation Engine objects can only be instantiated from PeopleCode.

Copyright © 1988-2007, Oracle. All rights reserved.

These objects can be used anywhere you have PeopleCode, that is, in a PeopleSoft Application Engine program, an application class, record field PeopleCode, and so on.

Analytic Calculation Engine objects can be of scope Local, Component, or Global.

Analytic Calculation Engine Classes Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateAnalyticInstance

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetAnalyticInstance

AnalyticInstance Class Methods

In this section, we discuss the AnalyticInstance methods. The methods are discussed in alphabetical order.

CheckAsyncStatus

Syntax

CheckAsyncStatus(*OperationID*)

Description

Use the CheckAsyncStatus method to determine the current state of the asynchronous methods, as well as the Load method when executed asynchronously.

Parameters

Parameter	Description
OperationID	Specify the operation ID of the asynchronous operation. This operation ID is returned when one of the asynchronous operations is executed.

Returns

This method returns one of the following values:

Value	Description
Less than 0	The operation completed with errors.
0	The operation has not completed yet.
Greater than 0	The operation completed successfully.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CheckStatus, page 15 , Chapter 2, "Analytic Calculation Engine Classes," Load, page 17 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 20.

CheckStatus

Syntax

```
CheckStatus()
```

Description

Use the CheckStatus method to determine the current state of the analytic calculation engine.

Parameters

None.

Returns

This method returns one of the following values:

Value	Description
%AnalyticInstance_NotLoaded	The analytic instance has not been loaded into the application server.
%AnalyticInstance_Loading	The analytic instance is being loaded.
%AnalyticInstance_Idle	The analytic instance is idle, that is, not currently running.
%AnalyticInstance_Busy	The analytic instance is currently running.
%AnalyticInstance_ Inaccessible	The analytic instance is currently inaccessible.
%AnalyticInstance_ Terminating	The analytic instance is currently ending.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Load, page 17 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 20.

Copy

Syntax

```
Copy(NewID, ForceDelete&Record)
```

Description

Use the Copy method to copy all the data and metadata to a new analytic instance from the current analytic instance.

If the specified *NewID* exists, or any data for *NewID* exists, the data is not copied unless *ForceDelete* has been specified as true.

If a tree is attached to the existing analytic instance, all tree data is also copied to the new analytic instance.

Copyright © 1988-2007, Oracle. All rights reserved.

In addition, the analytic instance and the tree information are not copied unless the record specified with the *&Record* parameter is populated with the existing analytic instance ID.

Parameters

Parameter	Description
NewID	Specify the new analytic instance ID as a string.
ForceDelete	If the specified *NewID* exists, and *ForceDelete* is specified as false, the copy is cancelled. If the specified *NewID* exists, and *ForceDelete* is specified as true, the analytic instance specified by *NewID* is deleted, and a new analytic instance is created from the current ID.
&Record	Specify an already instantiated record object to pass in values to the Copy application package method that's defined with the specified analytic type definition.

Returns

None.

Example

```
&ai.Copy("MYTESTCOPY", True, &MyRecord);
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," Delete, page 16 , Chapter 2, "Analytic Calculation Engine Classes," Load, page 17 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 20.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions"

Delete

Syntax

Delete(*ForceUnload*, *&RecordRef*)

Description

Use the Delete method to remove all the metadata related to the given analytic instance.

Every analytic type definition is defined with an application package that contains three methods: Create, Delete, and Copy. The values in *&RecordRef* are passed to the Delete method.

This method does not delete the analytic instance if the analytic instance is currently loaded. Specify *ForceUnload* as true to unload the analytic instance before deleting it.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ForceUnload	Specify whether to force the deletion of the analytic instance. This parameter takes a boolean value. If the analytic instance is currently loaded, specifying true for this parameter forces the analytic instance to be unloaded before it is deleted.
&RecordRef	Specify a record to pass in values to the application package Delete class that's associated with the analytic type definition.

Returns

None.

Example

```
&ai.Delete(TRUE, &Record);
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," Unload, page 20

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions"

GetAnalyticModel

Syntax

GetAnalyticModel(**Model.***ModelName*)

Description

Use the GetAnalyticModel method to return a reference to an AnalyticModel object.

Parameters

Parameter	Description
ModelName	Specify the name of the analytic model for which you want to return a reference. The model definition must already exist in PeopleSoft Application Designer.

Returns

A reference to an AnalyticModel object or Null.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AnalyticModel Class Methods, page 21

Load

Syntax

Load(*Sync*, *IdleTimeOut*, **Message.***messagename*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the Load method to load the AnalyticInstance object executing the method into the analytic server.

If this analytic instance is already loaded, this method fails.

Parameters

Parameter	Description
Sync	Specify whether the analytic instance should be run synchronously or asynchronously. This parameter takes a boolean value: true if it should be run synchronously, false otherwise. See Chapter 2, "Analytic Calculation Engine Classes," Running Synchronously, page 12.
IdleTimeOut	Specify, in minutes, the idle time out value. A value of 0 is an infinite time out. A value of -1 indicates that it should use the value specified in the default configuration for the analytic server. **Note.** When loading an analytic grid, you should always specify a value of -1. See *Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration*, "Managing Analytic Servers," Loading and Unloading Analytic Instances.
Message. *MessageName*	Specify an application message that should be sent if the analytic server crashes while the analytic instance is loaded. **Note.** The message is sent when the analytic server process restarts itself after crashing.

Returns

A string. For asynchronous loads, this string is passed to the CheckAsyncStatus method.

This method returns one of the following values:

Value	Description
%AnalyticInstance_NotLoaded	The analytic instance has not been loaded into the analytic server.
%AnalyticInstance_Loading	The analytic instance is being loaded.
%AnalyticInstance_Idle	The analytic instance is idle, that is, not currently running.
%AnalyticInstance_Busy	The analytic instance is currently running.
%AnalyticInstance_ Inaccessible	The analytic instance is currently inaccessible.
%AnalyticInstance_ Terminating	The analytic instance is currently terminating.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CheckAsyncStatus, page 14 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 20.

Copyright © 1988-2007, Oracle. All rights reserved.

RunAsync

Syntax

```
RunAsync()
```

Description

Use the RunAsync method to specify if the analytic instance executing the method should be run in an asynchronous manner.

This method is used only to define the transaction in the analytic type definition in optimization. PeopleSoft Analytic Calculation Engine programs uses the Load method for defining synchronous or asynchronous operation.

Parameters

None.

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions

RunSync

Syntax

```
RunSync()
```

Description

Use the RunSync method to specify if the analytic instance executing the method should be run in a synchronous manner.

This method is used only to define the transaction in the analytic type definition in optimization. PeopleSoft Analytic Calculation Engine programs uses the Load method for defining synchronous or asynchronous operation.

Parameters

None.

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions

Copyright © 1988-2007, Oracle. All rights reserved.

Terminate

Syntax

```
Terminate()
```

Description

Use the Terminate method to force the termination of an analytic instance loaded in an analytic server.

This method should only be used from PeopleCode running in a PeopleSoft Application Engine program to cause an analytic instance loaded in an analytic server to be terminated.

Attempting to terminate an analytic instance that is not loaded or is loaded in a PeopleSoft Application Engine process results in a PeopleCode exception.

Although the Terminate method returns instantly, it may take up to a minute before the analytic instance is actually terminated.

Parameters

None.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Load, page 17 and Chapter 2, "Analytic Calculation Engine Classes," Unload, page 20.

Unload

Syntax

```
Unload()
```

Description

Use the Unload method to unload the analytic instance executing the method from the analytic server. After unloading the analytic instance, the analytic server process is restarted.

Parameters

None.

Returns

None.

Example

```
&ai.UnLoad();
```

See Also

Chapter 2, "Analytic Calculation Engine Classes," Load, page 17

 Copyright © 1988-2007, Oracle. All rights reserved.

AnalyticInstance Class Properties

In this section, we discuss the AnalyticInstance properties. The properties are discussed in alphabetical order.

AnalyticType

Description

This property returns the name of the analytic type definition for this AnalyticInstance object as a string.

This property is read-only.

ID

Description

This property returns the analytic instance ID of this AnalyticInstance object as an integer.

This property is read-only.

Messages

Description

This property returns a multi-dimensional array of any, containing the messages that occurred.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

After you access this property, only new messages are returned, that is, messages are only returned once.

This property is read-only.

See Also

Chapter 2, “Analytic Calculation Engine Classes,” RunSync, page 19

AnalyticModel Class Methods

In this section, we discuss the AnalyticModel methods. The methods are discussed in alphabetical order.

AddMember

Syntax

AddMember(*DimName, MemberName*)

Description

Use the AddMember method to add the specified dimension member to the specified dimension.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the dimension to which you want to add the member.
MemberName	Specify the name of the member that you want to add to the specified dimension.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," GetMembers, page 26 and Chapter 2, "Analytic Calculation Engine Classes," RenameMember, page 29.

AttachTree

Syntax

AttachTree(*DimName*, *TreeName*, *SetID*, *UserKeyValue*, *EffDt*, *NodeName*, *OverrideRecord*, *DetailStartLvl*, *TreeDiscardLvl*)

Description

Use the AttachTree method to attach a tree to a dimension. Only one tree can be attached to a dimension at a time. You can only attach a tree before the analytic instance is loaded.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to attach a tree to.
TreeName	Specify the name of the tree that you want to attach to the dimension.
SetId	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter
EffDt	Specify an effective date for this tree. This parameter takes a string value.
NodeName	Specify the tree node used as the dimension hierarchy root. This can be different from the tree root, that is, you can pick a subtree.
OverrideRecord	Specify the name of a record to be used for overriding the aggregate. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine*, "Creating Hierarchies," Working with Overrides.
DetailStartLvl	Specify the tree level number (such as 1 for the node hierarchy root specified by the *NodeName* parameter) at which detail nodes start. If you specify a value other than 0, any nodes at this level or greater than this level are considered details node, and any nodes with levels less than the start level are considered aggregate nodes If you specify 0, detail nodes are nodes that don't have parents (a node can be a parent either by having children or by having child ranges).
TreeDiscardLvl	Specify the tree level number (such as 1 for the node hierarchy root specified by the *NodeName* parameter) at which to stop loading the tree. Nodes at this level or at levels greater are discarded. This parameter is only used if it has a value other than 0, and if it has a value greater than *DetailStartLvl*.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," DetachTree, page 24 and Chapter 2, "Analytic Calculation Engine Classes," GetTree, page 27.

Chapter 40, "Tree Classes," page 1609

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Hierarchies," Understanding the Relationship of PeopleSoft Trees to Analytic Models

Copyright © 1988-2007, Oracle. All rights reserved.

CalculateCube

Syntax

CalculateCube(*CubeName*[, *Sync*])

Description

Use the CalculateCube method to calculate the named cube in synchronous mode.

Parameters

Parameter	Description
CubeName	Specify the name of the cube that you want to calculate.
Sync	This parameter is optional and is not used. It is only being included for previous releases. This parameter takes a boolean value. The value of this parameter is ignored. The cube is always calculated in synchronous mode.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Running Synchronously, page 12 and Chapter 2, "Analytic Calculation Engine Classes," Recalculate, page 28.

DetachTree

Syntax

DetachTree(*DimName*)

Description

Use the DetachTree method to detach a tree from the specified dimension.

You can only attach a tree when the analytic instance is not loaded. This method fails if you try to detach a tree while the analytic instance is loaded.

Parameters

Parameter	Description
DimName	Specify the name of the dimension from which you want to detach a tree.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AttachTree, page 22 and Chapter 2, "Analytic Calculation Engine Classes," GetTree, page 27.

Chapter 40, "Tree Classes," page 1609

 Copyright © 1988-2007, Oracle. All rights reserved.

GetCubeCollection

Syntax

GetCubeCollection(*CubeCollName*)

Description

Use the GetCubeCollection method to return a reference to a CubeCollection object.

Parameters

Parameter	Description
CubeCollName	Specify the name of the cube collection to which you want a reference.

Returns

A CubeCollection object if successful.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CubeCollection Class, page 30

GetCellProperties

Syntax

GetCellProperties(*CubeName, &Node*)

Description

Use the GetCellProperties method to get information about a cell.

Parameters

Parameter	Description
CubeName	Specify the name of the cube that contains the cell you want information about.
&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail.

Returns

An array of string.

The four strings returned in the array are:

1. Cell type. Values are "Aggregate" or "Detail"
2. User function.
3. Dimension for aggregation, the dimension name.
4. Aggregation reason. Values are: "Dimension Level Override", "Member Level Override", "Cube Dimension Level Override", "None."

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 8, “Array Class,” page 215

GetMembers

Syntax
GetMembers(*DimName, DimFilter*)

Description

Use the GetMembers method to return the names of the members in the specified dimension. You can specify a user function to be used as a filter for the dimension using *DimFilter*.

When filtering hierarchy nodes, if the parent member is filtered, all the child nodes are also filtered.

In order to filter the parent member if all of its children are filtered, the user function should be written such that for the parent node, apply the condition on all of its children, using the FORCHILDREN built-in function. If none of the children satisfy the condition, return zero from the user function, which filters the parent node.

Parameters

Parameter	Description
DimName	Specify the name of the dimension from which you want to get the members.
DimFilter	Specify a user function that you want to use to filter the dimension.

Returns

A two-dimensional array of any.

The first dimension contains the member names. The second dimension contains the parent of the member. The members are sorted in the hierarchy order.

If the dimension does not have a hierarchy, all members are returned as level one.

For example, if the dimension has the following members:

```
ALLREGIONS
   NORTH AMERICA
      USA
      CANADA
      MEXICO
   EUROPE
      FRANCE
      GERMANY
      ENGLAND
   ASIA
      JAPAN
      CHINA
```

The GetMembers method returns the following:

 Copyright © 1988-2007, Oracle. All rights reserved.

Member Name	Level
ALLREGIONS	Null
NORTHAMERICA	ALLREGIONS
USA	NORTHAMERICA
CANADA	NORTHAMERICA
MEXICO	NORTHAMERICA
EUROPE	ALLREGIONS
FRANCE	EUROPE
ENGLAND	EUROPE
GERMANY	EUROPE
ASIA	ALLREGIONS
JAPAN	ASIA
CHINA	ASIA

If the total member is present in the metadata, it is returned as level zero.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AddMember, page 21 and Chapter 2, "Analytic Calculation Engine Classes," RenameMember, page 29.

GetTree

Syntax

GetTree(*DimName*)

Description

Use the GetTree method to return an array of string that contains information about the tree that is attached to the specified dimension.

Parameters

Parameter	Description
DimName	Specify the name of the dimension to which a tree is attached.

Returns

An array of string.

The array items have the following format:

Copyright © 1988-2007, Oracle. All rights reserved.

1. TreeName
2. SetID
3. UserKeyValue
4. EffDt
5. BranchName
6. Override record name
7. DetailStartLvl
8. TreeDiscardLvl

See Also

Chapter 2, "Analytic Calculation Engine Classes," AttachTree, page 22 and Chapter 2, "Analytic Calculation Engine Classes," DetachTree, page 24.

Chapter 40, "Tree Classes," page 1609

Recalculate

Syntax

Recalculate(*Sync*)

Description

Use the Recalculate method to recalculates the loaded analytic model. If *Sync* is true, the transaction is synchronous, else it is performed asynchronously.

The Recalculate method always writes any changes to readable, readable and writable, as well as writable records to the database as specified by the analytic type definition.

If you are running in asynchronous mode, this method returns an optional string that contains the operation ID to be used with the CheckAsyncStatus property.

Parameters

Parameter	Description
Sync	Specify whether the recalculation should be run synchronously or asynchronously. This parameter takes a boolean value: true if it should be run synchronously, false otherwise.

Returns

A string.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CalculateCube, page 24 and Chapter 2, "Analytic Calculation Engine Classes," CheckAsyncStatus, page 14.

Chapter 3, "Analytic Calculation Engine Metadata Classes," AnalyticModelDefn, page 47

 Copyright © 1988-2007, Oracle. All rights reserved.

RenameMember

Syntax

RenameMember(*DimName*, *OrigMemberName*, *NewMemberName*)

Description

Use the RenameMember method to rename a member in the specified dimension.

Parameters

Parameter	Description
DimName	Specify the dimension that contains the member you want to rename.
OrigMemberName	Specify the name of the member that you want to rename.
NewMemberName	Specify the new name for the member specified by *OrigMemberName*.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AddMember, page 21 and Chapter 2, "Analytic Calculation Engine Classes," GetMembers, page 26.

AnalyticModel Class Property

In this section, we discuss the AnalyticModel class property.

Messages

Description

This property returns a multi-dimensional array of any containing the messages that occurred during execution of the Recalculate method.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

After you access this property, only new messages are returned, that is, messages are only returned once.

This property is read only.

See Also

Chapter 2, "Analytic Calculation Engine Classes," Recalculate, page 28

Copyright © 1988-2007, Oracle. All rights reserved.

CubeCollection Class

Instantiate a CubeCollection object using the AnalyticModel class GetCubeCollection method.

In cube collections, the total member behaves just as the root of the hierarchy. If it's present, it's displayed first in the list.

Filtering of members in a dimension in the cube collection is similar to filtering members using the AnalyticModel class GetMembers method. If you filter the parent node, the children of that node are filtered as well.

See Chapter 2, "Analytic Calculation Engine Classes," GetCubeCollection, page 25.

CubeCollection Class Methods

In this section we discuss the CubeCollection class methods. The methods are described in alphabetical order.

CollapseNode

Syntax

CollapseNode(*DimName, &Node*)

Description

Use the CollapseNode method to collapse a node in the dimension specified by *DimName*.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that contains the node you want to collapse.

&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail. For example:
	`PRODUCTS Fax Modem`
	`REGION East`

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," ExpandNode, page 32

 Copyright © 1988-2007, Oracle. All rights reserved.

DrillIntoNode

Syntax

DrillIntoNode(*DimName, &Node*)

Description

Use the DrillIntoNode method to display more data for the specified node and dimension. This is very similar to expanding, except that when you expand a node, the top level (node) is still present. When you drill into a node, the top level or levels are no longer displayed.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that contains the node that you want to drill into.

&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail. For example:
	`PRODUCTS Fax Modem`
	`REGION East`

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," DrillOutOfNode, page 31

DrillOutOfNode

Syntax

DrillOutOfNode(*DimName, &Node*)

Description

Use the DrillOutOfNode method to collapse the data that you just expanded.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that contains the node you want to drill out of.

&Node	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail. For example:
	`PRODUCTS Fax Modem`
	`REGION East`

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," DrillIntoNode, page 31

ExpandNode

Syntax

ExpandNode(*DimName*, *&Nodes*, *ExpandAll*)

Description

Use the ExpandNode method to expand the nodes in the specified dimension.

Parameters

Parameter	Description
DimName	Specify the name of the dimension contains the node that you want to expand.
&Nodes	Specify an already instantiated array of array of string (a two-dimensional array of string) containing name-value pairs of the node name and the node detail.
ExpandAll	Specify whether to expand the sub-nodes or not. This parameter takes a boolean value; true to expand the nodes, false to not expand the nodes.

Returns

None.

Example

In the following example, both the Accounts and Dept dimensions have hierarchies.

```
Account             Dept

All Acoounts            All Depts
         | ---- 100               |  - ---- A
         |                        |
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
                | ---- 200                |------   B -| ---D
          |                     |
                |                     | --- E
                | ---- 300
```

When the data is displayed, it will be as follows:

```
All Accounts
          All Depts
     100
          All Depts
     200
          All Depts
     300
          All Depts
```

When you want to expand All Depts under Account 100 , use the following PeopleSoft Analytic Calculation Engine code:

```
BAMCoordinate coord;
 coord.SetValue( "Account", "100");
 coord.SetValue("Dept" , "All Depts")
 ExpandNode("Dept", coord, false)
```

The result of this will have the following:

```
All Accounts
          All Depts
     100
          All Depts
                    A
                    B
     200
          All Depts
     300
          All Depts
```

See Also

Chapter 2, “Analytic Calculation Engine Classes,” CollapseNode, page 30

GetData

Syntax

GetData(*&DataRowset*, *StartRow*, *EndRow*[, *NetChanges*])

Description

Use the GetData method to populate a rowset with data from the cube collection.

Copyright © 1988-2007, Oracle. All rights reserved.

DataRowset is the rowset into which is put the data. If you specify an already instantiated rowset, the given rowset is populated with the data. If you specify a rowset that has not been instantiated, a new rowset is created, with the record field definition of the desired cube collection.

Parameters

Parameter	Description
&DataRowset	Specify the rowset into which to put the data. If you specify an already instantiated rowset, the given rowset is populated with the data. If you specify a rowset that has not been instantiated, a new rowset is created, with the record field definition of the desired cube collection.
StartRow	Specify the first row of data to be retrieved, as an integer.
EndRow	Specify the last row of data to be retrieved, as an integer. If you specify a zero for this parameter, all rows are retrieved.
NetChanges	Specify whether all values within the range are returned, or whether just values changed since the last time the Recalculate method was used. This parameter takes a boolean value: true, return all values, false, return just the changed values. The default value is false.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetData, page 37 and Chapter 2, "Analytic Calculation Engine Classes," Messages, page 42.

Chapter 33, "Rowset Class," page 1421

GetDimFilter

Syntax

GetDimFilter(*DimName*)

Description

Use the GetDimFilter method to return the name of the current user function (filter) used with the specified dimension.

Parameters

Parameter	Description
DimName	Specify the name of the dimension for which you want the associated user function (filter).

Returns

A string containing the name of the user function. If there is no filter applied to the dimension, this method returns an empty string.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimFilter, page 38

GetDimSort

Syntax

GetDimSort(*DimName*)

Description

Use the GetDimSort to return the current sort settings on the specified dimension.

Parameters

Parameter	Description
DimName	Specify the name of the dimension for which you'd like the sort settings.

Returns

An array of any. The array contains the same structure as the signature for the SetDimSort method.

Value	Description
Array index 1	IsAscending as Boolean
Array index 2	Key1 as string
Array index 3	IsAscending2 as Boolean
Array index 4	Key2 as string
Array index 5	IsAscending3 as Boolean
Array index 6	Key3 as string

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimSort, page 38

GetLayout

Syntax

GetLayout(*&SlicerArray, &RowAxisArray, &ColumnAxisArray*)

Description

Use the GetLayout method to return the current layout for the three axes, (slice, row, and column) on the specified dimension.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&SlicerArray	Specify an already instantiated array of string containing the list of fields used on the slice axis.
&RowAxisArray	Specify an already instantiated array of string containing the list of fields used on the row axis.
&ColumnAxisArray	Specify an already instantiated array of string containing the list of fields used on the column axis.

Returns

None.

GetRowCount

Syntax

```
GetRowCount()
```

Description

Use the GetRowCount method to return the total number of rows in the cube collection.

The number of rows is a function of the hierarchy, that is, what is shown or not shown, and filtered values.

Parameters

None.

Returns

An integer.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” GetDimension, page 62

GetSlice

Syntax

```
GetSlice(&SliceRecord)
```

Description

Use the GetSlice method to return the current slice values of all the dimensions.

Parameters

Parameter	Description
&SliceRecord	Specify an already instantiated record object to hold the values of the slicer record.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 2, “Analytic Calculation Engine Classes,” SetSlice, page 39

SetData

Syntax

```
SetData(&DataRowset)
```

Description

Use the SetData method to populate the data in the cube collection with the data in the specified rowset.

The system extracts only changed values from the rowset, that is, values marked as updated. .

SetData method also writes the changes to the database for any record marked as writable or readable and writable in the analytic type definition, and mapped to the specified cube collection.

Parameters

Parameter	Description
&DataRowset	Specify the rowset object that contains the data you want to use to populate the cube collection.

Returns

None.

See Also

Chapter 2, “Analytic Calculation Engine Classes,” GetData, page 33

Chapter 32, “Row Class,” IsChanged, page 1415

Chapter 33, “Rowset Class,” page 1421

SetDimensionOrder

Syntax

```
SetDimensionOrder(&FieldNames)
```

Description

Use the SetDimensionOrder method to specify the order of dimensions for this cube collection.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&FieldNames	Specify an already instantiated array of string containing the field names that are the dimensions in the cube collection, in the order you want the dimensions.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetDimSort, page 92

SetDimFilter

Syntax

```
SetDimFilter(DimName, DimFilter)
```

Description

Use the SetDimFilter method to set a dimension filter for the specified dimension in the cube collection.

Parameters

Parameter	Description
DimName	Specify the name of the dimension for which you want to specify a dimension filter.
DimFilter	Specify the name of the dimension filter (user function) that you want to use the dimension.

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Dimensions," Understanding Dimensions

SetDimSort

Syntax

```
SetDimSort(DimName, IsAscending [, Key1, IsAscending2, Key2, IsAscending3, Key3])
```

Description

Use the SetDimSort method to specify the sorting order for the cube collection.

*Key1, Key2*and *Key3* are optional parameters. If they are not specified, the sorting works as a name sort, that is, it sorts by the name of members in the given dimension, either in ascending or descending order.

 Copyright © 1988-2007, Oracle. All rights reserved.

If keys are specified, the keys values refer to the names of the cubes to be used as they key. The method orders the members of the dimension based on the value of the cubes for each member. The boolean parameter *IsAscending* defines whether sorting should be in ascending or descending order. Up to three sort keys may be specified. *Key1* is the primary sort key. *Key2* is used to sub-sort any members that have the same key value under *Key1*, and so on.

The scope of this method is the same as the scope of the AnalyticModel object created in PeopleCode, that is, if the AnalyticModel object was declared with scope Global, this method would have a global scope as well, if it was declared as Local, it would have Local, and so on.

Specifying a null string for *DimName* or for keys one through three removes the current value as a sort value.

Parameters

Parameter	Description
DimName	Specify the dimension name that you would like sorted.
IsAscending	Specify if the sort order is ascending or descending. This parameter takes a boolean value: true if the sort is ascending, false if it's descending.
Key1	Specify the name of a key field by which the sort should be ordered. If you do not specify keys, the sort is ordered by the names of the members in the given dimensions. If you do specify keys, the sort is order by the keys.
IsAscending2	Specify if the sort order for *Key1*is ascending or descending. This parameter takes a boolean value: true if the sort is ascending, false if it's descending.
Key2	Specify the name of a key field to be used to sub-sort the primary key field *Key1*.
IsAscending3	Specify if the sort order for *Key2*is ascending or descending. This parameter takes a boolean value: true if the sort is ascending, false if it's descending.
Key3	Specify the name of a key field to be used to sub-sort the primary key field *Key2*.

Returns

None.

SetSlice

Syntax

```
SetSlice(&SliceRecord)
```

Description

Use the SetSlice method to specify a slice record for this cube collection.

Parameters

Parameter	Description
&SliceRecord	Specify an already instantiated record as the record to be used for the slice for the cube collection.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 2, “Analytic Calculation Engine Classes,” GetSlice, page 36

ShowHierarchy

Syntax

ShowHierarchy(*DimName*, *Show*, *ExpandToLevel*)

Description

Use the ShowHierarchy method to either hide or display the cube collection hierarchy.

If there is a tree attached to the dimension, and *Show* is false, only the detail level members are shown in the Analytic grid. If *Show* is true, all members including nodes are shown.

When the hierarchy is hidden, the GetData method returns only leaf values. When the hierarchy is displayed, GetData returns node values and leaf values.

The *ExpandToLevel* parameter expands the hierarchy to the level specified. Level indexes start from 0 , the root level.

For example, if you specify *ExpandToLevel* as 1, the grid displays the total (root member) and all the detail members in a dimension with no tree attached.

In the following example, if you specify *ExandToLevel* as 2, the hierarchy would be displayed only to the quarter level, and not to the month level.

```
ALL_TIME
     2003
          Q1
               Jan
               Feb
               Mar
          Q2
               Apr
               May
               Jun
          Q3
               . . .
               . . .
          Q4
     2004
          Q1
          Q2
          Q3
          Q4
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension for which you want to either display or hide the hierarchy.
Show	Specify whether to display or hide the hierarchy. This parameter takes a boolean value; true to display the hierarchy, false to hide it.
ExpandToLevel	Specify to what level to expand the hierarchy, as a number.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," CollapseNode, page 30 , Chapter 2, "Analytic Calculation Engine Classes," ExpandNode, page 32 and Chapter 2, "Analytic Calculation Engine Classes," GetData, page 33.

UnsetDimFilter

Syntax

UnsetDimFilter(*DimName*)

Description

Use the UnsetDimFilter method to remove the filter on the specified dimension for the cube collection.

Parameters

Parameter	Description
DimName	Specify the name of the dimension from which you want to remove the filter.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimFilter, page 38

UnsetDimSort

Syntax

UnsetDimSort(*DimName*)

Description

Use the UnsetDimSort method to remove the sorting on the specified dimension for a cube collection. After using this method, the members are returned in the global member order as specified in the analytic model definition.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension from which you want to remove sorting.

Returns

None.

See Also

Chapter 2, "Analytic Calculation Engine Classes," SetDimSort, page 38

CubeCollection Class Property

In this section we discuss the CubeCollection class property.

Messages

Description

This property returns a multi-dimensional array of any containing the messages that occurred during execution of the GetData, SetData, and CalculateCube methods.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

After you access this property, only new messages are returned, that is, messages are only returned once.

This property is read-only.

See Also

Chapter 2, "Analytic Calculation Engine Classes," GetData, page 33 and Chapter 2, "Analytic Calculation Engine Classes," SetData, page 37.

Chapter 2, "Analytic Calculation Engine Classes," CalculateCube, page 24

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 3

Analytic Calculation Engine Metadata Classes

This chapter provides an overview of the Analytic Engine Metadata classes, and discuses:

- Using the Analytic Calculation Engine Metadata classes
- Error handling
- Data types of the Analytic Calculation Engine Metadata objects
- Scope of Analytic Calculation Engine Metadata objects
- How to import the Analytic Calculation Engine Metadata classes
- How to create an Analytic Calculation Engine Metadata class object
- Analytic Calculation Engine Metadata classes reference

Understanding the PeopleSoft Analytic Calculation Engine Metadata Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the Analytic Grid classes to manipulate the display of analytic calculation data on a page.
- Use the Analytic Type classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 2, “Analytic Calculation Engine Classes,” page 11

Chapter 4, “Analytic Grid Classes,” page 129

Chapter 5, “Analytic Type Classes,” page 137

Using the Analytic Calculation Engine Metadata Classes

You can create analytic model definitions using PeopleSoft Application Designer. PeopleSoft provides the Analytic Calculation Engine Metadata classes for accessing these definitions at runtime.

You can also create an analytic model definition in PeopleCode, using the Create method, then saving it to the database using the Save method. After you save the definition, you can access it using PeopleSoft Application Designer.

All of the primary objects (dimensions, cube collections, cubes, and so on) are instantiated from an analytic model object, so you only need to create an instance of the analytic model. You can also build a rule as an object instead of as a string.

Inserting and Deleting Objects

If you insert or delete an object from a group of objects, any existing object references are no longer valid. For example, if you use the DeleteCube method, any existing references to cube objects are no longer valid. You need to create any object references after using any insert or delete method.

Building a Rule as an Object

The RuleExpressions sub-package contains several classes. Each of these classes represents the different parts of the analytic calculation engine rule grammar. These objects are created using the Create built-in function and the constructor for that class, then added to the RuleDefn object with the AddRuleExpression method.

The following example creates a cube that uses the Analytic Calculation Engine built-in function AT.

```
Function AT_Cube
   /** AT Cube Rule **/
   &Expected = "AT(GENERAL2, [GENERAL2:4], LOOKUPVALUES)";
   &CubeName = "AT";

   &FunCall = create PT_ANALYTICMODELDEFN:RuleExpressions:FunctionCall⇒
(&Constants.FuncCall_Type_Builtin, &Constants.FuncCall_Builtin_AT);

   &Constant = create PT_ANALYTICMODELDEFN:RuleExpressions:Constant⇒
(&Constants.Constant_Type_Literal, "GENERAL2");
   &FunCall.AddArgument(&Constant);

   &MemberRefArgument = create PT_ANALYTICMODELDEFN:RuleExpressions:Member⇒
Reference("GENERAL2", "4");
   &FunCall.AddArgument(&MemberRefArgument);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   &CubeArgument = create PT_ANALYTICMODELDEFN:RuleExpressions:Cube("LOOKUPVALUES");
   &FunCall.AddArgument(&CubeArgument);

   &RuleDefn = create PT_ANALYTICMODELDEFN:RuleDefn("");
   &CubeDefn = &Model.GetCube(&CubeName);
   &RuleDefn.AddRuleExpression(&FunCall);
   &RuleDefn.GenerateRule();
   &CubeDefn.SetRule(&RuleDefn);
   If ( Not (Exact(&RuleDefn.RuleString, &Expected))) Then
      &Model.Save();
      Error ("Rule Experession API Failed [Cube " | &CubeName | " Rule]");
   End-If;
End-Function;
```

Error Handling

All the Analytic Calculation Engine Metadata classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

See Also

Chapter 14, “Exception Class,” Try-Catch Sections, page 483

Data Types of the Analytic Calculation Engine Metadata Objects

Every PeopleSoft Analytic Calculation Engine metadata object is declared as its own data type, that is, AnalyticModelDefn objects are declared as type AnalyticModelDefn, CubeDefn objects are declared as type CubeDefn, and so on.

Scope of Analytic Calculation Engine Metadata Objects

The Analytic Calculation Engine Metadata objects can only be instantiated from PeopleCode.

These objects can be used anywhere you have PeopleCode, that is, in a PeopleSoft Application Engine program, an application class, record field PeopleCode, and so on.

Analytic Calculation Engine Metadata objects can be of scope Local, Component or Global.

Copyright © 1988-2007, Oracle. All rights reserved.

How to Import the Analytic Calculation Engine Metadata Classes

The Analytic Calculation Engine metadata classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the Analytic Calculation Engine metadata classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_ANALYTICMODELDEFN contains the following subclasses:

- AnalyticModelDefn
- CubeCollectionDefn
- CubeDefn
- DimensionDefn
- ExplicitDimSet
- OrganizerDefn
- RuleDefn
- UserFunctionDefn

Additionally, there is a sub-application package, RuleExpressions, that contains classes used to build a rule as object instead of as a simple string. This package contains the following classes:

- Assignment
- Comparison
- Constant
- Constants
- Cube
- ExpressionBlock
- FunctionCall
- MemberReference
- Operation
- RuleExpression
- Variable

The import statements that you should use should be as follows:

```
import PT_ANALYTICMODELDEFN:*;
import PT_ANALYTICMODELDEFN:RuleExpressions:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 6, “Application Classes,” page 153

How to Create an Analytic Calculation Engine Metadata Class Object

After you’ve imported the Analytic Calculation Engine metadata classes, you need to instantiate an instance of the AnalyticModelDefn class, using the constructor for that class and the Create function. After you create the object, you must populate it using either the Get or Create method.

The following example creates an AnalyticModelDefn object from the QE_ALLFUNCTION analytic model definition.

```
import PT_ANALYTICMODELDEFN:*;

Local AnalyticModelDefn &Model;

&Model = create AnalyticModelDefn("QE_ALLFUNCTION");
```

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” Analytic Calculation Engine Metadata Classes Constructor, page 47

Analytic Calculation Engine Metadata Classes Constructor

You must use the constructor for the AnalyticModelDefn class to instantiate an instance of that class. All other metadata objects are instantiated from this class.

AnalyticModelDefn

Syntax

AnalyticModelDefn(*ModelName*)

Description

Use the AnalyticModelDefn constructor to create an AnalyticModelDefn object.

Once the AnalyticModelDefn object is created, you can then execute either the Get or Create methods to ‘instantiate’ the model.

Note. The Delete and Rename methods can only be executed before a model is ‘instantiated’.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ModelName	Specify the name of an analytic model definition.

Returns

An AnalyticModelDefn object.

Example

```
&Model = create AnalyticModelDefn("QE_ALLFUNCTION");
```

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AnalyticModelDefn Class, page 48

AnalyticModelDefn Class

Use the AnalyticModelDefn class to view or manipulate an analytic model definition that's already been created in PeopleSoft Application Designer, or to create a new definition.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine

AnalyticModelDefn Class Methods

The following section discusses the AnalyticModelDefn class methods. The methods are discussed in alphabetical order.

AddCube

Syntax

AddCube(*CubeName*)

Description

Use the AddCube method to add a new cube to the analytic model definition. This method fails if the cube specified by *CubeName* already exists.

Parameters

Parameter	Description
CubeName	Specify the name of a new cube that you want to add to the analytic model.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A CubeDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 52 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 56 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 60 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 62 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 67.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78

AddCubeCollection

Syntax

AddCubeCollection(*CubeCollectionName*)

Description

Use the AddCubeCollection method to add a new cube collection to the analytic model definition. This method fails if *CubeCollectionName* already exists.

Parameters

Parameter	Description
CubeCollectionName	Specify the name of a new cube collection that you want to add to the analytic model definition.

Returns

A CubeCollectionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 52 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 61 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 68.

AddDimension

Syntax

AddDimension(*DimName*)

Description

Use the AddDimension method to add a new dimension to the analytic model. This method fails if the specified dimension already exists.

When you add a dimension to a cube, the dimension is also automatically added to any cube collections that already contain the cube.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension you want to add. This must be a new dimension.

Returns

A DimensionDef object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 57 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 62.

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

AddExplicitDimensionSet

Syntax

AddExplicitDimensionSet(*ExplicitDimSetName*)

Description

Use the AddExplicitDimensionSet method to add a new explicit dimension set to the analytic model. This method fails if the specified explicit dimension set already exists.

Parameters

Parameter	Description
ExplicitDimSetName	Specify the name of the explicit dimension you want to add. This must be a new explicit dimension.

Returns

An ExplicitDimensionSet object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 63 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 64 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 69.

AddOrganizer

Syntax

AddOrganizer(*OrganizerName*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AddOrganizer method to add a new organizer to the analytic model. This method fails if the organizer specified by *OrganizerName* already exists.

Parameters

Parameter	Description
OrganizerName	Specify the name of the organizer you want to add.

Returns

An OrganizerDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 64 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 65 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 98

AddUserFunction

Syntax

AddUserFunction(*UserFunctionName*)

Description

Use the AddUserFunction method to add a new user function to the analytic model. This method fails if the user function specified by *UserFunction* already exists.

Parameters

Parameter	Description
UserFunctionName	Specify the name of the new user function that you want to add.

Returns

A UserFunction object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 59 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 65 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 66 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

Copyright © 1988-2007, Oracle. All rights reserved.

CopyCube

Syntax

CopyCube(*CubeName, NewCubeName*)

Description

Use the CopyCube method to copy the specified cube to the new cube. This method fails if the cube specified by *CubeName* already exists.

Parameters

Parameter	Description
CubeName	Specify the name of the cube that you want to copy.
NewCubeName	Specify the name of the new cube that you want to create.

Returns

A CubeDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 48 , Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 48 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 60 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 62 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 67.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78

CopyCubeCollection

Syntax

CopyCubeCollection(*CubeCollectionName, NewCubeCollectionName*)

Description

Use the CopyCubeCollection method to copy the specified cube collection to a new collection. If the new cube collection specified by *NewCubeCollectionName* already exists, this method fails.

Parameters

Parameter	Description
CubeCollectionName	Specify the name of the cube collection that you want to copy.
NewCubeCollectionName	Specify the name you want for the new cube collection.

Returns

A CubeCollectionDefn object.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 61 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 61 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 68.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 86

CopyDimension

Syntax

CopyDimension(*DimName, NewDimName*)

Description

Use the CopyDimension method to copy the dimension specified by *DimName* to a new dimension. If the dimension specified by *NewDimName* already exists, this method fails.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to copy.
NewDimName	Specify the name of the new dimension that you want the data copied to.

Returns

A DimensionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 57 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 68.

CopyExplicitDimensionSet

Syntax

CopyExplicitDimensionSet(*ExplicitDimSetName, NewExplicitDimSetName*)

Description

Use the CopyExplicitDimensionSet method to copy the explicit dimension set specified by *ExplicitDimSetName* to a new explicit dimension set. If the explicit dimension set specified by *NewExplicitDimSetName* already exists, this method fails.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ExplicitDimSetName	Specify the name of the explicit dimension set that you want to copy.
NewExplicitDimSetName	Specify the name of the new explicit dimension set that you want the data copied to.

Returns

An ExplicitDimensionSet object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 69 , Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 63 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 64.

CopyTo

Syntax

CopyTo(*NewModelName*)

Description

Use the CopyTo method to copy the AnalyticModelDefn object to a new analytic model specified by *NewModelName*. If the model set specified by *NewModelName* already exists, this method fails.

Parameters

Parameter	Description
NewModelName	Specify the name of a new analytic model definition that you want to create.

Returns

An AnalyticModelDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Delete, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 60 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 67 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71.

CopyUserFunction

Syntax

CopyUserFunction(*UserFunctionName, NewUserFunctionName*)

Description

Use the CopyUserFunction method to copy the user function specified by *UserFunctionName* to a new user function. If the user function specified by *NewUserFunctionName* already exists, this method fails.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
UserFunctionName	Specify the name of the user function that you want to copy.
NewUserFunctionName	Specify the name of the new user function that you want to add.

Returns

A UserFunctionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 51 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 59 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 65 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 66 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

Create

Syntax

```
Create()
```

Description

Use the Create method to create, and instantiate, a new model. If you save the new model after you create it, you can then access it in PeopleSoft Application Designer as an analytic model definition.

If the model already exists, an exception is thrown.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyTo, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Delete, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 60 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 67.

Delete

Syntax

```
Delete()
```

Description

Use the Delete method to delete the analytic model executing the method.

Copyright © 1988-2007, Oracle. All rights reserved.

You can only use this method on a *closed* analytic model definition, that is, before you use a Get or Create method.

Warning! The delete occurs immediately, that is, the analytic model definition is removed from the database. Only use this method if you're certain you want to delete the definition.

If this model is used by any existing analytic type definition, this method fails.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyTo, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 60 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 67 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71.

DeleteCube

Syntax

DeleteCube(*CubeName, ForceDelete*)

Description

Use the DeleteCube method to delete the cube specified by *CubeName*.

Parameters

Parameter	Description
CubeName	Specify the name of the cube you want to delete from the analytic model.
ForceDelete	Specify if the cube should always be deleted. This parameter takes a boolean value. If you specify *ForceDelete* as false, and the cube is used by another part (such as a cube collection) this method fails. If you specify *ForceDelete* as true, and the cube is used by another part, the cube is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 48 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 60 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 62 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 67.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 86

 Copyright © 1988-2007, Oracle. All rights reserved.

DeleteCubeCollection

Syntax

DeleteCubeCollection(*CubeCollectionName, ForceDelete*)

Description

Use the DeleteCubeCollection method to delete the cube collection specified by *CubeCollectionName*.

Parameters

Parameter	Description
CubeCollectionName	Specify the name of the cube collection that you want to delete from the analytic model.
ForceDelete	Specify if the cube collection should always be deleted. This parameter takes a boolean value. If you specify *ForceDelete* as false, and the cube collection is used by another part (such as an Organizer), this method fails. If you specify *ForceDelete* as true, and the cube collection is used by another part, the cube collection is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 61 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 61 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 68.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 86

DeleteDimension

Syntax

DeleteDimension(*DimensionName, ForceDelete*)

Description

Use the DeleteDimension method to delete the dimension specified by *DimensionName*.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension you want to delete.
ForceDelete	Specify if the dimension should always be deleted. This parameter takes a boolean value. If you specify *ForceDelete* as false, and the dimension is used by another part (such as a cube), this method fails. If you specify *ForceDelete* as true, and the dimension is used by another part, the dimension is still deleted.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 62 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 63 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 68.

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

DeleteExplicitDimensionSet

Syntax

DeleteExplicitDimensionSet(*ExplicitDimensionSetName, ForceDelete*)

Description

Use the DeleteExplicitDimensionSet method to delete the explicit dimension set specified by *ExplicitDimensionSetName*.

Parameters

Parameter	Description
ExplicitDimensionSetName	Specify the name of the explicit dimension set you want to delete.
ForceDelete	The value of this parameter is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 63 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 64 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 69.

DeleteOrganizer

Syntax

DeleteOrganizer(*OrganizerName, ForceDelete*)

Description

Use the DeleteOrganizer method to delete the organizer specified by *OrganizerName*.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
OrganizerName	Specify the name of the organizer you want to delete.
ForceDelete	This value is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 64 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 65 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 98

DeleteUserFunction

Syntax

DeleteUserFunction(*UserFunctionName, ForceDelete*)

Description

Use the DeleteUserFunction method to delete the user function specified by *UserFunctionName*.

Parameters

Parameter	Description
UserFunctionName	Specify the name of the user function you want to delete.
ForceDelete	Specify if the user function should always be deleted. This parameter takes a boolean value. If you specify *ForceDelete* as false, and the user function is used by another part, this method fails. If you specify *ForceDelete* as true, and the user function is used by another part, the user function is still deleted.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 51 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 65 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 66 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

Copyright © 1988-2007, Oracle. All rights reserved.

Get

Syntax

```
Get()
```

Description

Use the Get method to instantiate an instance of the analytic model.

If the model doesn't exist, an exception is thrown.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyTo, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Delete, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 67 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71.

GetCube

Syntax

```
GetCube(CubeName)
```

Description

Use the GetCube method to return a reference to the cube specified by *CubeName*. This method fails if the cube specified by *CubeName* doesn't exist.

Parameters

Parameter	Description
CubeName	Specify the name of the cube that you want a reference to.

Returns

A CubeDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 48 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 52 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 56 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 62 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCube, page 67.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 86

Copyright © 1988-2007, Oracle. All rights reserved.

GetCubeCollection

Syntax

GetCubeCollection(*CubeCollectionName*)

Description

Use the GetCubeCollection method to return a reference to a cube collection. This method fails if the cube collection specified by *CubeCollectionName* doesn't exist.

Parameters

Parameter	Description
CubeCollection	Specify the name of the cube collection you want a reference to.

Returns

A CubeCollectionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 52 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 61 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 61 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameCubeCollection, page 68.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeCollectionDefn Class, page 86

GetCubeCollectionNames

Syntax

GetCubeCollectionNames()

Description

Use the GetCubeCollectionNames method to return an array of strings containing all the names of the cube collections associated with this analytic model.

Parameters

None.

Returns

An array of string.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” AddCubeCollection, page 49 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” CopyCubeCollection, page 52 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” DeleteCubeCollection, page 57 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” GetCubeCollection, page 61 and Chapter 3, “Analytic Calculation Engine Metadata Classes,” RenameCubeCollection, page 68.

Chapter 3, “Analytic Calculation Engine Metadata Classes,” CubeCollectionDefn Class, page 86

Chapter 8, “Array Class,” page 215

GetCubeNames

Syntax

```
GetCubeNames()
```

Description

Use the GetCubeNames method to return an array of strings containing all the names of the cubes associated with this analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” AddCube, page 48 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” CopyCube, page 52 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” DeleteCube, page 56 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” GetCube, page 60 and Chapter 3, “Analytic Calculation Engine Metadata Classes,” RenameCube, page 67.

Chapter 3, “Analytic Calculation Engine Metadata Classes,” CubeCollectionDefn Class, page 86

Chapter 8, “Array Class,” page 215

GetDimension

Syntax

```
GetDimension(DimName)
```

Description

Use the GetDimension method to return a reference to the dimension specified by *DimName*. This method fails if the dimension specified by *DimName* doesn’t exist.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want a reference to.

Returns

A DimensionDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 63 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 68.

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

GetDimensionNames

Syntax

```
GetDimensionNames()
```

Description

Use the GetDimensionNames method to return an array of string containing the names of all the dimensions associated with the analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 62 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameDimension, page 68.

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

Chapter 8, "Array Class," page 215

GetExplicitDimensionSet

Syntax

```
GetExplicitDimensionSet(ExplicitDimSetName)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetExplicitDimensionSet method to return a reference to the explicit dimension set specified by *ExplicitDimSetName*. This method fails if the explicit dimension set specified by *ExplicitDimSetName* doesn't exist.

Parameters

Parameter	Description
ExplicitDimSetName	Specify the name of the explicit dimension set that you want a reference to.

Returns

An ExplicitDimensionSet object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 58 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSetNames, page 64.

GetExplicitDimensionSetNames

Syntax

```
GetExplicitDimensionSetNames()
```

Description

Use the GetExplicitDimensionSetNames method to return an array of string containing the names of all the explicit dimension sets associated with the analytic model.

Parameters

None.

Returns

An array of strings.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," ExplicitDimensionSet Class, page 76

GetOrganizer

Syntax

```
GetOrganizer(OrganizerName)
```

Description

Use the GetOrganizer method to return a reference to the organizer specified by *OrganizerName*. This method fails if the organizer specified by *OrganizerName* doesn't exist.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
OrganizerName	Specify the name of the organizer you want a reference to.

Returns

An OrganizerDefn object.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 65 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 98

GetOrganizerNames

Syntax

```
GetOrganizerNames()
```

Description

Use the GetOrganizerNames method to return an array of string containing the names of all the organizers associated with the analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 64 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameOrganizer, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 98

Chapter 8, "Array Class," page 215

GetUserFunction

Syntax

```
GetUserFunction(UserFunctionName)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetUserFunction method to return a reference to the user function specified by *UserFunctionName*. This method fails if the user function specified by *UserFunctionName* doesn't exist.

Parameters

Parameter	Description
UserFunctionName	Specify the name of a user function that you want a reference to.

Returns

A UserFunctionDefn object

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 51 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 59 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 66 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

GetUserFunctionNames

Syntax

```
GetUserFunctionNames()
```

Description

Use the GetUserFunctionNames method to return an array of string containing the names of all the user functions associated with the analytic model.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 51 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 59 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 65 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameUserFunction, page 70.

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

Chapter 8, "Array Class," page 215

 Copyright © 1988-2007, Oracle. All rights reserved.

Rename

Syntax

Rename(*NewModelName*)

Description

Use the Rename method to rename the analytic model.

You can only use this method on a *closed* analytic model definition, that is, before you use a Get or Create method.

Parameters

Parameter	Description
NewModelName	Specify the new name of the model.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Get, page 60 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71.

RenameCube

Syntax

RenameCube(*CubeName, NewCubeName, ForceRename*)

Description

Use the RenameCube method to rename the cube specified by *CubeName* to *NewCubeName*.

Parameters

Parameter	Description
CubeName	Specify the name of the cube you want to rename.
NewCubeName	Specify the new name for the cube.
ForceRename	Specify if the cube should always be renamed. This parameter takes a boolean value. If you specify *ForceRename* as false, and the cube is used by another part (such as a cube collection) this method fails. If you specify *ForceRename* as true, and the cube is used by another part, the cube is still renamed.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCube, page 48 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCube, page 52 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCube, page 56 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCube, page 60 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 62.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78

RenameCubeCollection

Syntax

RenameCubeCollection(*CubeCollectionName, NewCubeCollectionName, ForceRename*

Description

Use the RenameCubeCollection method to rename the cube collection specified by *CubeCollectionName* to the new name *NewCubeCollectionName*

Parameters

Parameter	Description
CubeCollectionName	Specify the name of the cube collection that you want to rename.
NewCubeCollectionName	Specify the new name of the cube collection.
ForceRename	Specify if the cube collection should always be renamed. This parameter takes a boolean value. If you specify *ForceRename* as false, and the cube collection is used by another part (such as an organizer) this method fails. If you specify *ForceRename* as true, and the cube collection is used by another part, the cube collection is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddCubeCollection, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyCubeCollection, page 52 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteCubeCollection, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollection, page 61 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 61.

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeCollectionNames, page 61

RenameDimension

Syntax

RenameDimension(*DimName, NewDimName, ForceRename*)

Description

Use the RenameDimension method to rename the dimension specified by *DimName* to the new name *NewDimName*.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension you want to rename.
NewDimName	Specify the new name of the dimension.
ForceRename	Specify if the dimension should always be renamed. This parameter takes a boolean value. If you specify *ForceRename* as false, and the dimension is used by another part (such as a cube) this method fails. If you specify *ForceRename* as true, and the dimension is used by another part, the dimension is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyDimension, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteDimension, page 57 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimension, page 62 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 63.

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

RenameExplicitDimensionSet

Syntax

RenameExplicitDimensionSet(*ExplicitDimSetName, NewExplicitDimSetName, ForceRename*)

Description

Use the RenameExplicitDimensionSet method to rename the explicit dimension set specified by *ExplicitDimSetName* to the new name *NewExplicitDimSetName*.

Parameters

Parameter	Description
ExplicitDimSetName	Specify the name of the explicit dimension set you want to rename.
NewExplicitDimSetName	Specify the new name of the explicit dimension set.
ForceRename	The value for this parameter is ignored in the current release.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddExplicitDimensionSet, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyExplicitDimensionSet, page 53 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteExplicitDimensionSet, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetExplicitDimensionSet, page 63 and Chapter 3, "Analytic Calculation Engine Metadata Classes," RenameExplicitDimensionSet, page 69.

RenameOrganizer

Syntax

RenameOrganizer(*OrganizerName, NewOrganizerName, ForceRename*)

Description

Use the RenameOrganizer method to rename the organizer specified by *OrganizerName* to the new name *NewOrganizerName*.

Parameters

Parameter	Description
OrganizerName	Specify the name of the organizer you want to rename.
NewOrganizerName	Specify the new name for the organizer.
ForceRename	The value for this parameter is ignored in the current release.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddOrganizer, page 50 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteOrganizer, page 58 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizer, page 64 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetOrganizerNames, page 65.

Chapter 3, "Analytic Calculation Engine Metadata Classes," OrganizerDefn Class, page 98

RenameUserFunction

Syntax

RenameUserFunction(*UserFunctionName, NewUserFunctionName, ForceRename*)

Description

Use the RenameUserFunction method to rename the user function specified by *UserFunctionName* to the new name *NewUserFunctionName*.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
UserFunctionName	Specify the name of the user function you want to rename.
NewUserFunctionName	Specify the new name for the user function.
ForceRename	Specify if the user function should always be renamed. This parameter takes a boolean value. If you specify *ForceRename* as false, and the user function is used by another part this method fails. If you specify *ForceRename* as true, and the user function is used by another part, the user function is still renamed.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddUserFunction, page 51 , Chapter 3, "Analytic Calculation Engine Metadata Classes," CopyUserFunction, page 54 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DeleteUserFunction, page 59 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunction, page 65 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 66.

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

Save

Syntax

```
Save()
```

Description

Use the Save method to save any changes to the analytic model definition to the database. If the saved model is valid, the value of the IsValid property is set to true, if it is not valid, it is set to false.

The Messages property is populate with messages from the validation that occurs with the save.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 55 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Validate, page 72.
Chapter 3, "Analytic Calculation Engine Metadata Classes," IsValid, page 73 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Messages, page 74.

Copyright © 1988-2007, Oracle. All rights reserved.

Validate

Syntax

```
Validate()
```

Description

Use the Validate method to validate the analytic model definition. This method returns true if the model successfully validates. If this method returns false, the detailed error messages are available through the Messages property.

The IsValid property is not set with this method. The Save method sets the IsValid property.

Parameters

None.

Returns

A boolean value: true if the analytic model definition validates successfully, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Create, page 55 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71 , Chapter 3, "Analytic Calculation Engine Metadata Classes," Messages, page 74 and Chapter 3, "Analytic Calculation Engine Metadata Classes," IsValid, page 73.

AnalyticModelDefn Properties

The following section discusses the AnalyticModelDefn class properties. The properties are discussed in alphabetical order.

CircularFormulaWarn

Description

Use this property to specify whether warnings occur when there are circular dependencies as rules are added. This property takes a boolean value: true if warnings should occur, false otherwise. The default value for this property is false.

This property is read-write.

Description

Description

Use this property to specify the description of the analytic model. This property takes a string value.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

IsValid

Description

This property indicates if the analytic model is valid, that is, if it would successfully load.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Validate, page 72 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Analytic Model Definitions," Validating Analytic Models

LongDescription

Description

Use this property to specify the long description of the analytic model definition. This property takes a string value.

This property is read-write.

MaxDelta

Description

This property specifies the maximum number of value changes for the analytic model

This property takes a float value. The default value is 0.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Analytic Model Definitions"

MaxIterations

Description

Use this property to specify the maximum number of iterations for the analytic model.

This property take an integer value. The default value is 100.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Designing and Editing Analytic Models"

Copyright © 1988-2007, Oracle. All rights reserved.

Messages

Description

This property returns a multi-dimensional array of any that contains the messages that occurred during execution of the Validate or Save methods. This array is only populated after the Validate or Save method has completed successfully.

The first dimension contains the message set number. The second dimension contains the message number. The third dimension contains the number of parameters in that message. The subsequent dimensions contain the values for the parameters. The maximum number of possible dimensions is 8.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Validate, page 72 and Chapter 3, "Analytic Calculation Engine Metadata Classes," Save, page 71.

Name

Description

This property indicates the name of the analytic model.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Rename, page 67

ResolveCircularDeps

Description

Use this property to specify if the analytic model should resolve circular dependencies through iteration. This property takes a boolean value: true if the dependencies should be resolved, false otherwise. The default value is false.

This property is read-write.

DimensionDefn Class

Use the DimensionDefn class to access a dimension that's associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn methods:

- AddDimension
- CopyDimension
- GetDimension

 Copyright © 1988-2007, Oracle. All rights reserved.

DimensionDefn Class Properties

The following section discusses the DimensionDefn class properties. The properties are discussed in alphabetical order.

AggregateSequence

Description

This property indicates the sequence number of this dimension used during aggregation of the analytic model

This property is read-only..

AggregationUserFunction

Description

Use this property to specify the name of a user function that can be used to override the default aggregation for this dimension.

This property is read-write.

Comments

Description

Use this property to specify additional notes about the dimension.

This property is read-write.

Name

Description

This property indicates the name of the dimension.

This property is read-only.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” GetDimensionNames, page 63 and Chapter 3, “Analytic Calculation Engine Metadata Classes,” RenameDimension, page 68.

TotalMemberName

Description

Use this property to specify a field value that is used at runtime to host the total value for this dimension. If a total isn’t applicable, this property has no value.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

ExplicitDimensionSet Class

Use the ExplicitDimensionSet class to access an explicit dimension set associated with an analytic model. You instantiate an object of this class using one of the following AnalyticModelDefn methods:

- AddExplicitDimensionSet
- CopyExplicitDimensionSet
- GetExplicitDimensionSet

You can also use the GetExplicitDimensionSetNames to return an array containing the names of all the explicit dimension sets associated with the analytic model.

ExplicitDimensionSet Methods

The following section discusses the ExplicitDimensionSet class methods. The methods are discussed in alphabetical order.

AttachDimension

Syntax

AttachDimension(*DimName*)

Description

Use the AttachDimension method to attach a dimension to an explicit dimension set.

Parameters

Parameter	Description
DimName	Specify the name of the dimension you want to attach.

Returns

None.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” DetachDimension, page 76

DetachDimension

Syntax

DetachDimension(*DimName*)

Description

Use the DetachDimension method to detach a dimension from an explicit dimension set.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to detach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 76

GetDimensionNames

Syntax

```
GetDimensionNames()
```

Description

Use the GetDimensionNames method to return an array of string that contains the names of all the dimensions associated with the ExplicitDimensionSet.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 76 and Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 76.

ExplicitDimensionSet Properties

The following section discusses the ExplicitDimensionSet class properties. The properties are discussed in alphabetical order.

Name

Description

The Name property returns the name of the explicit dimension set as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

SequenceNumber

Description

This property returns the sequence number of the explicit dimension set.

This property is read-only.

CubeDefn Class

Use the CubeDefn class to access a cube associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn methods:

- AddCubeDefn
- CopyCubeDefn
- GetCubeDefn

CubeDefn Class Methods

The following section discusses the CubeDefn class methods. The methods are discussed in alphabetical order.

AttachDimension

Syntax

AttachDimension(*DimName*)

Description

Use the AttachDimension method to attach an existing dimension specified by *DimName* to the cube.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to add.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 79 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesDimension, page 84.

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

DetachDimension

Syntax

DetatchDimension(*DimName*)

Description

Use the DetachDimension method to detach the dimension specified by *DimName* from the cube.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to detach from the cube.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 78 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 82 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesDimension, page 84.

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddDimension, page 49

Chapter 3, "Analytic Calculation Engine Metadata Classes," DimensionDefn Class, page 74

GetCauses

Syntax

GetCauses(*CauseType*)

Description

Any cube that affects another cube is a *cause* of that cube. Use the GetCauses method to return a list of all the other cubes that are the causes for this cube.

The cube names are returned in an array of string.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
CauseType	Specify the type of causes you want returned.

Value	Description
AnalyticModel_DirectCauses	Only return direct causes.
AnalyticModel_AllCauses	Return all cubes that are causes for this cube, both direct and indirect.
AnalyticModel_AllInputs	Return all cubes that have input to this cube.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetEffects, page 81 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCircularDeps, page 80.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Data Cubes," Understanding Causes and Inputs

GetCircularDeps

Syntax

```
GetCircularDeps(DimName)
```

Description

Use the GetCircularDeps method to return all cubes that have circular dependencies based on the dimension specified by *DimName*.

The cube names are returned in an array of string.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to find circular dependencies for.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCauses, page 79 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetEffects, page 81.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine

 Copyright © 1988-2007, Oracle. All rights reserved.

GetDimensionAggregate

Syntax

GetDimensionAggregate(*DimName*)

Description

Use the GetDimensionAggregate method to return the name of the user function used for the aggregate for the dimension specified by *DimName*.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want the aggregate user function.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetDimensionAggregate, page 83

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetUserFunctionNames, page 66

Chapter 3, "Analytic Calculation Engine Metadata Classes," UserFunctionDefn Class, page 97

GetEffects

Syntax

GetEffects(*EffectType*)

Description

Any cube that is affected by another cube is an *effect* of that cube. Use the GetEffects method to return a list of all the cubes affected by this cube.

The list of cubes is returned as an array of string.

Parameters

Parameter	Description
EffectType	Specify the type of effects you want returned.

Value	Description
AnalyticModel_DirectEffects	Only return cubes that are a direct effect on this cube.
AnalyticModel_AllEffects	Return all cubes that are an effect on this cube.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCauses, page 79 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCircularDeps, page 80.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine

GetDimensionNames

Syntax

```
GetDimensionNames()
```

Description

Use the GetDimensionNames method to return an array of string containing the names of all the dimensions associated with the cube.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 78 and Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 79.

Chapter 8, "Array Class," page 215

GetRule

Syntax

```
GetRule()
```

Description

Use the GetRule method to get a reference to a RuleDefn object that represents the rule for this cube.

Parameters

None.

Returns

A RuleDefn object.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleDefn Class, page 102

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetRule, page 83

SetDimensionAggregate

Syntax

SetDimensionAggregate(*DimName, UserFunctionName*)

Description

Use the SetDimensionAggregate method to specify the user function to be used for the aggregate for the dimension *DimName*.

Parameters

Parameter	Description
DimName	Specify the name of the dimension for which you want to set the aggregate.
UserFunctionName	Specify the name of the user function to be used as the aggregate function.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionAggregate, page 81

SetRule

Syntax

SetRule(*&RuleDefn*)

Description

Use the SetRule method to specify a RuleDefn object to be used as the rule for this cube.

Parameters

Parameter	Description
&RuleDefn	Specify an already instantiated RuleDefn object that you want to associate with this cube.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetRule, page 82

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleDefn Class, page 102

UsesDimension

Syntax

UsesDimension(*DimName*)

Description

Use the UsesDimension method to determine if the dimension specified by *DimName* is used by this cube.

Parameters

Parameter	Description
DimName	Specify the name of the dimension that you want to check for.

Returns

A boolean value: true if the dimension is used, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachDimension, page 78 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachDimension, page 79 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetDimensionNames, page 82.

CubeDefn Class Properties

The following section discusses the CubeDefn class properties. The properties are discussed in alphabetical order.

CalcAggregates

Description

Use this property to specify if the aggregate is calculated for this cube. This property takes a boolean value: true if the aggregate is calculated, false otherwise.

This property is read-write.

Comments

Description

Use this property to specify additional notes about the cube.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

DimensionCount

Description

This property indicates the number of dimensions associated with this cube.

This property is read-only.

FormatType

Description

Use this property to specify the format of the cube. The values are:

Value	Description
AnalyticModel_Format_ Number	The format is of type number.
AnalyticModel_Format_Date	The format is of type date.
AnalyticModel_Format_ Member	The format is of type member.
AnalyticModel_Format_Text	The format is of type text.

This property is read-write.

IsVirtual

Description

Use this property to specify if the cube is a virtual cube, that is, if it doesn't store data. This property takes a boolean value: true if the cube is a virtual cube, false otherwise.

This property is read-write.

Name

Description

This property specifies the name of the cube.

This property is read-only.

ValueDimensionName

Description

Use this property to specify the dimension name for cubes that have a format of member.

This property is only valid when a cube has a format of member.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," FormatType, page 85

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine

CubeCollectionDefn Class

Use the CubeCollectionDefn class to access cube collections associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn class methods:

- AddCubeCollection
- CopyCubeCollection
- GetCubeCollection

CubeCollectionDefn Class Methods

The following section discusses the CubeCollectionDefn class methods. The methods are discussed in alphabetical order.

AttachCube

Syntax

`AttachCube``(`*`CubeName`*`)`

Description

Use the AttachCube method to attach the existing cube specified by *CubeName* to the cube collection.

You should only specify cube collections comprised of derived/work records as the main record for a presentation cube collection.

Parameters

Parameter	Description
CubeName	Specify the name of the cube you want to attach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachCube, page 87 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 88.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78

 Copyright © 1988-2007, Oracle. All rights reserved.

DetachCube

Syntax

DetachCube(*CubeName*)

Description

Use the DetachCube to detach the cube specified by *CubeName* from the cube collection.

Parameters

Parameter	Description
CubeName	Specify the name of the cube that you want to detach.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 88.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78

GetAggregateMapping

Syntax

GetAggregateMapping(*PartName, IsCube*)

Description

Use the GetAggregateMapping method to return the aggregate mapping field specified by the part *PartName*.

Parameters

Parameter	Description
PartName	Specify the part name for which you want the aggregate mapping field name.
IsCube	Specify whether the part is a cube or not. This parameter takes a boolean value, true if the part is a cube, false otherwise.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 89 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 90 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 91 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 93 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 94.

Copyright © 1988-2007, Oracle. All rights reserved.

GetCubeNames

Syntax

```
GetCubeNames()
```

Description

Use the GetCubeNames method to return an array of strings containing all the names of the cubes associated with this cube collection.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” AddCube, page 48 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” CopyCube, page 52 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” DeleteCube, page 56 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” GetCube, page 60 and Chapter 3, “Analytic Calculation Engine Metadata Classes,” RenameCube, page 67.

Chapter 3, “Analytic Calculation Engine Metadata Classes,” CubeDefn Class, page 78

Chapter 8, “Array Class,” page 215

GetDimensionNames

Syntax

```
GetDimensionNames()
```

Description

Use the GetDimensionNames method to return an array of string containing the names of all the dimensions associated with the cube collection.

The dimension names are returned in an array of string.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” AttachDimension, page 78 and Chapter 3, “Analytic Calculation Engine Metadata Classes,” DetachDimension, page 79.

Chapter 3, “Analytic Calculation Engine Metadata Classes,” DimensionDefn Class, page 74

 Copyright © 1988-2007, Oracle. All rights reserved.

GetDimSort

Syntax

GetDimSort(*DimensionName*)

Description

Use the GetDimSort method return the sorting keys used for the dimension.

The sorting keys are returned as an array of any.

The first string is either true or false. If the first string is true it indicates that the dimension is sorted by the names of the members in the dimension.

The second string then is either true or false, indicating whether the sort is ascending (or descending). If the first string is false it indicates that the dimension is sorted by keys, which are the specified by the other strings in the array. The keys are names of cubes. The strings are as follows:

- Ascend1
- CubeName1
- Ascend2
- CubeName2
- Ascend3
- CubeName3

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension for which you want the sorting keys.

Returns

An array of any.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetDimSort, page 92
Chapter 2, "Analytic Calculation Engine Classes," GetDimSort, page 35 and Chapter 2, "Analytic Calculation Engine Classes," SetDimSort, page 38.

GetFieldMapping

Syntax

GetFieldMapping(*PartName, IsCube*)

Description

Use the GetFieldMapping method to return the name of the mapped field name for the part specified by *PartName*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PartName	Specify the name of the part for which you want the mapped field name.
IsCube	Specify whether this part is a cube or not. This parameter takes a boolean value: true if the part is a cube, false otherwise.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 87 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 93 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 91.

GetFilter

Syntax

GetFilter(*DimensionName*)

Description

Use the GetFilter method to return the name of the user function used as a filter for the dimension specified by *DimensionName*.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension for which you want the filter name.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 91

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Rules, Formulas, and User Functions," Filter User Functions

GetPersistAggregate

Syntax

GetPersistAggregate(*DimensionName*)

Description

Use the GetPersistAggregate method to return the value for the persist aggregate for the dimension specified by *DimensionName.*

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension for which you want to find the persist aggregate value.

Returns

An integer, which is one of the following values:

Value	Description
AnalyticModel_AggrType_ Root	The persist aggregate type is *Root*, that is, only the root node's data is saved to the database.
AnalyticModel_AggrType_All	The persist aggregate type is *All*, that is, save all of the dimension's aggregate data to the database.
AnalyticModel_AggrType_ None	The persist aggregate type is *None*, that is, do not save any of the dimension's aggregate data to the database.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 87 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 89 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 91 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 93 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 94.

SetAggregateMapping

Syntax

SetAggregateMapping(*PartName, IsCube, AggregateFieldName*)

Description

Use the SetAggregateMapping method to specify an aggregate field for the part specified by *PartName.*

You can map a field to only one data cube or dimension within one cube collection.

Parameters

Parameter	Description
PartName	Specify the name of the part for which you want to add an aggregate field.
IsCube	Specify whether the part is a cube. This parameter takes a boolean value, true if the part is a cube, false otherwise.
AggregateFieldName	Specify the name of the field to be used to hold the aggregate value.

Returns

A string containing the name of the field that contains the aggregate value.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 94 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 89 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 90 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 93 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 94.

SetDimSort

Syntax

SetDimSort(*DimName*, *IsAscending* [, *CubeName1*, *IsAscending2*, *CubeName2*, *IsAscending3*, *CubeName3*])

Description

Use the SetDimSort method to specify the sorting order for the dimension.

If *CubeName1* is an empty string, the dimension is sorted by the names of the members of the dimension, and the other parameters are not used.

If *CubeName1* is not an empty string, the dimension is sorted by cubes, as specified by the other parameters.

The boolean parameter *IsAscending* defines whether sorting should be in ascending or descending order. Up to three sort keys may be specified. *CubeName1* is the primary sort key. *CubeName2* is used to sub-sort any members that have the same key value under *CubeName1*, and so on.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension that you would like to specify a sort order for.
IsAscending	Specify if the dimension should be sorted in ascending or descending order. This parameter takes a boolean value: true if the sort should be ascending, false if it should be descending.
CubeName1	Specify whether the dimension is to be sorted by members or by cubes. If the dimension is to be sorted by members, specify a null string for this parameter. If the dimension is to be sorted by cubes, specify a cube name.
IsAscending2	Specify if the sort for *CubeName1* is in ascending or descending order. This parameter takes a boolean value: true if the sort is ascending, false if its descending.
CubeName2	Specify the name of a cube to be used to sub-sort the primary sort specified by *CubeName1*.
IsAscending3	Specify if the sort for *CubeName2* is in ascending or descending order. This parameter takes a boolean value: true if the sort is ascending, false if its descending.
CubeName3	Specify the name of a cube to be used to sub-sort the primary sort specified by *CubeName2*.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 94

Chapter 2, "Analytic Calculation Engine Classes," GetDimSort, page 35

SetFieldMapping

Syntax

```
SetFieldMapping(PartName, FieldName, IsCube)
```

Description

Use the SetFieldMapping method to specify the name of the main field used for mapping the part specified by *PartName*.

You can map a field to only one data cube or dimension within one cube collection.

Parameters

Parameter	Description
PartName	Specify the name of the part for which you want to assign a mapping field.
FieldName	Specify the name of the field to be used for the field mapping.
IsCube	Specify whether or not the part is a cube. This parameter takes a boolean value: true if the part is a cube, false otherwise.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 87 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 89 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 90 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 91 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetPersistAggregate, page 94.

SetFilter

Syntax

```
SetFilter(DimensionName, UserFunctionName)
```

Description

Use the SetFilter method to specify the user function to be used as a filter for the dimension specified by *DimensionName*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension for which you want to set a filter.
UserFunctionName	Specify the name of a user function to be used as a filter for this dimension.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFilter, page 90

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Creating Rules, Formulas, and User Functions," Filter User Functions

SetPersistAggregate

Syntax

```
SetPersistAggregate(DimensionName, AggregateType)
```

Description

Use the SetPersistAggregate to specify the value for the persist aggregate for the dimension specified by *DimensionName*.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension for which you want to set the persist aggregate.
AggregateType	Specify the aggregate type. Values are:

Value	Description
AnalyticModel_AggrType_ Root	The persist aggregate type is *Root*, that is, only the root node's data is saved to the database.
AnalyticModel_AggrType_All	The persist aggregate type is *All*, that is, save all of the dimension's aggregate data to the database.
AnalyticModel_AggrType_ None	The persist aggregate type is *None*, that is, do not save any of the dimension's aggregate data to the database.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GetAggregateMapping, page 87 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetFieldMapping, page 89 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPersistAggregate, page 90 , Chapter 3, "Analytic Calculation Engine Metadata Classes," SetAggregateMapping, page 91 and Chapter 3, "Analytic Calculation Engine Metadata Classes," SetFieldMapping, page 93.

UsesCube

Syntax

UsesCube(*CubeName*)

Description

Use the UsesCube method to determine if the cube collection uses the cube specified by *CubeName*.

Parameters

Parameter	Description
CubeName	Specify the name of the cube that you want to verify.

Returns

A boolean value, true if the specified cube is part of the cube collection, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachCube, page 86 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachCube, page 87 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetCubeNames, page 88.

Chapter 3, "Analytic Calculation Engine Metadata Classes," CubeDefn Class, page 78

UsesDimension

Syntax

UsesDimension(*DimensionName*)

Description

Use the UsesDimension method to determine if the cube collection uses the dimension specified by *DimensionName*.

Parameters

Parameter	Description
DimensionName	Specify the name of the dimension that you want to verify.

Returns

A boolean value: true if the dimension is part of the cube collection, false otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” CubeDefn Class, page 78

Chapter 3, “Analytic Calculation Engine Metadata Classes,” DimensionDefn Class, page 74

CubeCollectionDefn Class Properties

The following section discusses the CubeCollectionDefn class properties. The properties are discussed in alphabetical order.

AggregateRecName

Description

Use this property to specify the name of the aggregate record associated with this cube collection. This property takes a string value.

This property is read-write.

Comments

Description

Use this property to specify additional notes about the cube collection.

This property is read-write.

CubeCount

Description

This property returns the number of cubes in the cube collection.

This property is read-only.

DimensionCount

Description

This property returns the number of dimensions associated with this cube collection.

This property is read-only.

Name

Description

This property returns the name of the cube collection as a string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

RecordName

Description

Use this property to specify the name of the main record associated with the cube collection. This property takes a string value.

This property is read-write.

UserFunctionDefn Class

Use the UserFunctionDefn class to access the user functions that are associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn class methods:

- AddUserFunction
- CopyUserFunction
- GetUserFunction

UserFunctionDefn Class Methods

In this section we discuss the UserFunctionDefn class methods. They are described in alphabetical order.

GetRule

Syntax

```
GetRule()
```

Description

Use the GetRule method to return a RuleDefn object that represents the rule associated with this user function.

Parameters

None.

Returns

A RuleDefn object.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” RuleDefn Class, page 102

SetRule

Syntax

```
SetRule(&Rule)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetRule method to specify a rule defined by a RuleDefn object to be the rule for this user function.

Parameters

Parameter	Description
&Rule	Specify an already instantiated RuleDefn object that you want to use as the rule for this user function.

Returns

None.

UserFunctionDefn Class Properties

The following section discusses the UserFunctionDefn class properties. The properties are discussed in alphabetical order.

Comments

Description

Use this property to specify additional notes about the user function.

This property is read-write.

Name

Description

This property specifies the name of the user function.

This property is read-only.

OrganizerDefn Class

Use the OrganizerDefn class to access the organizers that are associated with an analytic model. You instantiate an object of this class using the following AnalyticModelDefn class methods:

- AddOrganizer
- CopyOrganizer
- GetOrganizer

Copyright © 1988-2007, Oracle. All rights reserved.

OrganizerDefn Class Methods

The following section discusses the OrganizerDefn class methods. The methods are discussed in alphabetical order.

AttachPart

Syntax

AttachPart(*PartName, PartType*)

Description

Use the AttachPart method to attach the part specified by *PartName* to the organizer. The part must exist in the analytic model.

Parameters

Parameter	Description
PartName	Specify the name of the part that you want to attach.
PartType	Specify the type of part that you want to attach. Values are:

Value	Description
AnalyticModel_Dimension	Attach a dimension.
AnalyticModel_Cube	Attach a cube.
AnalyticModel_ CubeCollection	Attach a cube collection.
AnalyticModel_UserFunction	Attach a user function.

Returns

None.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” DetachPart, page 99 , Chapter 3, “Analytic Calculation Engine Metadata Classes,” GetPartNames, page 100 and Chapter 3, “Analytic Calculation Engine Metadata Classes,” UsesPart, page 100.

DetachPart

Syntax

DetachPart(*PartName, PartType*)

Description

Use the DetachPart method to detach the part specified by *PartName* from the organizer.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PartName	Specify the name of the part that you want to detach.
PartType	Specify the type of the part that you want to detach. Valid values are:

Value	Description
AnalyticModel_Dimension	Detach a dimension.
AnalyticModel_Cube	Detach a cube.
AnalyticModel_ CubeCollection	Detach a cube collection.
AnalyticModel_UserFunction	Detach a user function.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachPart, page 99 , Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPartNames, page 100 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesPart, page 100.

GetPartNames

Syntax

```
GetPartNames()
```

Description

Use the GetPartNames method to return a list of the names of the parts used by the organizer. The names are returned as an array of string.

Parameters

None.

Returns

An array of string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachPart, page 99 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachPart, page 99 and Chapter 3, "Analytic Calculation Engine Metadata Classes," UsesPart, page 100.

UsesPart

Syntax

```
UsesPart(PartName, PartType)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the UsesPart method to determine if the organizer uses the part specified by *PartName*.

Parameters

Parameter	Description
PartName	Specify the name of the part that you want to verify.
PartType	Specify the type of the part that you want to verify. Valid values are:

Value	Description
AnalyticModel_Dimension	A dimension.
AnalyticModel_Cube	A cube.
AnalyticModel_ CubeCollection	A cube collection.
AnalyticModel_UserFunction	A user function.

Returns

A boolean value: true if the part is associated with the organizer, false otherwise.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AttachPart, page 99 , Chapter 3, "Analytic Calculation Engine Metadata Classes," DetachPart, page 99 and Chapter 3, "Analytic Calculation Engine Metadata Classes," GetPartNames, page 100.

OrganizerDefn Class Properties

The following section discusses the OrganizerDefn class properties. The properties are discussed in alphabetical order.

Comments

Description

Use this property to specify additional notes about the organizer.

This property is read-write.

Name

Description

This property specifies the name of the organizer.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

RuleDefn Class

Use the RuleDefn class to access a rule that is associated with an analytic model. You instantiate an object of this class using the GetRule method for both the CubeDefn and UserFunctionDefn classes.

RuleDefn Class Methods

In this section, we discuss the RuleDefn class methods. The methods are described in alphabetical order.

AddRuleExpression

Syntax

AddRuleExpression(*&Expr*)

Description

Use the AddRuleExpression method to add a rule expression to the rule. Use the classes in the RuleExpression subpackage to create the expression specified by *&Expr*.

After you have finished adding rule expressions to the rule, you need to use the GenerateRule method to generate the rule.

Parameters

Parameter	Description
&Expr	Specify the rule expression that you want added to the rule definition.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," GenerateRule, page 102

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 103

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to create the rule string for the rule that represents the current rule expressions that have been added using AddRuleExpression.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddRuleExpression, page 102

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 103

RuleDefn Class Property

The following describes the RuleDefn class property.

RuleString

Description

This property specifies a string representation of the rule.

After the rule is returned using the GetRule method, this string represents the existing rule cube or user function.

After AddRuleExpression and GenerateRule are called, this is the string representation of the rule.

This property is read-only.

RuleExpressions Classes

The RuleExpressions classes represents the different parts of the analytic calculation engine rule grammar. These objects are created using the Create built-in function and the constructor for that class, then added to the RuleDefn object with the AddRuleExpression method.

The following are the RuleExpressions classes:

- Assignment
- Comparison
- Constant
- Constants
- Cube
- ExpressionBlock
- FunctionCall
- MemberReference

Copyright © 1988-2007, Oracle. All rights reserved.

- Operation
- RuleExpression
- Variable

Note. The RuleExpression object is the object that the other objects are derived from. You should not create or use this object directly.

Using the Constants Class

All of the constants used with the RuleExpression classes, such as FunctionCall, comparison, operation, and so on, are actually properties of the constants class. You must always instantiate an object of the constants class to use any constants in your program.

For example, the comparison class uses a constant to test whether two operands are equal. Use the following code to create a comparison testing this:

```
&Comparison = create Comparison(&Constants.Comparison_Type_Equal);
```

All of the properties for the constants class are listed with the classes that use them.

Assignment Class

An assignment object represents an assignment statement in an analytic calculation rule.

Use the following to create an assignment object:

```
&Assignment = create Assignment();
```

Assignment Class Method

The following is the method for the assignment class.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A string.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddRuleExpression, page 102

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 103

Assignment Class Properties

In this section we discuss the assignment class properties. The properties are described in alphabetical order.

Expression

Description

Use the Expression property to specify the right-hand side of the assignment statement.

This property is read-write.

Variable

Description

Use the Variable property to specify the left-hand side of the assignment statement.

This property is read-write.

Comparison Class

A comparison object represents a comparison statement in an analytic calculation rule.

Use the following to create a comparison class object:

```
&Comparison = create Comparison(&Constants.Comparison_Type);
```

Where *Comparison_Type* is one of the following:

Value	Description
Comparison_Type_Equal	Compare if the value of Operand1 equals Operand2.
Comparison_Type_Greater	Compare if the value of Operand1 is greater than the value of Operand2.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
Comparison_Type_Less	Compare if the value of Operand1 is less than the value of Operand2.
Comparison_Type_GreaterEq	Compare if the value of Operand1 is greater or equal to the value of Operand2.
Comparison_Type_LessEq	Compare if the value of Operand1 is less than or equal to the value of Operand2.
Comparison_Type_NotEq	Compare if the value of Operand1 is not equal to the value of Operand2.

Specify Operand1 and Operand2 with the Operand1 and Operand2 comparison class properties.

The following code example creates a rule that compares if the first operand is greater than or equal to the second operand, then adds the rule using the AddArgument method.

```
&Comparison = create Comparison(&Constants.Comparison_Type_GreaterEq);
&Constant = create Constant(&Constants.Constant_Type_Literal, "1000");
&Comparison.Operand1 = &Constant;
&Constant = create Constant(&Constants.Constant_Type_Literal, "100");
&Comparison.Operand2 = &Constant;
&FunCall.AddArgument(&Comparison);
```

Comparison Class Method

The following is the comparison class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddRuleExpression, page 102

Chapter 3, "Analytic Calculation Engine Metadata Classes," RuleExpressions Classes, page 103

Comparison Class Properties

In this section we discuss the comparison class properties. The properties are described in alphabetical order.

Operand1

Description

Use this property to specify the RuleExpression object that is the first operand to be used in the comparison.

This property is read-write.

Operand2

Description

Use this property to specify the RuleExpression object that is the second operand to be used in the comparison.

This property is read-write.

Type

Description

This property returns the type of the comparison that was used to instantiate the comparison object.

The values are:

Value	Description
Comparison_Type_Equal	Compare if the value of Operand1 equals Operand2.
Comparison_Type_Greater	Compare if the value of Operand1 is greater than the value of Operand2.
Comparison_Type_Less	Compare if the value of Operand1 is less than the value of Operand2.
Comparison_Type_GreaterEq	Compare if the value of Operand1 is greater than or equal to the value of Operand2.
Comparison_Type_LessEq	Compare if the value of Operand1 is less than or equal to the value of Operand2.
Comparison_Type_NotEq	Compare if the value of Operand1 is not equal to the value of Operand1.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Constant Class

A constant object represents a constant statement in an analytic calculation rule.

Note. This is not the same as the constants class.

A constant object exposes all the constants that are passed to the constructors of the various RuleExpression objects.

Use the following to create a constant class object:

```
&Constant = create Constant(&Constants.Constant_Type, [&Constants.]Constant_Value);
```

where *Constant_Type* is one of the following:

Value	Description
Constant_Type_Builtin	The *Constant_Value*is an analytic calculation engine built-in function. In this instance, the constant value must be prefaced with an already instantiated constants object. For possible values, see below.
Constant_Type_Number	The *Constant_Value* is a number.
Constant_Type_Literal	The *Constant_Value* is a string.

When the *Constant_Type* is Constant_Type_Builtin, the *Constant_Value* is one of the following:

- Constant_Builtin_ALL
- Constant_Builtin_BASE_E
- Constant_Builtin_DEFAULT
- Constant_Builtin_DETAILS
- Constant_Builtin_DIRECT
- Constant_Builtin_FALSE
- Constant_Builtin_FORWARD
- Constant_Builtin_PI
- Constant_Builtin_REVERSE
- Constant_Builtin_TRUE

The following code creates a PeopleSoft Analytic Calculation Engine built-in function to be used as a constant:

```
&Constant = create Constant(&Constants.Constant_Type_Builtin,⇒
&Constants.Constant_Builtin_REVERSE);
```

The following code creates a literal (string) constant:

```
&Constant = create Constant(&Constants.Constant_Type_Literal, "GENERAL");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

The following code creates a number constant:

```
&Constant = create Constant(&Constants.Constant_Type_Number, "-1");
```

Constant Class Method

The following is the constant class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

Constant Class Properties

In this section we discuss the constant class properties. The properties are described in alphabetical order.

Type

Description

This property returns the value of the *Constant Type* that was used to instantiate the constant class object.

This property is read-only.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," Constant Class, page 108

Value

Description

This property returns the value of the *Constant Value* that was used to instantiate the constant class object.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” Constant Class, page 108

Constants Class

All of the constants used with the RuleExpression classes, such as FunctionCall, comparison, operation, and so on, are actually properties of the constants class. You must always instantiate an object of the constants class to use any constants in your program.

For example, the comparison class uses a constant to test whether two operands are equal. Use the following code to create a comparison testing this:

```
&Comparison = create Comparison(&Constants.Comparison_Type_Equal);
```

All of the properties for the constants class are listed with the classes that use them.

Cube Class

A cube object represents a cube statement in an analytic calculate engine rule.

Use the following to create a cube object.

```
&Assignment = create Cube("CubeName");
```

Cube Class Methods

In the following section, we discuss the cube class methods. The methods are described in alphabetical order.

AddIndex

Syntax

AddIndex(*&MemberReference*)

Description

Use the AddIndex method to add a MemberReference object to the cube.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&MemberReference	Specify an already instantiated MemberReference object to be added to the cube.

Returns

None.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," MemberReference Class, page 118

GenerateRule

Syntax

```
GenerateRule()
```

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

GetIndexes

Syntax

```
GetIndexes()
```

Description

Use the GetIndexes method to return an array of MemberReference's (indexes) for this cube. These MemberReferences are the MemberReferences that were added with the AddIndex method.

Parameters

None.

Returns

An array of MemberReference objects.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," MemberReference Class, page 118

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddIndex, page 110

Cube Class Property

The following is the cube class property.

Name

Description

This property indicates the name of the cube that was used to instantiate the cube object.

This property is read-only.

ExpressionBlock Class

For some function, you need multi-statement nested expressions. You need to use the ExpressionBlock class to group these expressions. For example, all the statements inside of a FOR statement should be included in an expression block.

```
FOR(&Index, 1, PERIOD,
   SET(&Value, &Value + 1);
   SET(&countPl4, &countPl4 + 4);
);
```

Use the following code to create an ExpressionBlock object:

```
&ExpressionBlock = create ExpressionBlock();
```

ExpressionBlock Methods

In the following section we discuss the ExpressionBlock class methods. The methods are described in alphabetical order.

AddRuleExpression

Syntax

AddRuleExpression(*&Expr*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AddRuleExpression method to add an expression to the expression block and the rule. Use the classes in the RuleExpression subpackage to create the expression specified by *&Expr*.

Note. As the rule string is generated, the system adds a ";" after each RuleExpression, except the last.

Parameters

Parameter	Description
&Expr	Specify the rule expression that you want added to the expression block.

Returns

None.

GetRuleExpressions

Syntax

```
GetRuleExpressions()
```

Description

Use the GetRuleExpressions to return an array of rule expression objects that have been added using the AddRuleExpression method.

Parameters

None.

Returns

An array of RuleExpression objects.

FunctionCall Class

A FunctionCall object represents a function call statement in an analytic calculation engine rule.

Use the following code to create a FunctionCall object:

```
&Assignment = create FunctionCall(&Constants.Function_Call_Type, [&Constants.]⇒
Function_Call_Name);
```

Where *Function_Call_Type* can be one of the following:

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
FunctionCall_Type_Builtin	Specifies that the function call is a PeopleSoft Analytic Calculation Engine built-in functions. In this instance, the function name must be prefaced with an already instantiated constants object. For the built-in function values used with *Function_Call_Name*, see below.
FunctionCall_Type_UserFunc	Specifies that the function call is a user function. The value specified for *Function_Call_Name* must be an already created user function defined in this analytic model. Specify this value as a string.

When the *Function_Call_Type* is specified as Function_Call_Type_Builtin, the *Function_Call_Name* must be one of the following:

- FunctionCall_Builtin_ABS
- FunctionCall_Builtin_ACOS
- FunctionCall_Builtin_ARGUMENTS
- FunctionCall_Builtin_ASC
- FunctionCall_Builtin_ASIN
- FunctionCall_Builtin_AT
- FunctionCall_Builtin_ATAN
- FunctionCall_Builtin_BREAK
- FunctionCall_Builtin_CASE
- FunctionCall_Builtin_CHANGE
- FunctionCall_Builtin_CHILDCOUNT
- FunctionCall_Builtin_CHR
- FunctionCall_Builtin_CONSOL
- FunctionCall_Builtin_COS
- FunctionCall_Builtin_CUBEID
- FunctionCall_Builtin_CUBSUM
- FunctionCall_Builtin_CUMAVG
- FunctionCall_Builtin_DAVG
- FunctionCall_Builtin_DAY
- FunctionCall_Builtin_DCOUNT
- FunctionCall_Builtin_DDB
- FunctionCall_Builtin_DEC
- FunctionCall_Builtin_DLOOKUP
- FunctionCall_Builtin_DMAX
- FunctionCall_Builtin_DMIN
- FunctionCall_Builtin_DSUM

 Copyright © 1988-2007, Oracle. All rights reserved.

- FunctionCall_Builtin_E
- FunctionCall_Builtin_FIRST
- FunctionCall_Builtin_FOR
- FunctionCall_Builtin_FORCHILDREN
- FunctionCall_Builtin_FORMEMBERS
- FunctionCall_Builtin_FV
- FunctionCall_Builtin_GROUPAVG
- FunctionCall_Builtin_GROUPBY
- FunctionCall_Builtin_GROUPMAX
- FunctionCall_Builtin_GROUPMIN
- FunctionCall_Builtin_GROUPSUM
- FunctionCall_Builtin_GROW
- FunctionCall_Builtin_IF
- FunctionCall_Builtin_IFNPV
- FunctionCall_Builtin_INC
- FunctionCall_Builtin_INDICATE
- FunctionCall_Builtin_INPUT
- FunctionCall_Builtin_INSUBTREE
- FunctionCall_Builtin_INTERCEPT
- FunctionCall_Builtin_IRR
- FunctionCall_Builtin_ISINPUT
- FunctionCall_Builtin_LEFT
- FunctionCall_Builtin_LEN
- FunctionCall_Builtin_LN
- FunctionCall_Builtin_LOWER
- FunctionCall_Builtin_MATCH
- FunctionCall_Builtin_MAX
- FunctionCall_Builtin_MBR2TEXT
- FunctionCall_Builtin_MEDIAN
- FunctionCall_Builtin_MEMBER
- FunctionCall_Builtin_MID
- FunctionCall_Builtin_MIN
- FunctionCall_Builtin_MOD
- FunctionCall_Builtin_MONTH
- FunctionCall_Builtin_NEXT
- FunctionCall_Builtin_NPER

Copyright © 1988-2007, Oracle. All rights reserved.

- FunctionCall_Builtin_NUM2TEXT
- FunctionCall_Builtin_NUMMEMBERS
- FunctionCall_Builtin_NPV
- FunctionCall_Builtin_PARAMETER
- FunctionCall_Builtin_PARENT
- FunctionCall_Builtin_PCT
- FunctionCall_Builtin_PERCENTILE
- FunctionCall_Builtin_PI
- FunctionCall_Builtin_PMT
- FunctionCall_Builtin_PREV
- FunctionCall_Builtin_PREVSELF
- FunctionCall_Builtin_PV
- FunctionCall_Builtin_QUARTILE
- FunctionCall_Builtin_RAND
- FunctionCall_Builtin_RATE
- FunctionCall_Builtin_REPLACE
- FunctionCall_Builtin_RETURN
- FunctionCall_Builtin_RIGHT
- FunctionCall_Builtin_ROUND
- FunctionCall_Builtin_SELF
- FunctionCall_Builtin_SET
- FunctionCall_Builtin_SIN
- FunctionCall_Builtin_SLN
- FunctionCall_Builtin_SLOPE
- FunctionCall_Builtin_SQRT
- FunctionCall_Builtin_STDEV
- FunctionCall_Builtin_SYD
- FunctionCall_Builtin_TAN
- FunctionCall_Builtin_TEXT2MBR
- FunctionCall_Builtin_TEXT2NUM
- FunctionCall_Builtin_THIS
- FunctionCall_Builtin_THISCUBE
- FunctionCall_Builtin_TRUNC
- FunctionCall_Builtin_UPPER
- FunctionCall_Builtin_VAR
- FunctionCall_Builtin_WHILE

Copyright © 1988-2007, Oracle. All rights reserved.

- FunctionCall_Builtin_YEAR

The following code creates a function call using a PeopleSoft Analytic Calculation Engine built-in function.

```
 &FunCall = create FunctionCall(&Constants.FuncCall_Type_Builtin,⇒
&Constants.FuncCall_Builtin_ABS);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Using Built-in Functions in Analytic Models"

FunctionCall Class Methods

In this section we discuss the FunctionCall class methods. The methods are described in alphabetical order.

AddArgument

Syntax

AddArgument(*&RuleExpression*)

Description

Use the AddArgument method to add an argument to this function call.

Parameters

Parameter	Description
&RuleExpression	Specify an already instantiated rule expression object, such as a cube object.

Returns

None.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A string.

GetArguments

Syntax

```
GetArguments()
```

Description

Use the GetArguments method to return an array of RuleExpression objects associated with this FunctionCall. These are the RuleExpression objects that were added using the AddArgument method.

Parameters

None.

Returns

An array of RuleExpression objects.

See Also

Chapter 3, "Analytic Calculation Engine Metadata Classes," AddArgument, page 117

FunctionCall Class Properties

In this section we discuss the FunctionCall class properties. The properties are described in alphabetical order.

Name

Description

This property specifies the name of the function call used to instantiate the FunctionCall object.

This property is read-only.

Type

Description

This property specifies the type of the function call used to instantiate the FunctionCall object.

This property is read-only.

MemberReference Class

A MemberReference object represents a member in an analytic calculation engine rule.

 Copyright © 1988-2007, Oracle. All rights reserved.

Use the following code to create a MemberReference object:

```
&Member = create MemberReference(DimensionName, MemberName);
```

MemberReference Class Method

This section discusses the MemberReference class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

MemberReference Class Properties

In this section we discuss MemberReference class properties. The properties are described in alphabetical order.

Dimension

Description

This property specifies the name of the dimension used to create the MemberReference object.

This property is read-only.

Member

Description

This property specifies the name of the member used to create the MemberReference object.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Operation Class

An operation object represents an operation in an analytic calculation engine rule. The operation is generally some type of mathematical function performed between two operands. Specify the operands for the operation using the Operand1 and Operand2 operation class properties.

Use the following code to create an operation object:

```
&Comparison = create Operation(&Constants.Operation_Type);
```

Where *Operation_Type* has a value of one of the following:

Value	Description
Operation_Type_Plus	The operation is addition.
Operation_Type_Minus	The operation is subtraction.
Operation_Type_Mult	The operation is multiplication.
Operation_Type_Div	The operation is division.
Operation_Type_And	Use a logical AND with the operation.
Operation_Type_Or	Use a logical OR with the operation.
Operation_Type_Not	Use a logical NOT with the operation.
Operation_Type_Neg	Use a mathmatical negate with the operation.
Operation_Type_Exp	Use a mathmatical exponent with the operation.

The following code creates an operations that is a multiplication between two operands.

```
&operation = create Operation(&Constants.Operation_Type_Mult);
 &Constant = create Constant(&Constants.Constant_Type_Literal, "PI");
 &operation.Operand1 = &Constant;
 &Constant = create Constant(&Constants.Constant_Type_Literal, "SLOPE");
 &operation.Operand2 = &Constant;
 &FunCall.AddArgument(&operation);
```

Operation Class Method

The following is the operation class method.

GenerateRule

Syntax

GenerateRule()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

Operation Class Properties

In this section we discuss the operation class properties. The properties are described in alphabetical order.

Operand1

Description

Use this property to specify the RuleExpression object that is the first operand to be used in the operation.

This property is read-write.

Operand2

Description

Use this property to specify the RuleExpression object that is the second operand to be used in the operation.

This property is read-write.

Type

Description

This property returns the type of the operation used to instantiate the operation object.

The values for this property are:

Value	Description
Operation_Type_Plus	The operation is addition.
Operation_Type_Minus	The operation is subtraction.
Operation_Type_Mult	The operation is multiplication.
Operation_Type_Div	The operation is division.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
Operation_Type_And	Use a logical AND for the operation.
Operation_Type_OR	Use a logical OR for the operation.
Operation_Type_Not	Use a logical NOT for the operation.
Operation_Type_Neg	Use a mathmatical negate for the operation.
Operation_Type_Exp	Use a mathmatical exponent for the operation.

Variable Class

A variable object represents a variable in an analytic calculation engine rule.

Use the following code to create a variable object:

```
&Variable = create Variable(&Constants.Variable_Type, "Variable_Name");
```

Where *Variable_Type* is one of the following:

Value	Description
Variable_Type_Auto	Use a local variable.
Variable_Type_Expression	The variable is in an expression. Specify an already instantiated rule expression object for the *Variable_Name.*
Variable_Type_Dimension	The variable is a dimension name. Specify a dimension name for the *Variable_Name*.

The following code example creates a variable, then adds it to the rule.

```
&Variable = create Variable(&Constants.Variable_Type_Auto, "StartVal");
&FunCall.AddArgument(&Variable);
```

Variable Class Method

The following is the variable class method.

GenerateRule

Syntax

GenerateRule()

Description

Use the GenerateRule method to return a string that contains the rule for this object.

Copyright © 1988-2007, Oracle. All rights reserved.

Generally you wouldn't use this method on the individual RuleExpression objects, but instead would use the GenerateRule method on the RuleDefn object.

Parameters

None.

Returns

A string.

Variable Class Properties

In this section we discuss the variable class properties. The properties are described in alphabetical order.

Name

Description

This property specifies the name of the variable that was used to create the variable object, passed in the constructor.

This property is read-only.

Type

Description

This property specifies the type of the variable, passed in the constructor. The possible values are:

Value	Description
Variable_Type_Auto	A local variable.
Variable_Type_Expression	The variable is an expression. Specify an already instantiated rule expression object for the *Variable_Name.*
Variable_Type_Dimension	The variable is a dimension name. Specify a dimension name for the *Variable_Name*.

This property is read-only.

Analytic Model Metadata Classes Examples

The following are some examples of the most general cases of using the Analytic Model Metadata classes.

Copyright © 1988-2007, Oracle. All rights reserved.

Creating an Analytic Model Code Example

The following code creates an analytic model definition “from scratch” without retrieving an existing definition from the PeopleSoft Application Designer, adds parts such as dimensions, cubes, and cube collections, then saves the definition.

```
import PT_ANALYTICMODELDEFN:*;

Function CreateGENXModel

   Local PT_ANALYTICMODELDEFN:AnalyticModelDefn &Model;
   Local PT_ANALYTICMODELDEFN:UserFunctionDefn &UserFunc;
   Local PT_ANALYTICMODELDEFN:DimensionDefn &Dim;
   Local PT_ANALYTICMODELDEFN:CubeDefn &Cube;
   Local PT_ANALYTICMODELDEFN:CubeCollectionDefn &CubeColl;
   Local array of string &arr;

   &ACEMODELID = QE_ACE_META_WK.QE_ACE_MODELID.Value;

   &Model = create PT_ANALYTICMODELDEFN:AnalyticModelDefn(&ACEMODELID);

   &Model.Create();

   /* Model Properties */


   &Model.Description = QE_ACE_META_WK.DESCR;
   &Model.LongDescription = QE_ACE_META_WK.DESCR200;
   &Model.MaxDelta = QE_ACE_META_WK.QE_ACE_MAXDELTA_FL;
   &Model.MaxIterations = QE_ACE_META_WK.QE_ACE_MAXITER_FLD;

   If (QE_ACE_META_WK.QE_ACE_CIRCWARN_FL.Value = 0) Then
      &Circ = True;
   Else
      &Circ = False;
   End-If;

   If (QE_ACE_META_WK.QE_ACE_RESOLVE_FLD.Value = 0) Then
      &Resolve = True;
   Else
      &Resolve = False;
   End-If;

   &Model.ResolveCircularDeps = &Resolve;
   &Model.CircularFormulaWarn = &Circ;


   /* Add User Functions */

   &UserFunction = &Model.AddUserFunction("FILTERPRODUCTS");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   &Rule = &UserFunction.GetRule();
   &Rule.RuleString = "IF( ((UNIT_COST = 0) .AND. (UNITS_SOLD = 0) .AND. (PROD_⇒
SALES =0 )) , RETURN(0), RETURN(1))";
   &UserFunction.SetRule(&Rule);


   /* Add the Dimensions */
   &Dim = &Model.AddDimension("MONTH");

   &Dim = &Model.AddDimension("PRODUCTS");
   &Dim.TotalMemberName = "TOTAL";

   &Dim = &Model.AddDimension("PROD_CAT");

   &Dim = &Model.AddDimension("REGION");

   /* Add the cubes */
   &Cube = &Model.AddCube("PROD_SALES");
   &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
   &Cube.IsVirtual = False;
   &Cube.CalcAggregates = False;
   &Cube.Rule = "UNIT_COST * UNITS_SOLD";
   &Cube.AttachDimension("MONTH");
   &Cube.AttachDimension("PRODUCTS");
   &Cube.AttachDimension("REGION");

   &Cube = &Model.AddCube("SALES");
   &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
   &Cube.IsVirtual = False;
   &Cube.CalcAggregates = False;

   &Cube = &Model.AddCube("TGT_COST");
   &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
   &Cube.IsVirtual = False;
   &Cube.CalcAggregates = False;
   &Cube.AttachDimension("PRODUCTS");

   &Cube = &Model.AddCube("UNITS_SOLD");
   &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
   &Cube.IsVirtual = False;
   &Cube.CalcAggregates = False;
   &Cube.AttachDimension("MONTH");
   &Cube.AttachDimension("PRODUCTS");
   &Cube.AttachDimension("REGION");

   &Cube = &Model.AddCube("UNIT_COST");
   &Cube.FormatType = &Cube.AnalyticModel_Format_Number;
   &Cube.IsVirtual = False;
   &Cube.CalcAggregates = False;
   &Cube.AttachDimension("MONTH");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
    &Cube.AttachDimension("PRODUCTS");
    &Cube.AttachDimension("REGION");

    /* Add Cube Collections */
    &CubeColl = &Model.AddCubeCollection("REG_SALES_IN");
    &CubeColl.RecordName = "QE_BAM_FACT_TBL";
    &CubeColl.AttachCube("PROD_SALES");
    &CubeColl.AttachCube("UNITS_SOLD");
    &CubeColl.AttachCube("UNIT_COST");
    &CubeColl.SetFieldMapping("MONTH", "QE_BAM_MONTH_FLD", False);
    &CubeColl.SetPersistAggregate("MONTH", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("MONTH", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PRODUCTS", "QE_BAM_PRODUCT_FLD", False);
    &CubeColl.SetPersistAggregate("PRODUCTS", &CubeColl.AnalyticModel_AggrType_⇒
None);
    &CubeColl.SetDimSort("PRODUCTS", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("REGION", "QE_BAM_REGION_FLD", False);
    &CubeColl.SetPersistAggregate("REGION", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("REGION", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PROD_SALES", "QE_BAM_PRDSALES_FL", True);
    &CubeColl.SetFieldMapping("UNITS_SOLD", "QE_BAM_SALES_FLD", True);
    &CubeColl.SetFieldMapping("UNIT_COST", "QE_BAM_UNIT_FLD", True);

    &CubeColl = &Model.AddCubeCollection("REG_SALES_PROD");
    &CubeColl.RecordName = "QE_BAM_CCSMOKE";
    &CubeColl.AttachCube("PROD_SALES");
    &CubeColl.AttachCube("UNITS_SOLD");
    &CubeColl.AttachCube("UNIT_COST");
    &CubeColl.SetFieldMapping("MONTH", "QE_BAM_MONTH_FLD", False);
    &CubeColl.SetPersistAggregate("MONTH", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("MONTH", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PRODUCTS", "QE_BAM_PRODUCT_FLD", False);
    &CubeColl.SetPersistAggregate("PRODUCTS", &CubeColl.AnalyticModel_AggrType_⇒
None);
    &CubeColl.SetFilter("PRODUCTS", "FILTERPRODUCTS");
    &CubeColl.SetDimSort("PRODUCTS", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("REGION", "QE_BAM_REGION_FLD", False);
    &CubeColl.SetPersistAggregate("REGION", &CubeColl.AnalyticModel_AggrType_None);
    &CubeColl.SetDimSort("REGION", False, "", False, "", False, "");
    &CubeColl.SetFieldMapping("PROD_SALES", "QE_BAM_PRDSALES_FL", True);
    &CubeColl.SetFieldMapping("UNITS_SOLD", "QE_BAM_SALES_FLD", True);
    &CubeColl.SetFieldMapping("UNIT_COST", "QE_BAM_UNIT_FLD", True);

    &CubeColl = &Model.AddCubeCollection("TGT_COST_PROD");
    &CubeColl.RecordName = "QE_BAM_CC_TRGT";
    &CubeColl.AttachCube("TGT_COST");
    &CubeColl.SetFieldMapping("PRODUCTS", "QE_BAM_PRODUCT_FLD", False);
    &CubeColl.SetPersistAggregate("PRODUCTS", &CubeColl.AnalyticModel_AggrType_⇒
None);
    &CubeColl.SetDimSort("PRODUCTS", False, "", False, "", False, "");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   &CubeColl.SetFieldMapping("TGT_COST", "QE_BAM_TARGET_FLD", True);

   &Model.Save();
   /* &Valid = &Model.Validate(); */

   QE_ACE_META_WK.QE_BAM_PCSTATUS = "The Model " | &ACEMODELID | " was created and⇒
 saved.";

End-Function;
```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 4

Analytic Grid Classes

This chapter provides an overview of the analytic grid classes, and discusses:

- The analytic grid in PeopleCode
- Error handling
- Data types of analytic grid objects
- Scope of analytic grid objects
- Analytic grid reference

Note. Use the AnalyticGrid classes to manipulate the display of an analytic grid, that is, one associated with PeopleSoft Analytic Calculation Engine data. If you want to manipulate a grid control, use the grid classes.

See Also

Chapter 17, "Grid Classes," page 575

Understanding the Analytic Calculation Engine Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing, and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze, and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the Analytic Grid classes to manipulate the display of analytic calculation data on a page.
- Use the Analytic Type classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” page 43

Chapter 2, “Analytic Calculation Engine Classes,” page 11

Using the Analytic Grid in PeopleCode

The PeopleCode AnalyticGrid object is a reference to a page runtime object for the analytic grid. These particular page runtime objects aren’t present until the component is started.

Note. PeopleSoft builds a page grid one row at a time. Because the AnalyticGrid class applies to a complete grid, you can’t attach PeopleCode that uses the AnalyticGrid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

If you’re using the analytic grid within a secondary page, the runtime object for the grid isn’t created until the secondary page is run. The grid object can’t be obtained until then, which means that the earliest PeopleCode event you can use to activate a grid that’s on a secondary page is the Activate event for the secondary page.

The attributes you set for displaying an analytic grid remain in effect only while the page is active. When you switch between pages in a component, you have to reapply those changes *every time* the page is displayed.

In addition, the Activate event associated with a page fires every time the page is displayed. Any PeopleCode associated with that Activate event runs, which may undo the changes you made when the page was last active. For example, if you hide a grid column in the Activate event, then display it as part of a user action, when the user tabs to another page in the component, then tabs back, the Activate event runs again, hiding the grid column again.

If a user at runtime hides a column of a grid, tabs to another page in the component, then tabs back to the first page, the page is refreshed and the grid column is displayed again.

You can use the rowset class methods and properties on analytic grid data. You can access the data loaded by the analytic grid by accessing a rowset object after the grid is populated.

Use the analytic grid classes to access an analytic grid. Use the grid classes to access an ordinary grid.

See Also

Chapter 17, “Grid Classes,” page 575

Chapter 22, “Message Classes,” GetRowset, page 755

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateRowset

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetRowset

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer’s Guide, “PeopleCode and the Component Processor,” Activate Event

 Copyright © 1988-2007, Oracle. All rights reserved.

Using Freeze Column Mode

If you specify Freeze Column Mode for an analytic grid in application designer, the analytic grid isn't populated from the component buffer by default. It is your responsibility to write your application code so that it populates the rowset bound to the analytic grid.

Note. You should not be adding and deleting data from the analytic grid when you are not in Freeze Column Mode; this is an unsupported feature and might cause un-expected behavior.

In addition, no layout information is available, and there is no slicer, row, or column axis.

The following is an example of populating the analytic grid with data from a normal grid in Freeze Column Mode.

```
Local Rowset &RSAGRID;
Local Rowset &RSGrid;


/* Get hold of the rowset associated with normal grid whose primary record is QE_⇒
BAM_FACTTBL */
&RSGRID = GetLevel0()(1).GetRowset(Scroll.QE_BAM_FACTTBL);


/* Get hold of the rowset associated with Analytic grid whose primary record is QE_⇒
BAM_CCSMOKE */
&RSAGRID = GetLevel0()(1).GetRowset(Scroll.QE_BAM_CCSMOKE);

/* Flush out existing Data from the Analytic Grid */
&RSAGRID.Flush();

/* Copy data from Normal Grid to Analytic Grid in Freeze Column Mode*/
&RSGRID.CopyTo(&RSAGRID, Record.QE_BAM_FACTTBL, Record. QE_BAM_CCSMOKE);
```

Do not use the following methods with an analytic grid that is in Freeze Column Mode:

- Analytic grid GetCubeCollection method
- Analytic grid LoadData method
- Analytic grid SetAnalyticInstance method
- Analytic grid SetLayout method
- Analytic grid SlicerVisible property

Copyright © 1988-2007, Oracle. All rights reserved.

Error Handling

All the analytic type classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

See Also

Chapter 14, “Exception Class,” Try-Catch Sections, page 483

Data Types of the Analytic Grid Objects

Analytic grids are declared using the AnalyticGrid data type. For example,

```
Local AnalyticGrid &MyAnalyticGrid;
```

Analytic grid columns are declared using the AnalyticGridColumn data type. For example,

```
Local AnalyticGridColumn &MYGRIDCOL;
```

Scope of Analytic Grid Objects

Both the analytic grid and analytic grid column objects can be instantiated from PeopleCode only.

An analytic grid is a control on a page. You generally use these objects only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

Analytic grid objects can be of scope Local, Component or Global.

Analytic Grid Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetAnalyticGrid

AnalyticGrid Class Methods

In this section, we discuss the AnalyticGrid class methods. The methods are described in alphabetic order.

 Copyright © 1988-2007, Oracle. All rights reserved.

GetColumn

Syntax

GetColumn(*ColumnName*)

Description

Use the GetColumn method to instantiate an analytic grid column object.

Note. The properties for the grid column and the analytic grid column are the same. Any differences are indicated in the description for the property.

Specify the grid column name in the page field properties for that field, consisting of any combination of uppercase letters, digits and "#", "$", "@", and "_".

To specify a grid column name:

1. Open the page in PeopleSoft Application Designer, select the analytic grid and access the Analytic Grid control properties.
2. On the General tab, type the new grid name in Page Field Name.

Parameters

Parameter	Description
ColumnName	Specify the name of the column that you want to access.

Returns

An AnalyticGridColumn object.

Note. The properties for the grid column and the analytic grid column are the same. Any differences are indicated in the description for the property.

See Also

Chapter 17, “Grid Classes,” GridColumn Class, page 580

GetCubeCollection

Syntax

GetCubeCollection()

Description

Use the GetCubeCollection method to return a reference to the cube collection associated with the analytic grid.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A CubeCollection object.

See Also

Chapter 2, "Analytic Calculation Engine Classes," page 11

Chapter 2, "Analytic Calculation Engine Classes," CubeCollection Class, page 30

LoadData

Syntax

```
LoadData()
```

Description

Use the LoadData method to cause the system to get fresh data for the grid from the analytic calculation engine. Generally, you would use this method after you perform some operation, such as SetLayout, that might change the value or layout of the data.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

Parameters

None.

Returns

None.

SetAnalyticInstance

Syntax

```
SetAnalyticInstance(ID)
```

Description

Use the SetAnalyticInstance method to specify the analytic instance to be associated with this analytic grid.

An AnalyticGrid object can be bound only once to an analytic instance. If the SetAnalyticInstance method is called after the analytic grid is bound to an instance, the method call has no effect.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

Parameters

Parameter	Description
ID	Specify the analytic instance ID as a string.

Returns

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 2, "Analytic Calculation Engine Classes," AnalyticInstance Class Methods, page 14

Chapter 2, "Analytic Calculation Engine Classes," page 11

SetLayout

Syntax

```
SetLayout(&SlicerArray, &RowAxisArray, &ColumnAxisArray)
```

Description

Use the SetLayout method to set the layout for the three axes, slice, row, and column.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

If you specify No Drag drop mode for an analytic grid in PeopleSoft Application Designer, the analytic grid has the column axis, row axis and slicer axis, but the end user isn't allowed to change the layout by dragging and dropping elements between axes. However, you can still change the layout using the SetLayout method.

Note. You can specify a null value ("") for all the required parameters for this method.

Parameters

Parameter	Description
&SlicerArray	Specify an already instantiated array of string containing the list of fields to be put on the slice axis.
&RowAxisArray	Specify an already instantiated array of string containing the list of fields to be put on the row axis.
&ColumnAxisArray	Specify an already instantiated array of string containing the list of fields to be put on the column axis.

Returns

None.

AnalyticGrid Class Properties

In this section we discuss the AnalyticGrid properties. The properties are described in alphabetic order.

InActive

Description

Use this property to specify if the analytic grid is active or not. This property takes a boolean value, true if the grid is inactive, false otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

If you specify this property as true, the analytic grid is not displayed to the user and no data is fetched from the database.

This property is read-write.

Label

Description

Use this property to specify the label that appears as the title of the grid.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters. Always put any changes to labels in the Activate event for the page. This way the label is set every time the page is accessed.

This property is read-write.

ShowGridLines

Description

Use this property to specify whether grid lines are displayed with the analytic grid. This property takes a boolean value, true to show the lines, false otherwise.

This property is read-write.

SlicerVisible

Description

Use this property to specify whether the slicer is displayed with the analytic grid. This property takes a boolean value, true to show the slicer, false otherwise.

Note. Do not use this method with an analytic grid that is in Freeze Column Mode.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 5

Analytic Type Classes

This chapter provides an overview of the analytic type classes, and discuses:

- The analytic type classes
- Error handling
- Data types of the analytic type objects
- Scope of analytic type objects
- How to import the analytic type classes
- How to create an analytic type class object
- Analytic type reference

Understanding the Analytic Calculation Engine Classes

PeopleSoft Analytic Calculation Engine comprises a calculation engine plus several PeopleTools features which enable application developers to define both the calculation rules and the display of calculated data within PeopleSoft applications for the purposes of multi-dimensional reporting, data editing, and analysis.

More specifically, developers create *analytic models* in order to define the rules which are used to calculate data. Developers also create PeopleSoft Pure Internet Architecture pages with *analytic grids* in order to display the data within PeopleSoft applications. End users view, analyze and make changes to this data. When end users save their changes, PeopleSoft Analytic Calculation Engine recalculates the data and saves the calculated data to the database.

PeopleCode enables developers to manipulate analytic calculation data as follows:

- Use the Analytic Calculation Engine classes to either retrieve or specify data in an instance of an analytic model loaded into the system, and also to calculate (or recalculate) cube values.
- Use the Analytic Calculation Engine metadata classes to manipulate an analytic model definition. For example, you can add cubes to a cube collection or rename an existing user function for a model.
- Use the analytic grid classes to manipulate the display of analytic calculation data on a page.
- Use the analytic type metadata classes to manipulate an analytic type definition. For example, you can specify a new analytic model for an analytic type definition.

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” page 43

Chapter 4, “Analytic Grid Classes,” page 129

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine

Copyright © 1988-2007, Oracle. All rights reserved.

Using the Analytic Type Classes

You can create analytic type definitions using PeopleSoft Application Designer. PeopleSoft provides the analytic type classes for accessing these definitions at runtime.

You can also create an analytic type definition in PeopleCode, using the Create method, then saving it to the database using the Save method. After you save the definition, you can access it using PeopleSoft Application Designer.

You only need to create an instance of the analytic type. All other objects are instantiated from the analytic type object.

Once the AnalyticTypeDefn object is created, you can then execute either the Get or Create methods to "instantiate" the analytic type object.

Error Handling

All the analytic type classes throw PeopleCode exceptions for any fatal error that occurs in the execution of the operation. PeopleSoft recommends enclosing your analytic model programs in try-catch statements. This way, if your program catches the exception, the message set and message number that are associated with the exception object indicate the error.

See Also

Chapter 14, "Exception Class," Try-Catch Sections, page 483

Data Types of the Analytic Type Objects

Every PeopleSoft analytic type object is declared as its own data type, that is, AnalyticTypeDefn objects are declared as type AnalyticTypeDefn, AnalyticTypeModelDefn objects are declared as type AnalyticTypeModelDefn, and so on.

The following are the data types for the PeopleSoft analytic type classes:

- AnalyticTypeDefn
- AnalyticTypeModelDefn
- AnalyticTypeRecordDefn

Scope of Analytic Type Objects

The analytic type objects can only be instantiated from PeopleCode.

These objects can be used anywhere you have PeopleCode, that is, in a PeopleSoft Application Engine program, an application class, record field PeopleCode, and so on.

 Copyright © 1988-2007, Oracle. All rights reserved.

Analytic type objects can be of scope Local, Component or Global.

How to Import the Analytic Type Classes

The analytic type classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the analytic type classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_ANALYTICTYPEDEFN contains the following subclasses:

- AnalyticTypeDefn
- AnalyticTypeModelDefn
- AnalyticTypeRecordDefn

The import statement you should use should be as follows:

```
import PT_ANALYTICTYPEDEFN:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available.

See Also

Chapter 6, “Application Classes,” page 153

How to Create an Analytic Type Class Object

After you’ve imported the analytic type classes, you need to instantiate an instance of the AnalyticTypeDefn class, using the constructor for that class and the Create function. After you create the object, you must populate it using either the Get or Create method.

The following example creates an AnalyticTypeDefn object from the QE_ACE_ALLFUNCTION analytic type definition.

```
Local PT_ANALYTICTYPEDEFN:AnalyticTypeDefn &AnalyticType;

&AnalyticType = create PT_ ANALYTICTYPEDEFN:AnalyticTypeDefn("QE_ACE_ALLFUNCTION");
```

See Also

Chapter 3, “Analytic Calculation Engine Metadata Classes,” Analytic Calculation Engine Metadata Classes Constructor, page 47

Copyright © 1988-2007, Oracle. All rights reserved.

Analytic Type Classes Constructor

You must use the constructor for the AnalyticTypeDefn class to instantiate an instance of that class. All other objects are instantiated from this class.

AnalyticTypeDefn

Syntax

AnalyticTypeDefn(*TypeName*)

Description

Use the AnalyticTypeDefn constructor to create an AnalyticTypeDefn.

Once the AnalyticTypeDefn object is created, you can then execute either the Get or Create methods to instantiate the model.

Note. The Delete and Rename methods can only be executed before a model is instantiated.

Parameters

Parameter	Description
TypeName	Specify the name of an analytic type definition.

Returns

An AnalyticTypeDefn object.

Example

```
&Model = create AnalyticTypeDefn("QE_ACE_ALLFUNCTION");
```

See Also

Chapter 5, “Analytic Type Classes,” AnalyticTypeDefn, page 140

AnalyticTypeDefn Class Methods

This section describes the analytic type definition class methods. The methods are discussed in alphabetical order.

AddModel

Syntax

AddModel(*ModelName*, *ModelType*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AddModel method to add either an analytic model or optimization model to the analytic type definition. If the model doesn't exist, this method fails.

Parameters

Parameter	Description
ModelName	Specify the name of an existing model.
ModelType	Specify the model type. Values are:

Value	Description
%ModelType_ACE	The model is an Analytic Calculation Engine.
%ModelType_OPT	The model is an optimization.

Returns

The new AnalyticTypeModelDefn object.

See Also

Chapter 5, "Analytic Type Classes," DeleteModel, page 143 and Chapter 5, "Analytic Type Classes," GetModelNames, page 145.

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 148

AddRecord

Syntax

AddRecord(*RecordName*)

Description

Use the AddRecord method to add a record to the analytic type definition. If the record definition doesn't exist in PeopleSoft Application Designer, or if the record already exists in this analytic type definition, this method fails.

Parameters

Parameter	Description
RecordName	Specify the name of the record that you want to add.

Returns

An AnalyticTypeRecordDefn object that represents the record.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 5, “Analytic Type Classes,” DeleteRecord, page 144 and Chapter 5, “Analytic Type Classes,” GetRecordNames, page 146.

Chapter 5, “Analytic Type Classes,” AnalyticTypeRecordDefn Class Methods, page 149

Create

Syntax
```
Create()
```

Description
Use the Create method to create, and instantiate, a new analytic type. If the analytic type already exists, an exception is thrown.

Parameters
None.

Returns
None.

See Also
Chapter 5, “Analytic Type Classes,” Save, page 147

Chapter 14, “Exception Class,” Try-Catch Sections, page 483

CopyTo

Syntax
```
CopyTo(NewAnalyticTypeName)
```

Description
Use the CopyTo method to copy the current AnalyticTypeDefn object to the AnalyticTypeDefn object with the specified name. This method fails if the analytic type specified with *NewAnalyticTypeName* already exists.

Parameters

Parameter	Description
NewAnalyticTypeName	Specify the name of the new analytic type definition that you want to create.

Returns
The new AnalyticTypeDefn object.

See Also
Chapter 5, “Analytic Type Classes,” Create, page 142 and Chapter 5, “Analytic Type Classes,” Save, page 147.

 Copyright © 1988-2007, Oracle. All rights reserved.

Delete

Syntax

```
Delete()
```

Description

Use the Delete method to delete the analytic type definition. You must use this method immediately after you create the AnalyticType object, before you instantiate it, that is, before you use either the Get or Create method.

Note. This method executes immediately, and deletes the definition from PeopleSoft Application Designer.

Parameters

None.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Create, page 142 , Chapter 5, "Analytic Type Classes," Get, page 144 and Chapter 5, "Analytic Type Classes," Save, page 147.

DeleteModel

Syntax

```
DeleteModel(ModelName)
```

Description

Use the DeleteModel method to delete the specified model from the analytic type definition.

Note. The model is not deleted from the analytic type definition until you use the Save method.

Parameters

Parameter	Description
ModelName	Specify the name of a model that exists in the analytic type definition.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," AddModel, page 140 and Chapter 5, "Analytic Type Classes," GetModelNames, page 145.

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 148

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteRecord

Syntax

DeleteRecord(*RecordName*)

Description

Use the DeleteRecord method to delete a record from the analytic type definition.

Note. The record deleted from the analytic type definition until you use the Save method.

Parameters

Parameter	Description
RecordName	Specify the name of the record that you want to remove from the analytic type definition.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," AddRecord, page 141 and Chapter 5, "Analytic Type Classes," GetRecordNames, page 146.

Chapter 5, "Analytic Type Classes," AnalyticTypeRecordDefn Class Methods, page 149

Get

Syntax

Get()

Description

Use the Get method to instantiate an existing analytic type definition as an AnalyticTypeDefn object. If the analytic type doesn't exist, this method throws an exception.

Parameters

None.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Create, page 142 and Chapter 5, "Analytic Type Classes," Save, page 147.

GetModel

Syntax

GetModel(*ModelName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetModel method to return a reference to an AnalyticTypeModelDefn object.

This method fails if the model specified by *ModelName* doesn't exist.

Parameters

Parameter	Description
ModelName	Specify the name of the analytic type model definition that you want to access.

Returns

An AnalyticTypeModelDefn object.

See Also

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 148

GetModelNames

Syntax

```
GetModelNames()
```

Description

Use the GetModelNames method to return an array of string containing all the model names in this analytic type definition.

Parameters

None.

Returns

An array of string.

See Also

Chapter 5, "Analytic Type Classes," AddModel, page 140 and Chapter 5, "Analytic Type Classes," DeleteModel, page 143.

Chapter 5, "Analytic Type Classes," AnalyticTypeModelDefn Class Properties, page 148

GetRecord

Syntax

```
GetRecord(RecordName)
```

Description

Use the GetRecord method to return a reference to an AnalyticTypeRecordDefn object.

This method fails if the record specified by *RecordName* doesn't exist.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RecordName	Specify the name of the record that you want to access.

Returns

An AnalyticTypeRecordDefn object.

See Also

Chapter 5, “Analytic Type Classes,” AnalyticTypeRecordDefn Class Methods, page 149

GetRecordNames

Syntax

```
GetRecordNames()
```

Description

Use the GetRecordNames method to return an array of string containing the names of all the records associated with the analytic type definition.

Parameters

None.

Returns

An array of string.

See Also

Chapter 5, “Analytic Type Classes,” AddRecord, page 141 and Chapter 5, “Analytic Type Classes,” DeleteRecord, page 144.

Chapter 5, “Analytic Type Classes,” AnalyticTypeRecordDefn Class Methods, page 149

Rename

Syntax

```
Rename(NewAnalyticTypeName)
```

Description

Use the Rename method to rename an existing analytic type definition to a new name. You must use this method immediately after you create the AnalyticType object, before you instantiate it, that is, before you use either the Get or Create method. The new name is not saved to the database until you use the Save method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
NewAnalyticTypeName	Specify the new name for the analytic type definition.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," Save, page 147

Save

Syntax

```
Save()
```

Description

Use the Save method to save any changes that were made to the AnalyticTypeDefn object to the database.

Parameters

None.

Returns

None.

AnalyticTypeDefn Class Properties

In this section we discuss the AnalyticTypeDefn class properties. The properties are described in alphabetic order.

AppClassPath

Description

Use this property to specify the full name of the application package that contains the Create, Copy and Delete methods used with this analytic type definition.

This property is read-write.

Comments

Description

Use this property to specify comments for the analytic type.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Description

Description

Use this property to specify a description for the analytic type.

This property is read-write.

Name

Description

This property specifies the name of the AnalyticTypeDefn.

This property is read-only.

OwnerID

Description

This property specifies the owner ID for this AnalyticTypeDefn.

This property is read-write.

AnalyticTypeModelDefn Class Properties

In this section we discuss the AnalyticTypeModelDefn class properties. The properties are described in alphabetic order.

Name

Description

This property specifies the name of the analytic type model.

This property is read-only.

Type

Description

This property specifies the type of the analytic type model. Values are:

Value	Description
%ModelType_ACE	The model is an Analytic Calculation Engine.
%ModelType_OPT	The model is an optimization.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

AnalyticTypeRecordDefn Class Methods

In this section we describe the AnalyticTypeRecordDefn class methods. The methods are discussed in alphabetic order.

GetSelectedField

Syntax

```
GetSelectedField()
```

Description

Use the GetSelectedField to return an array of string containing the names of the fields that are selected for this record.

Parameters

None.

Returns

An array of string.

See Also

Chapter 5, "Analytic Type Classes," SetSelectedField, page 149

SetSelectedField

Syntax

```
SetSelectedFields(FieldName)
```

Description

Use the SetSelectedField method to specify the name of the field that is selected for this record.

Parameters

Parameter	Description
FieldName	Specify the name of the field that you want to select.

Returns

None.

See Also

Chapter 5, "Analytic Type Classes," GetSelectedField, page 149

Copyright © 1988-2007, Oracle. All rights reserved.

UnsetSelectedField

Syntax

UnsetSelectedField(*FieldName*)

Description

Use the UnsetSelectedField method to unselect the selected field.

This method throws an exception if the field specified by *FieldName* isn't selected.

Parameters

Parameter	Description
FieldName	Specify the name of the field that you want to deselect.

Returns

None.

AnalyticTypeRecordDefn Class Properties

In this section we discuss the AnalyticTypeRecordDefn class properties. The properties are described in alphabetic order.

All of the properties listed here can also be set for the analytic type definition using PeopleSoft Application Designer.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Analytic Calculation Engine, "Viewing and Debugging Analytic Models," Understanding Analytic Model Properties

Callback

Description

Use this property to specify whether this record has a callback. This property takes a boolean value, true if the record has a callback, false otherwise.

This property is read-write.

Description

Description

Use this property to specify a description for the record.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property specifies the name of the record.

This property is read-only.

Readable

Description

Use this property to specify whether this record is readable. This property takes a boolean value, true if the record is readable, false otherwise.

This record is read-write.

ReadOnce

Description

Use this property to specify whether this record is specified as read once. This property takes a boolean value, true if the record is only read once, false otherwise.

This property is read-write.

ScenarioManaged

Description

Use this property to specify if the record is specified as scenario managed. This property takes a boolean value, true if the record is scenario managed, false otherwise.

This property is read-write.

SyncOrder

Description

This property specifies the synchronization order for the record. It returns an integer value.

This property is read-write.

Writeable

Description

Use this property to specify if a record is specified as writeable. This property takes a boolean value, true if the record is writeable, false otherwise.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

AnalyticType Classes Example

The following is a typical example of using the AnalyticType classes.

```
import PT_ANALYTICTYPEDEFN:*;

Local PT_ANALYTICTYPEDEFN:AnalyticTypeDefn &AnalyticType;

Local PT_ANALYTICTYPEDEFN:AnalyticTypeRecordDefn &AnalyticTypeRecordDefn;

&AnalyticType = create PT_ ANALYTICTYPEDEFN:AnalyticTypeDefn("BB_TEST");

&AnalyticType.Create();

&AnalyticType.AddModel("QE_BAM_GENX", %ModelType_ACE);

&AnalyticTypeRecordDefn = &AnalyticType.AddRecordl("QE_BAM_CCSMOKE");

&AnalyticTypeRecordDefn = &AnalyticType.AddRecordl("QE_BAM_FACTTBL");

&AnalyticType.Save();
```


Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 6

Application Classes

This chapter provides an overview of application classes and discusses the following topics:

- Application classes general structure.
- Import declarations.
- Self-reference.
- Property overrides.
- Superclass reference.
- Downcasting.
- Exception handling.
- Design of base classes.
- Declaration of application classes.
- Scope of an application class.
- Application classes built-in functions and language constructs.

Understanding Application Classes

You can create Application Packages in PeopleSoft Application Designer. These *packages* contain application classes (and may contain other packages also). An application class, at its base level, is just a PeopleCode program. However, using the Application Packages, you can create your own classes, and extend the functionality of the existing PeopleCode classes.

The application classes provide capability beyond that offered by the existing PeopleCode classes. Unlike the existing classes, a subclass can inherit all the properties and methods of the class it extends. This provides all the advantages of true object-oriented programming:

- easier to debug because all the pieces are separate
- easier to maintain because functionality is gathered into a single place
- extensible by subclass

In addition, the application classes provide more structure. Using the Application Packages, you have a clear definition of each class, as well as its listed properties and methods. This makes it easier to create a complex program that uses many functions.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Creating Application Packages and Classes"

Copyright © 1988-2007, Oracle. All rights reserved.

When Would You Use Application Classes?

Use Application classes to help structure your existing PeopleCode functions. For example, prior to Application classes, most PeopleCode functions were put into FUNCLIB records. However, in larger, more complex programs, it is sometimes difficult to find (let alone maintain) a function. You could make every function a class in a package with very little reworking of your PeopleCode (as well as making the different fields in a record a package, and combining them into a single larger package.) This provides a better visual interface for grouping your functions.

In addition, use Application classes when you have to do generic processing, and just the details are different. For example, suppose one of the processes for your application is reading a file object and doing bulk insert. The process could all be contained in a single package, with the classes and subclasses providing all the details, different kinds of reading for different types of files, different inserts based on the record, and so on.

One of the main differences between a class and a function call is that the call to the class is dynamic, like a function variable, but more closely controlled. All the calls to the class (methods) must have the same signature.

For example, suppose you want to provide a more generic sort, with comparison function at the end of it. You want to use the array class Sort method, but your process has to be generic: you aren't certain if you're comparing two records or two strings. You could define a class that had as its main method a comparison, and return a -1, 0, or 1. Then write your personalized sort, extending the array class method Sort.

Another use is for business processing. Think in terms of core functionality, with vertical product solutions that extend the basic processing by providing specifics, in other words, by extending the general classes with subclasses. This could be appropriate for sales force automation or order entry.

Generally, use application classes where you can extract the common functionality.

Note. Do not extend a SOAPDoc with an application class. This is currently not supported.

Application Classes General Structure

All application classes are contained in a package. All application classes are composed of PeopleCode. In this section, we discuss the general form of an application class, which is:

- Class name
- Class extensions
- Declaration of public external interface
- Declaration of protected instance variables and methods
- Declaration of private instance variables and methods
- Definition of methods
- Constructors

This division enables a separation of the following:

- what the class provides to other classes
- what is applicable to the entire class
- what is applicable to the definition of a method

The data types used in an application class (for methods parameters, return values, and so on) can be any PeopleCode types, including application classes. Likewise, application classes can be used anywhere in a PeopleCode program where a general data type can be used.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

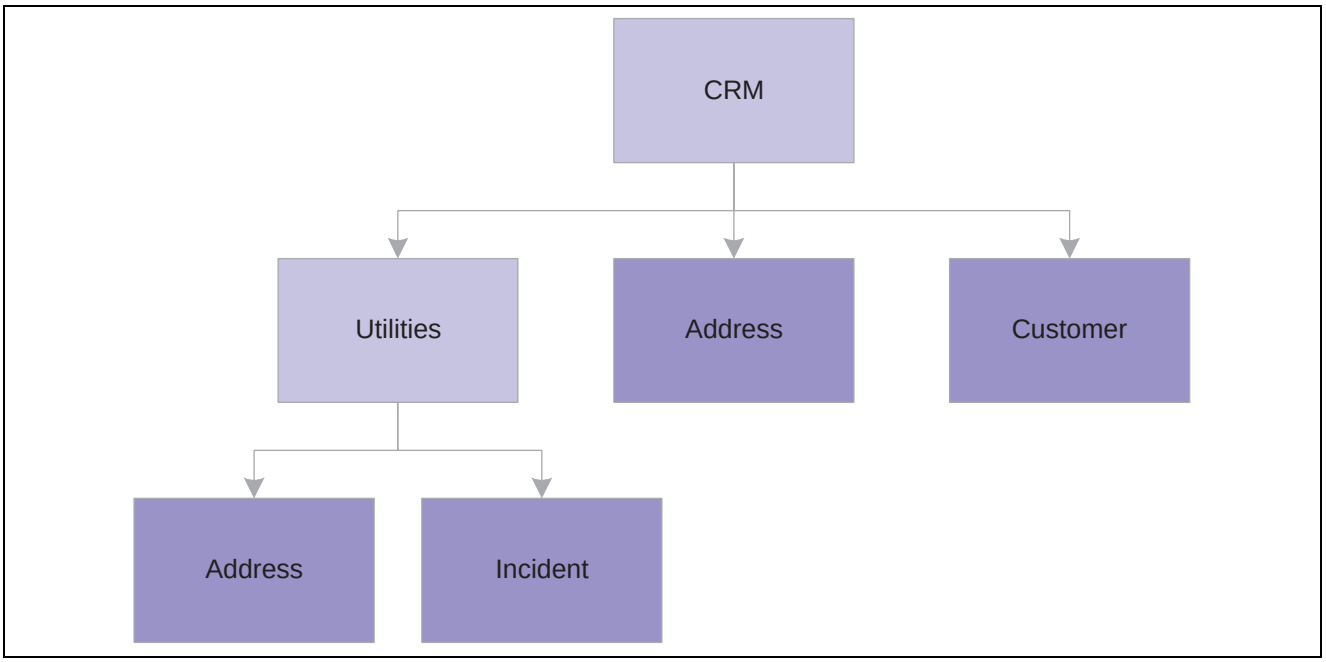
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding the PeopleCode Language," Data Types

Class Name

Application classes have a fully qualified name that is formed hierarchically by the name of the top-level package that contains them, the package that contains that package, and so on down to the short class name, that is, the one specified when the class was created in PeopleSoft Application Designer, using a colon for the separator between names.

The fully qualified name of the class must be unique, as this is what identifies the class. The short name of the class does not have to be unique.

For example, suppose package CRM contains the application classes Address and Customer as well as the package Utilities, which in turn contains the application classes Address and Incident:



Application classes example

There are four distinct application classes in this example:

- CRM:Address
- CRM:Customer
- CRM:Utilities:Address
- CRM:Utilities:Incident

Note. If you change the name of a package or an application class, that name change is *not* automatically propagated through all your PeopleCode programs. You must make the change manually in your programs. If you specify an incorrect name for a package or application class, you receive a warning when you try to save the PeopleCode.

Copyright © 1988-2007, Oracle. All rights reserved.

Class Extension

Extension represents the "is-a" relationship. When a class extends another class, it's called a *subclass* of that class. Conversely, the class being extended is called a *superclass* of that subclass.

A subclass inherits all of the public methods and properties (collectively called *members*) of the class it extends. These members can be overridden by declarations of methods and properties in the subclass.

Note. Application classes have no multiple inheritance (that is, being able to extend more than one class.)

Type checking in PeopleCode (both at design time and runtime) does strong type checking of application classes, tracking each application class as a separate type. A subclass can be used as the class it extends, because it implements the public interfaces of its superclass. This is called *subtyping*.

In the following example, the class Banana extends the class Fruit. Banana is the *subclass* of Fruit. From Fruit, you can extend to Bananas. However, from Bananas you can't extend to Fruit. Another way to think about this is you can only extend from the general to the specific, not the other way around.

```
class Fruit
   method DoFruit();
   property number FruitNum instance;
end-class;

class Banana extends Fruit
   method DoBanana();
   property number BananaNum instance;
end-class;
```

The following code shows a correct way to assign a class object, because Banana is a *subtype* of Fruit.

```
local Fruit &Bref = Create Banana();
```

The following is *not* correct. Banana is a subtype of Fruit. Fruit is *not* a subtype of Banana. This assignment causes an error at design time.

```
local Banana &Fref = Create Fruit();
```

Before you can extend a class, you must first import it. You must import all class names you use in a PeopleCode program before you use them.

See Also

Chapter 6, "Application Classes," Import Declarations, page 170

Declaration of Public External Interface

The names of the members (that is, all the methods and properties of a class) must be unique only within that class. Different classes can have members with the same name. As members are used only with objects, and each object knows its class, the member that's accessed is the one associated with that class.

For example, more than one class has the property Name which returns the name of the object executing the property. There isn't any confusion, because the field Name property returns the name of a field, while the record name property returns the name of a record. No class has more than one property called Name.

Note. Within a class, each member name *must* be unique.

 Copyright © 1988-2007, Oracle. All rights reserved.

The public part of a class declaration specifies the methods and properties that the class provides to other PeopleCode programs. These methods and properties are *dynamically bound*, that is, the actual method that is called depends on the actual class of the object, as it might be an overridden method or property from a subclass.

In the following code example, the text in bold indicates the public part of the class declaration.

```
/* generic building class */
class BuildingAsset
   method Acquire ();
   method DisasterPrep();
end-class;
```

Access Control and the Declaration of Protected Properties and Methods

Application class objects have private and public access control. Properties or methods that are declared outside the "private" declaration section have public access control. This means that they can be referenced and used by any PeopleCode. This access is of course subject to access controls such as "readonly". Private properties and methods are private to the class, not instance, and can only be referenced by objects of this application class.

Between these two access control schemes of public and private, lies the concept of protected methods and properties. Protected methods and properties can be accessed only by objects of this application class and those derived from this application class. Use protected methods or properties when you want to hide them from outside use, but allow the flexibility of using them in derived classes.

The declarations of protected variables and methods are done within the class declaration, before the declaration of private variables and methods. You can use protected instance variables in interface classes. Protected methods and properties can be overridden by subclasses.

Most of the time your design can be implemented through the use of private methods and properties without resorting to the use of protected methods and properties. The following examples demonstrates the rules and some of the subtleties about when to use protected methods and properties.

```
class A;
   method A();
   property string Q;
protected
   method MP() Returns string;
   property string p;
end-class;

method A;
   &p = "Class A: property p";
end-method;

method MP
   /+ Returns String +/
   Return "Class A: method MP";
end-method;

====================

class B extends A;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   method B();
   method M(&X As C);
   method m2(&Aobj As A);
   /* property string P; */
protected
   property string Q;
   method MP() Returns string;
end-class;

method B;
   %Super = (create A());
   &Q = "Class B: property Q";
   /* &P = "Class B: property P";*/
end-method;

method M
/+ &X as FOXTEST:C +/;

   MessageBox(0, "", 0, 0, "In B:M %This.P=" | %This.p);
/* %This.p is class A property P */
   MessageBox(0, "", 0, 0, "In B:M %This.Q=" | %This.Q);
/* %This.q is class B property Q */
   MessageBox(0, "", 0, 0, %This.MP());
/* %This.MP() calls class B method MP */

   if &X.p = "Error" then
   end-if;
/* error: cannot reference &X.p since class B is
not involved in the implementation of class C */
end-method;

method m2
/+ &Aobj as FOXTEST:A +/
/* MessageBox(0, "", 0, 0, "In B:M2 &Aobj.P=" | &Aobj.p);
/* Error: cannot reference &Aobj.p from class B
since class B is not involved in its implementation */

End-method;

method MP
   /+ Returns String +/
   Return "Class B: method MP";
end-method;

=======================

class C extends A;
   method C();
protected
   property string P;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
end-class;

method C;
   %Super = (create A());
   &P = "Class C: property P";
end-method;
==========================

/* AE program */
import FOXTEST:*;

Local A &A = (create A());

Local object &obj = &A;
MessageBox(0, "", 0, 0, "In AEMINITEST: &Obj.p=" | &obj.P);
/* run time error: anonymous access through type
object not allowed*/
```

The following example also illustrates some of the rules surrounding protected access control.

```
import FOXTEST:Point3d;

class Point
   method Point(&X1 As integer, &Y1 As integer);
   method Warp(&A As Point3d);
protected
   property integer x;
   property integer y;
end-class;

method Point
   /+ &X1 as Integer, +/
   /+ &Y1 as Integer +/;

   %This.x = &X1;
   %This.y = &Y1;
end-method;

method Warp
   /+ &A as FOXTEST:Point3d +/
   /* Local Integer &temp = &A.Z;  ERROR cannot access &A.Z */
end-method;
================================
import FOXTEST:Point;

class Point3d extends Point
   method Point3d(&X1 As integer, &Y1 As integer, &Z1 As integer);
   method Delta(&P As Point);
   method Delta3d(&Q As Point3d);
protected
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   property integer z;
end-class;

method Point3d
   /+ &X1 as Integer, +/
   /+ &Y1 as Integer, +/
   /+ &Z1 as Integer +/;
   %Super = (create Point(&X1, &Y1));
   %This.z = &Z1;
end-method;

method Delta
   /+ &P as FOXTEST:Point +/
   /* &P.x = %This.x;
ERROR cannot access &P.x since while Point3d
(the class in which references to fields x and y occur)
is a subclass of Point (the class in which x and y are declared),
it is not involved in the implementation of Point (the type of parameter p)*/
   /* &P.y = %This.y;
ERROR cannot access &P.y. Same reason as above */
end-method;

method Delta3d
/+ &Q as FOXTEST:Point3d +/
/* The protected members of Q can be accessed because the class point3d is a⇒
 subclass of Point and is involved in the implementation of Point3d */
&Q.x = %This.x;
&Q.y = %This.y;
   &Q.z = %This.z;

end-method;
```

Declaration of Private Instance Variables and Methods

The private part of a class declaration gives the declaration of any private methods, instance variables, and private class constants. Private methods *cannot* be overridden by subclasses, because they are completely private to the declaring class. Likewise, the instance variables can't be overridden.

The declarations of private variables and methods are done within the class declaration.

Note. You cannot have private methods or instance variables in an interface type of class.

In the following example, there are both public as well as private methods and properties. Any method that is declared before the keyword Private in the initial class declaration is public. Any method declared after the keyword Private is private. Properties are only declared in Public. Instances and Constants are only declared in Private

In the following example, the class Example extends the class ExampleBase. It has both public as well as private methods and properties. It also defines the private method in the definition of methods section of the code (the private definitions are marked **like this**.)

 Copyright © 1988-2007, Oracle. All rights reserved.

```
import PWWPACK:ExampleBase;

class Example extends ExampleBase
   method Example();
   method NumToStr(&Num As number) Returns string;
   method AppendSlash();
   property number SlashCount get;
   property number ImportantDayOfWeek get set;
   property string SlashString readonly;
   property date ImportantDate;
private
   method NextDayOfWeek(&Dow As number) Returns date;
   Constant &Sunday = 1;
   instance string &BaseString;
end-class;

Global string &CurrentBaseString;

/* Method declarations */
method NumToStr
   return String(&num);
end-method;

method AppendSlash
   &SlashString = &SlashString | "/";
end-method;

get SlashCount
   Return Len(&SlashString) - Len(&BaseString);
end-get;

get ImportantDayOfWeek
   Return Weekday(&ImportantDate);
end-get;

set ImportantDayOfWeek
   &importantdate = %This.nextdayofweek(&newvalue);
end-set;

/* Private method. */
method nextdayofweek
   Return &ImportantDate + (&dow - Weekday(&ImportantDate));
end-method;

/* Constructor. */
method Example
   &BaseString = &CurrentBaseString;
   &SlashString = &BaseString;
   &ImportantDate = Date(19970322);
end-method;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Definition of Methods

After the declaration of any global or component variables and external functions needed by the methods, comes the actual definition of the methods. This section discusses some of the differences in how application programs are processed by the system, and how application class method parameters are processed.

- The system *never* skips to the next top-level statement.
- Application programs generally pass parameters by value, which is not the same as for existing functions.
- Parameter passing with object data types is by reference.
- Application programs use the out specifier to pass a parameter by reference.

Not Skipping to The Next Top-Level Statement

For existing functions and programs, the system skips to the next top-level statement in some circumstances, such as a field not found error. This never happens in an application class program. The code is always executed or an error is produced.

Passing Parameters in Application Class Methods

The parameters to a method are passed by value in the absence of an *out* specification in the parameter list. However, if a parameter list has an out specification, that argument (when used in the call of a method) must be a value that can be assigned something, and is passed by *reference*. In particular this means you cannot pass an object property to a method which requires the parameter to be an "out" parameter. Application class properties are always passed by value.

For example:

```
/* argument passed by reference */
method increment(&value as number out);


/* argument passed by value */
method increment(&value as number);
```

This is in contrast with existing PeopleCode functions that pass *all* parameters by reference when they can be assigned something. This means the method signature is a specification of which parameters can be updated by the method. This makes the code easier to debug and maintain because it's easy to see what can be updated and what can't.

The following is an example of code that increments a variable by one.

```
local Number &Mine;
&Mine = 1;
&obj.Increment(&Mine);
```

&Mine now has the value 2.

You cannot pass properties by reference. For example, supposed MyProp is a property of class Xan, a statement that requires a reference doesn't work if you supply the property. The following code will always fail because &Xan.MyProp is always passed by value and the MySqlExec method requires it to be passed by reference (so a value can be returned.)

```
MySqlExec("select from bar", &Xan.MyProp)
```

This makes sense because the semantics of an object require that you can only change a property with standard property references, not as a side affect of some other action.

 Copyright © 1988-2007, Oracle. All rights reserved.

Passing Parameters with Object Data Types

Parameters with object data types are always passed by reference:

```
/* argument passed by reference  */
method storeInfo(&f as File);
```

If you specify the out modifier for a method parameter with an object data type, it becomes a *reference parameter*. This means that the parameter variable is passed by reference instead of the object that it is pointing at when passed.

For example, if you pass an object parameter with the out modifier:

```
method myMethod(&arg as MyObjectClass);

Local MyObjectClass &o1 = create MyObjectClass("A");
Local MyOtherObjectClass &o2 = create MyOtherObjectClass();

&o2.myMethod(&o1);
```

And inside myMethod this occurs:

```
Method myMethod
   &arg  =  create MyObjectClass("B");
end-method;
```

Since the method argument is reassigned within the body of myMethod, &o1 does not point at the new instance of MyObjectClass (initialized with "B") after the method call completes. This is because &o1 still references the original instance of MyObjectClass.

However, if &o1 had been passed with the out modifier, after the method call completes, &o1 points at whatever the parameter was last assigned to; in this case, the new instance of MyObjectClass. The parameter, rather than the object, is passed by reference.

Using the Out Specification for a Parameter

In the following example, a class, AddStuff, has a single public method, DoAdd. This adds two numbers together, then assigns them as different numbers. In the signature of the method declaration, the first parameter is *not* declared with an out statement, while the second one is.

```
class AddStuff
   method DoAdd(&P1 as number, &P2 as number out);
end-class;

method DoAdd
   &X = &P1 + &P2;
   &P1 = 1;
   &P2 = 2;
end-method;
```

In the following PeopleCode example, an object named &Aref is instantiated from the class AddStuff. Two parameters, &I and &J are also defined.

```
local AddStuff &Aref = Create AddStuff();
local number &I = 10;
local number &J = 20;
```

The following code example is correct. &J is changed, because of the *out*statement in the method signature, and because the value is being passed by reference. The value of &I is *not* updated.

Copyright © 1988-2007, Oracle. All rights reserved.

```
&Aref.DoAdd(&I, &J); /* changes &J but not &I */
```

The following code example causes a design time error. The second parameter must be passed *by reference*, not by value.

```
&Aref.DoAdd(10, 20); /* error - second argument not variable */
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding the PeopleCode Language"

Declaration of Abstract Methods and Properties

You can declare methods and properties as abstract, that is, a method or property that has a signature which specifies the parameters and results, but has no implementation in the class that defines it. The implementation may occur in one of the classes that extend the class where the method or property is initially defined.

Abstract methods and properties could be used, for example, when your application wants to provide a means for in-field customization. Your application might deliver a base class specifying the abstract methods and properties and you would allow the customization to complete those methods and properties by using a class that would extend the base class and provide implementations for some or all of the abstract methods and properties.

For a class that only contains abstract methods and properties, that is, a pure interface class, you would define a PeopleCode interface, instead of a class. You define a PeopleCode interface by using the keyword Interface instead of Class. In an interface, all the methods and properties are abstract by default.

The following example illustrates the use of abstract methods and properties. Class MyInterface is the base class which has mostly abstract methods and properties. However it is fully specified by the class MyImplementation which provides and implementation for all the methods and properties.

```
class MyInterface
   method MyMethod1() abstract;
method MyMethod2() Returns string abstract;
method MyMethod3();
   property string myproperty1 abstract readonly;
   property number myproperty2 abstract;
property number myproperty3 abstract;
end-class;
method MyMethod3
end-method;
/* ---------------------------------------------------------------------- */
import FOXTEST:MyInterface;

class MyImplementation extends MyInterface;
   method MyImplementation();
   method MyMethod1();
   method MyMethod2() Returns string;
   property string myproperty1 readonly;
   property number myproperty2;
   property number myproperty3;
end-class;

method MyImplementation
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   %Super = Create MyInterface();[Ugh.]
end-method;

method MyMethod1
   /* dummy */
end-method;

method MyMethod2
   /+ Returns String +/
   Return "MyMethod2's implementation is this";
end-method;
```

Considerations Using Abstract Methods and Properties

Be aware of the following considerations when using abstract methods or properties.

- You cannot have private abstract methods.
- You will receive an error if you try to provide a method body for an abstract method.
- The method signatures must be identical between the abstract method definition and the method implementation in the derived subclass.
- You will receive an error if you try to provide a Get or Set body with an abstract property.
- You will receive an error and your PeopleCode program will be terminated if you try to call an abstract method or property at runtime, unless caught in a try-catch statement.
- The class that implements an interface must still assign %Super in its constructor.

Interfaces

The concept of an interface class is purely a syntactic assist when you are defining an application class which is totally composed of abstract methods and properties. In this case by simply specify the keyword **interface** you are indicating that all the methods and properties are abstract. The following two definitions are semantically equivalent.

```
class MyInterface
   method MyMethod1() abstract;
method MyMethod2() Returns string abstract;
   property string myproperty1 abstract readonly;
   property number myproperty2 abstract;
property number myproperty3 abstract;
end-class;

Interface MyInterface
   method MyMethod1();
method MyMethod2() Returns string;
   property string myproperty1 readonly;
   property number myproperty2;
property number myproperty3;
end-class;
```

Copyright © 1988-2007, Oracle. All rights reserved.

When you provide an implementation for an Interface you can also use the keyword **Implements** instead of **Extends.** While this is not mandatory, specifying **Implements** describes more accurately what your application class is doing.

If your class implements an interface, you don't need to create a super object interface since the system does that automatically for you.

In addition if your constructor takes no parameters and simply exists to assign to the created instance to %super (that is, there is only one statement such as `%Super = create mySuper();` then you don't need to use a constructor at all.

Constructors

The constructor for a class is the public method with the same name as the (short name of the) class. The statements contained in this method (if any) provide the initialization of the class.

This constructor is *always* executed when an object of the class is instantiated.

Note. Before a constructor runs, the instance variables for the class are set by default based on their declared types.

Instantiate new objects of an application class by using the Create function. This function takes the name of the class, and any parameters needed for the constructor method.

If the short class name is unambiguous, that is, only one class by that short name has been imported, you can use just the short class name.

The following code instantiates an object of the class Fruit:

```
&Fref = Create Fruit();
```

If there's a possibility that the name is ambiguous, you can use the full class name to instantiate the object.

The following code instantiates an object of the Invoice class:

```
&InvObj = Create pt:examples:Invoice();
```

If the Create function is used in the context of further object expressions, that is, with a reference to members by a dot operation, the function call must be enclosed in parentheses. This is to emphasize that the creation happens before the member reference. For example:

```
&InvCust = (Create Invoice()).Cust();
```

If necessary, the constructor is supplied its parameters by the call-list after the class name in the create statement. For example:

```
&Example = create Example(&ConstructorParameter1, 2, "Third");
```

A class that does not extend some other class does not need any constructor.

A class that extends another class must have a constructor, and in the constructor, it must initialize its super class. This restriction is relaxed if the super class constructor takes no parameters, and all the constructor does is assign it to %Super. In this case, the runtime creates the super object and runs its constructor automatically. The benefit of this is two-fold: at design time you do not need to provide a constructor since the runtime automatically does the equivalent of `%Super = Create MySuperObject();` Also, at runtime, you may notice a performance increase since the super class object creation and construction are done without any PeopleCode. This latter benefit can be compounded in the case where there are multiple levels of inheritance.

To the general case, to initialize a superobject, an instance of the superclass is assigned to the keyword %Super in the constructor for the subclass. This assignment is allowed only in the constructor for the subclass.

 Copyright © 1988-2007, Oracle. All rights reserved.

The following example shows this type of assignment:

```
class Example extends ExampleBase
   method Example();
   ...
end-class;

Global string &CurrentBaseString;

method Example
   %Super = create ExampleBase();
   &BaseString = &CurrentBaseString;
   &SlashString = &BaseString;
   &ImportantDate = Date(19970322);
end-method
```

If your subclass' constructor only exists to create the superclass object and assign that to your %Super, you can dispense with providing a constructor completely as the following example illustrates.

```
class A
   ...
end-class;

Class B extends A
...
end-class;

Class C extends B
...
end-class;

...
Local C &c = Create C();

...
```

Classes A, B and C have no constructors and the one call to `Create C` creates objects of class C, B and A and run their constructors. PeopleSoft recommends that unless your constructors take parameters and need to do more than be assigned to their %Super, that you do not provide constructors since that simplifies design time as well as may improve runtime performance significantly.

The above example is semantically equivalent to the following, except that it may run much faster, since it does not have to run PeopleCode at each step of the construction process.

```
class A
   ...
end-class;

Class B extends A
   Method B();
...
end-class;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
method B
%Super = Create A();
end-method;

Class C extends B
   Method C();
...
end-class;

method C
%Super = Create B();
end-method;
```

Note. Application classes don't have destructors, that is, a method called just before an object is destroyed. The PeopleCode runtime environment generally cleans up any held resources, so they're not as necessary as they are in other languages.

See Also

Chapter 6, "Application Classes," Superclass Reference, page 178

External Function Declarations

External function declarations are allowed in application classes, in the global and component variable declarations, after the class declaration (after the end-class statement), and before the method definitions. For example:

```
import SP:Record;

class Person extends SP:Record
   method Person(&pid As string);

   method getPerson() Returns Record;
   method getUserName() Returns string;

private
   instance Record &recPerson;

end-class;

Declare Function getUserName PeopleCode FUNCLIB_PP.STRING_FUNCTIONS FieldFormula;

method Person
   /+ &pid as String +/

   %Super = create SP:Record(Record.SPB_PERSON_TBL);
   &recPerson = %Super.getRecord();
   &recPerson.PERSON_ID.Value = &pid;

end-method;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
method getPerson
   /+ Returns Record +/

   Return &recPerson;

end-method;

method getUserName
   /+ Returns String +/
   Local string &formattedName;

   &formattedName = getUserName(&recPerson.PERSON_ID);
   Return &formattedName;
end-method;
```

When application class properties and instance variables are used as the argument to functions or methods, they are always passed by value, never by reference. This means that the called function or method cannot change the value of the property or instance variable. If you want a function call to change a property or instance variable, use a local variable in the call and then assign the property or instance variable after the call.

Suppose you have the following class and you want to call Meth2 in the body of Meth1, passing it MyProp:

```
class MyClass
   property number MyProp;
   method Meth1();
   method Meth2(&i as number out);
end-class;
```

The following PeopleCode *fails* with a compile-time error:

```
method Meth1
   Meth2(%This.MyProp);   /* error - field or temp name required. */
   Meth2(&MyProp);   /* error - field or temp name required. */
end-method;
```

The following PeopleCode is valid:

```
method Meth1
/* Assignment needed if &i is input as well as output in Meth2. */
   local number &Var = %This.MyProp;
   Meth2(&Var);
   %This.MyProp = &Var;
end-method;
```

Note. You call a Java program from an Application Class the same way you do using any other PeopleCode program, that is, by using one of the existing Java class built-in functions.

See Chapter 19, “Java Class,” From Java to PeopleCode, page 620.

Copyright © 1988-2007, Oracle. All rights reserved.

Naming Standards

This section describes the naming standards for:

- Packages.
- Classes.
- Methods.
- Properties.

Naming Packages

The naming standard for application packages is in the following format.

```
CC_XXX[_YYY]
```

Where CC is the company name (for example, PS is PeopleSoft), XXX is the product line, and YYY is the product code.

Examples are PS_CRM and PS_CRM_RB.

Naming Classes

Use a constructor to be the same name as the short name for the class. PeopleSoft recommends that you do not name classes to be GetXxx.

Naming Methods

Make method names simple verb or verb object words, spelled out, first letter of each word capitalized. Examples: GetField, Save, SetDefault, WriteRecord

Naming Properties

Make property names nouns, likewise spelled out, first letter of each word capitalized. Examples: Dimension, HTTPMethod, PathInfo, SubscriptionProcessId.

For Boolean read-only properties, sometimes IsXxx is appropriate. Examples: IsOpen, IsNewFieldID, IsChanged, IsDeleted

State Boolean properties positively, that is, IsChanged, as opposed to negatively, IsUnChanged. This avoids double negatives when testing (If Not &MyField.IsUnChanged Then. . . .)

Import Declarations

Before you can use a class name in a PeopleCode program (either a class definition program or an ordinary PeopleCode program), you must first import it using an import statement at the beginning of your program.

An import statement names either all the classes in a package or one particular application class.

If you're importing all the classes in a package, the syntax for the import statement ends with the following:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
:*
```

This makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

If you're importing one particular application class, the syntax of the import statement ends with the short name of the class. Only that class is made available.

For example, the following imports all the classes of the package named Fruit:

```
Import Fruit:*;
```

The following imports only the application classes Banana and Kiwi. If the package named Fruit contains other classes, those are not made available to the PeopleCode program:

```
Import Fruit:Banana;
Import Fruit:Kiwi;
```

If you had a subpackage in Fruit named Drinks, you could access all the classes in that subpackage like this:

```
Import Fruit:Drinks:*;
```

A PeopleCode program (either a separate program or one contained within a class definition) can use only application class names that have been imported. However, a class definition can use its own name without having to import it.

It's possible to import two packages which contain application classes with the same short name. For example, suppose my package Fruit contained classes Apple and Banana. Suppose a second package, Drinks, contained Apple and Cherry.

```
Import Fruit:*;
Import Drinks:*;
```

You could refer to the Banana and Cherry classes using just the short name of the class. However, you couldn't refer to the Apple class by the short name. You must always use the full name reference (that is, the name of the package and the name of the class) when referring to Apple.

```
Global Banana &MyBanana;
Local Cherry &CherryTree;
Component Fruit:Apple &GreenApple;
```

The following line produces an error, because the program doesn't know which Apple class you're referring to:

```
Component Apple &GreenApple;
```

Note. Single class imports (such as, Import Fruit:Banana) are checked for existence when you save your PeopleCode.

Self-Reference

A method can refer to the current object using the %This system variable.

Due to subtyping, %This is an object of either the method's class or a subclass of the method's class.

In PeopleCode, the methods and properties of the class *cannot* be used without an object qualification.

In the following code example, the line in bold indicates an error. You can't call the class this way: you must use %This.

```
class FactorialClass
   method factorial(&I as number) returns number;
end-class;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
method factorial
   if &I <= 1 then
      return 1;
   end-if;

   return &I * factorial(&I - 1); /* error - factorial undefined */

   return &I * %This.factorial(&I - 1); /* okay */
end-method;
```

One of the implications of this is that when you're writing a method, there isn't any way to guarantee you'll access your own class' version of a public method (rather then a subclass') as it might be overridden by a subclass. The only way to make sure that you can access your own method is to make it private.

If you have some action that you also want to make public, and you want guaranteed access to it, you can make the action a private method, then define a public method for it that calls the private one.

In the following example, factorial is the public method that can be used by other classes. myFactorial is the private action.

```
class FactorialClass
   method factorial(&I as number) returns number;
private
   method myFactorial(&I as number) returns number;
end-class;

method factorial
   return myFactorial(&I);
end-method;

method myFactorial
   if &I <= 1 then
      return 1;
   end-if;

   return &I * %This.myFactorial(&I - 1);

end-method;
```

If you declare an instance variable as private you can still access it as a private property in another instance of the same class. For example, given the following declaration:

```
class Example
  private
     instance number &Num;
end-class;
```

A method of Example could reference another Example instance's &Num instance variable as follows:

```
&X = &SomeOtherExample.Num;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Using %This with Constructors

%This when used in application class constructor code refers to the object currently being constructed. It does not refer to the dynamic this pointer of some object under construction. Given these two application classes, the PeopleCode `%This.Test()` in the Base constructor *always* refers to the Test method in the Base class.

```
class Base
   method Base();
   method Test();
   method CallTest();
   property string sTestResult;
end-class;

method Base
   %This.Test();
end-method;

method Test
   &sTestResult = "BASE_METHOD_CALLED";
end-method;

method CallTest
   %This.Test();
end-method;

import PACKAGE:Base;

class Extend extends PACKAGE:Base;
   method Extend();

   method Test();
end-class;

method Extend
   %Super = create PACKAGE:Base();
end-method;

method Test
   /+ Extends/implements PACKAGE:Base.Test +/
   %This.sTestResult = "EXTENSION_METHOD_CALLED";
end-method;
```

Even though the Extend class method provides its own Test method which overrides Base's Test method, in PeopleCode `create Extend()` which ultimately runs Base's constructor, the `%This.Test()` call in Base's constructor still references Base's Test method, not Extend's Test method, because %This always refers to the object under construction

Copyright © 1988-2007, Oracle. All rights reserved.

Properties and Constants

This section discusses the following ways to use properties and constants with application classes:

- Differentiating between properties and methods.
- Overriding properties.
- Using the Get and Set keywords.
- Using read-only properties.
- Using methods and properties for collections.
- Declaring constants.

Differentiating Between Properties and Methods

A method is a procedure or routine, associated with one or more classes, that acts on an object.

A property is an attribute of an object.

PeopleSoft recommends that you use read-write or read-only properties instead of creating methods named GetXxx and SetXxx. For example, instead of creating a method GetName, that takes no parameter, just to return the name of an object, create a read-only property Name.

If your properties simply set or return the value of an instance variable, it is far more efficient to declare the property that way than have a get and set method. The overhead of having a get and set method means that each time the property is reference some PeopleCode has to be run.

Application classes can have read-only properties, but no indexed properties.

Considerations Creating Public Properties

There are two implementations of (public) properties:

- as instance variables.
- as get-set methods.

From outside the class, you cannot tell the difference between them (for example, both are dynamically bound). However, if you just want to expose an instance variable to users of the class, you can declare a property of that name and type, which declares the instance variable too:

```
class xxx
     property string StringProp;
end-class
```

If you want the property to be read-only outside the class, use the following code:

```
class xxx
     property string StringProp readonly;
end-class
```

Both of these code examples declare an instance variable called &StringProp and provide its value for getting and change its value for setting (if not readonly). You can think of this as shorthand for the usual implementation of get-set methods.

If you want to implement the same with code (perhaps deriving the property from other instance variables or other data), use the following sort of PeopleCode program:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
class xxx
   property string StringProp get set;
end-class;

get StringProp
   return "string"; /* Get the value from somewhere. */
end-get;

set StringProp
   /* Do something with &NewValue. */
end-set;
```

To specify a read-only property, omit the set portions of the code example.

Overriding Properties

A property of a superclass can be overridden by a subclass by providing a different implementation.

The properties of a class are implemented by the keywords Get and Set, or by access to an otherwise private instance variable.

To access the property, you do not have to specify Get or Set: the context tells the processor which task you're doing. To create read-only properties, specify only a Get method, with no Set method, or for instance variable properties, specify Readonly.

For an instance variable property, there are two different ways for method code in the class itself to refer to the property:

- %This.*PropertyName*
- *&PropertyName*

The first way is dynamically bound, as it uses whatever (possibly overridden by a subclass) implementation of *PropertyName* that the current object provides.

The second way always accesses the private instance variable of the class that the method is declared in.

The following example returns strings, by converting the property. It also demonstrates overriding.

```
class A
   property number Anum;
   property string Astring get;
   property string Astring2 get;
end-class;

/* The following is a dynamic reference to the Anum property of the current⇒
 object, which might be a subclass of A, so this might not access the A⇒
 class&rsquo; implementation of Anum as an instance variable. */
   Get Astring
   return String(%This.Anum);
end-get;

/* The following is a static reference to our &Anum instance variable. Since it⇒
 does not use %This, it will not be overridden even if the current object is a⇒
 subclass of A. */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Get Astring2
   return String(&Anum);
end-get;

/* The following overrides the Anum property of our superclass,
in this case by providing another instance variable implementation.
This gives us our own &Anum instance variable. */
class B extends A
   property number Anum;
end-class;
. . .
/* Set the Anum property in B. Because of the above definitions, this sets
the &Anum instance variable in the B object, but not the &Anum instance
variable in its superobject. In the absence of other setting, the latter will
be defaulted to 0. */

local B &B = Create B();
&B.Anum = 2;

&MyValue = &B.Astring; /* &MyValue is "2" */
&MyValue = &B.Astring2; /* &MyValue is now "0" */
```

Using Get and Set with Properties

PeopleSoft recommends against using read-only properties that are public and mutable in application classes (similar to public instance variables) because this allows external code to read and change the internal state of the object without any controls. Instead, modify the property declarations with the Get and Set keywords.

There is a slight performance advantage, and you write less code, if you use the Get and Set keywords over using get and set methods.

If you define a property with the Get and Set keywords, you also must define a get or set block in the same part of the code that contains the method bodies.

Using Read-Only Properties

PeopleCode read-only properties are functionally similar to static variables, as long as the class that defines the properties initializes them and does not change them. The difference is that you need an instance of the class to access them.

For example, consider a Category object that has a CategoryType member that can have one of the following values:

- ‘R’ for Regular
- ‘B’ for Browse
- ‘N’ for News

You can define three read-only properties corresponding to each of these values, REGULAR, BROWSE, and NEWS, then initialize them to the appropriate values in the constructor of the class.

When you write code to evaluate the CategoryType of the Category object, perform a test like this:

```
If (&category.getCategoryType() = &constants.NEWS) then...
```

 Copyright © 1988-2007, Oracle. All rights reserved.

PeopleSoft recommends referring to the read-only properties, rather than their literal values, for the following reasons:

- If you spell a literal value incorrectly, the PeopleCode editor does not catch it; if you spell a read-only property wrong, it does.
- If you ever need to change the value of a read-only property, you only have to do it in one place.
- It makes the code more maintainable and reusable (because NEWS is more intelligible than 'N').

Using Methods and Properties for Collections

Application classes can contain collections. Collections of objects generally have at least the following methods and properties.

- First()
- Item(*Index*)
- Next()
- Count (as a property)

All numeric indexes in PeopleCode are one-origin, that is, start all indexes at 1, not 0.

Declaring Constants

PeopleSoft recommends that if constants are inextricably linked to a class, you should define them within that class. If the constants are more general, potentially used by multiple classes, consolidate the constants into a constants class at application-level or package-level.

Message catalog set and message numbers are good candidates to centralize into a single constants class. It can improve the readability of code and make it easier to maintain these values.

For example, you can define a constant MSG_SET_NBR, and then define constant values for each message that explain a little about what the message represents (like REQUIRED_VALUE_MISSING, UNIMPLEMENTED_METHOD, or INSUFFICIENT_PRIVILEGES).

Using Variables in Application Classes

This section discusses how to use the following variables in application classes:

- Instance variables.
- Global variables.
- Overriding variables and properties.

Placement of Variable Declarations

Variables are declared after the class definition (the end-class statement) before anything else. After you have declared a variable, you can use it anywhere in your program.

```
class Example
   method Example_M1(&par As string);
end-class;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Global boolean &b;

method Example_M1
   /+ &par as String +/
   &b = False;
end-method;
```

Declaring Private Instance Variables

A private instance variable is private to the class, not just to the object instance. For example, consider a linked-list class where one instance needs to update the pointer in another instance. Another example is the following class declaration:

```
class Example private instance number &Num; end-class;
```

A method of Example could reference another instance of the Example &Num instance variable as:

```
&X = &SomeOtherExample.Num;
```

Avoid making every instance-scoped variable a public property. You should consider each variable individually and decide if it belongs in the public interface or not. If it does, decide if the variable warrants get or set modifiers and therefore should be a public property. If the variable only serves internal purposes to the class, it should be declared as a private instance variable.

Global Variables

PeopleSoft recommends avoiding global variables in application classes unless there is a situation where no other code works. Generally, an application class should not know about anything scoped outside of its instance. There are exceptions, since not all application classes should be standalone. For example, an application class that interacts with the Component Buffer must know about the Component Buffer objects.

Overriding Variables and Properties

PeopleSoft recommends against overriding of variables and of properties.

Since instance variables are private in scope, you cannot access them in subclasses, but you can redeclare them in subclasses. The same is true of public properties; you can even explicitly access the overridden or overriding property. However, overriding is not recommended.

Superclass Reference

A method can refer to a member of its superclass by using the %Super system variable. This construction is needed only to access superclass members that are hidden by overriding members in the current class.

In the following example, the classes that extend the general BuildingAsset class each have their own method DisasterPrep, that does the specific processing, then calls the superclass (generic) version of DisasterPrep.

```
/* generic building class */
class BuildingAsset
   method Acquire();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   method DisasterPrep();
end-class;

method Acquire
   %This.DisasterPrep();
end-method;

method DisasterPrep
   PrepareForFire();
end-method;

/* building in Vancouver */
class VancouverBuilding extends BuildingAsset
   method DisasterPrep();
end-class;

method DisasterPrep
   PrepareForEarthquake();
   %Super.DisasterPrep(); /* call superclass method */
end-method;

/* building in Edmonton */
class EdmontonBuilding extends BuildingAsset
   method DisasterPrep();
end-class;

method DisasterPrep
   PrepareForFreezing();
   %Super.DisasterPrep(); /* call superclass method */
end-method;

local BuildingAsset &Building = Create VancouverBuilding();

&Building.Acquire(); /* calls PrepareForEarthquake then PrepareForFire */

&Building = Create EdmontonBuilding();

&Building.Acquire(); /* calls PrepareForFreezing then PrepareForFire */
```

Downcasting

Downcasting is the process of determining if an object is of a particular subclass type. If the object has that subtype (either the object is of that subtype, or is a subtype of it), a reference to the subobject is returned, otherwise Null is returned. In either case, the type of the resulting value is compatible with the named subclass type.

Copyright © 1988-2007, Oracle. All rights reserved.

This downcast inquiry into the actual type of an object can be used when the object has been passed to some general facility, then passed back to more specific code. For example, the general facility might be a dispatch mechanism, or a new storage structure such as a binary balanced tree. These general facilities won't know about specific types, but it might be necessary for the specific program using the general facility to recover the actual type of an object that is returned to it by the general facility.

A downcast should *not* be used when you could add members to a common base class to provide the same functionality. In other words, if you find yourself using the downcast to test whether the object is one or another of several related types, you should add members to the common base class to provide the operations or properties that you are trying to access, then provide separate implementations of those actions or properties in each specific class.

If the downcast operator is to be used in the context of further object expressions (such as references to other members using dot notation) then the downcast must be enclosed in parentheses to emphasize that the downcast happens before the member reference.

```
class Fruit
   property number FruitCount;
end-class;

class Banana extends Fruit
   property number BananaCount;
end-class;
```

In the following code example, first the variables are assigned. All the variable assignments are legal, for Banana is a subtype of Fruit.

```
local Banana &MyBanana = Create Banana();
local Fruit &MyFruit = &MyBanana;  /* okay, Banana is a subtype of Fruit */
local number &Num = &MyBanana.BananaCount;
```

The next two lines of code don't produce an error, as &MyFruit is currently a Banana at runtime.

```
&MyBanana = &MyFruit as Banana;
&Num = (&MyFruit as Banana).BananaCount; /* same as &MyBanana.BananaCount */
```

In the next lines of code, we're reassigning &MyFruit to be of type Fruit, that is, of the superclass Fruit. When you try to reassign &MyFruit as Banana, &MyBanana is assigned as Null, because &MyFruit is no longer of type Banana.

```
&MyFruit = Create Fruit();

&MyBanana = &MyFruit as Banana; /* assigns Null - &MyFruit isn't a Banana */
```

Exception Handling

Use the Exception Class to do exception handling in your PeopleCode. This class provides a try-catch and throw mechanism so you don't need to check after each operation for errors, but can instead declare by the structure of the try-catch statements when you are interested in handling exceptions, and how you want to handle them.

PeopleSoft recommends that when applicable, application class methods should throw exceptions instead of communicating back to the calling code with return values. Exceptions are instances or subclasses of the PeopleCode Exception class. You can create exceptions by:

 Copyright © 1988-2007, Oracle. All rights reserved.

- Creating an Exception base class that encapsulates the built-in function call and handles its function parameters consistently. This is the preferred way.
- Calling the built-in function CreateException.

Since a method can throw exceptions for several different reasons, PeopleSoft recommends that you:

- create subclasses of Exception often.
- throw strongly-typed exceptions.

This allows the calling code to make decisions based on the exception type.

For example, a method that retrieves customer data from a database might throw an exception named SQLException if a database error occurs, or an exception named PrivilegeException if the current user doesn't have the appropriate permissions to perform the operation. There could be different errors causing either of these exception types to be thrown in this method; if necessary, you can write the calling code to examine the specific message numbers and react accordingly.

PeopleSoft recommends that you always catch an exception; if it is not caught, it causes a runtime error. Use a try-catch block.

Not catching an exception causes a runtime error and can cause PeopleTools to display a dialog box with the message catalog entry and information about the technical context of the exception. The reasons for avoiding this by using try-catch blocks are:

- Application code often continues some form of processing even in the event of an exception; you should not assume that the script stops all processing if an exception occurs.
- The error message that PeopleTools displays might not be appropriate for end-users because of its technical information.

See Also

Chapter 14, "Exception Class," page 481

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateException

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Throw

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Try

Commenting and Documenting Application Classes

This section contains guidelines for writing method header comments, class header comments, and tags inside the comments. It discusses:

- Comments and documentation
- Method header comments
- Class header comments

Copyright © 1988-2007, Oracle. All rights reserved.

Understanding Comments and Documentation

Documentation is important in all code, but even more so in application class code, which aims at, and incurs certain overhead in the name of, higher levels of reuse.

There are two types of inline comments:

- Internal comments, which start with /* and end with */.
- Comments which can potentially be used to generate API documentation, which start with /* and end with */. You can write method header and class header comments for application classes, and you can use the comment tags, with this type of comment.

Warning! In application classes you will see the use of /+ +/ style comments. *Do not use these in your PeopleCode.* These annotations are generated by the compiler. If you use them, they will be removed the next time you validate, compile, or save your PeopleCode. They are used to provide signature information on application class methods and properties, and are regenerated each time the compiler compiles your application class PeopleCode. Instead, use the standard commenting mechanisms listed above.

PeopleSoft recommends keeping a running line of asterisks along one side of an extensive comment to help the readability of the code by making the entire comment stand out from the surrounding code.

Method Header Comments

The following is an example of documentation conventions for method header comments, which are also suitable for PeopleCode. It contains comment tags, which are explained below the example.

```
/**
  * Description of what method does
  *
  * @param a optional description of param a
  * @param b optional descriptions of param b
  * @exception PrivilegeException thrown if etc, etc
  * @exception SQLException thrown if etc., etc.
  * @return String  -  description of potential values
  */
Method myMethod
   ...
End-method;
```

Method Comment Tags

The following are the suggested tags for commenting methods.

Tag	Description
@param *N*	Specify a description for each method parameter. The name of the parameter is specified after the keyword @param.

Copyright © 1988-2007, Oracle. All rights reserved.

Tag	Description
@exception *name*	Specify a description of each exception class. The name of the exception is specified after the keyword @exception. For example, the descriptions can explain what causes certain exceptions to be thrown.
@return *Type*	Specify a description of the return value. The type of data that is returned is specified after the keyword @return. For example, explain the potential values a method may return, whether it returns null in some circumstances, and so on.

Not all methods require documentation. For example, some simple set or get methods have method signatures that are descriptive enough. In other cases, it may suffice to include a brief sentence in the header explaining what the method does.

As with methods, PeopleCode get or set blocks should also carry method header comments.

Class Header Comments

Header comments for the class should provide descriptive information about the class, and may optionally include additional tags, indicating version and author:

```
/**
* class description
*
* @version 1.0
* @author amachado
*/
```

Class Header Comment Tags

The following are the suggested tags for commenting class headers.

Tag	Description
@version *number*	Specify a version number for a class.
@author *name*	Specify an author for a class.

Designing Base Classes

This section discusses some ways you can design application classes.

- Base data classes
- Abstract base classes
- Generic base classes
- Utility classes

Copyright © 1988-2007, Oracle. All rights reserved.

Base Classes

Base classes contain a representation of data elements, such as documents, categories, users, and miscellaneous security-related entities.

To design the document data element to be integration-friendly, PeopleSoft recommends you start with a relatively basic base class. Then use more specialized types of documents to create a subclass from that base class, adding the necessary specialized attributes.

Even if you think of ways to use data to drive the business rules, PeopleSoft recommends that you do not place them into your data objects. Replicating business rules for every instance of a data element can be detrimental to performance.

Abstract Base Classes

Abstract base classes contain reusable logic for PeopleCode classes; you do not instantiate them directly. Instead, you instantiate classes that are subclasses of the abstract base class. Abstract base classes use placeholder methods for specialized logic, and its subclasses override those methods to provide the specialized logic.

An example of a candidate for an abstract base class is a class that is used for administering a data entity, such as an invoice. Such a class has a Save method. A Save method can use normal methods to call generic validation routines, talk to the database, and handle errors; it can use the placeholder methods for specialized logic, such as mapping a given data class to a particular database table.

PeopleSoft recommends that you program an active validation check into your placeholder method that causes a runtime error, such as throwing an UnimplementedMethodException, if an abstract base class placeholder method hasn't been overridden.

Generic Base Classes

Generic base classes contain reusable logic for PeopleCode classes; you write them to be self-contained, fully implemented, and insatiable. This differs from abstract base classes, which are not instantiated directly and need subclasses that override methods to provide specialized logic.

Some examples of classes that are good candidates for creating generic base classes are:

- Service classes, which typically extend a class that provides infrastructure services, such as configurable application logging.
- Data classes, which typically extend a class that enables you to plug in adapters to support sorting their instances with custom sort algorithms.

Even when a class does not seem generic because it only has a couple of subclasses, you can still define it in your design as a generic base class rather than cloning common code across different classes. The main reason is maintainability: it's easier to maintain a single class than trying to keep track of the same code scattered across your database.

Utility Classes

Some reusable code is the kind you typically don't want to inherit, but you just want to use here and there. Utility classes can serve this purpose.

String parsing, data type conversion, and formatting routines are all good candidates for utility classes. PeopleSoft recommends that anytime you code something that's more than just a line or two, and that seems to be reusable, write it as a utility class.

 Copyright © 1988-2007, Oracle. All rights reserved.

Declaration of Application Classes

This section discusses instructions for declaring application classes, such as:

- Declaring application classes.
- Importing class names.
- Referencing Superclasses.
- Declaring private methods.

Declaring Application Classes

Application object references are declared in your PeopleCode programs by either the short name of the class (if the short name is unambiguous amongst all the imported classes), or by the full name of the class. Like all uses of class names, to use the name in a variable declaration, you must have imported the class. Application objects can be of Local, Global, or Component scope.

Every application class is a definition of a new data type in PeopleCode, like the record, rowset, or field data types. The application class includes both the definition of its external interface, its data (properties and instance variables), and its actions (methods), just like the PeopleSoft delivered classes.

Importing Class Names

When you import a class name, fully-qualifying it is optional. The syntax is:

```
<package>:<subpackage>:<...>:<classname or wildcard>
```

PeopleSoft recommends that you individually import each class when you import multiple classes from the same package, instead of using wildcards.

Referencing Superclasses

PeopleSoft recommends that you do not set %Super to a new instance of the current class you are defining because this would start an infinite loop.

```
/* Do not do this! */
%Super = %This;
```

Declaring Private Methods

PeopleSoft recommends that you use private methods for succinct operations, even if they aren't reused, to improve the readability and maintainability of the code. You can also do this by using many inline comments.

Scope of an Application Class

Application classes can be instantiated only from PeopleCode. These classes can be used anywhere you have PeopleCode, that is, in message notifcation PeopleCode, Component Interface PeopleCode, record field PeopleCode, and so on.

Copyright © 1988-2007, Oracle. All rights reserved.

Application Classes Built-in Functions and Language Constructs

In this section, we discuss the Application classes built-in functions and language constructs, in alphabetical order.

Class

Syntax

```
Class classname [{Extends | Implements} Classname]
   [Method_declarations]
   [Property_declarations]
[Protected
   [Method_declarations]
   [Instance_declarations]
   [Constant_declaration]]
[Private
   [Method_declarations]
   [Instance_declarations]
   [Constant_declaration]]
End-Class
```

Where Method_declarations are of the form:

```
Method methodname([MethodParam1 [, MethodParam2. . .] [Returns Datatype]] [abstract])
```

Where Property_declarations are of the form:

```
Property DataType PropertyName {{[Get] | [Set]} | [abstract] | [readonly]}
```

Where Instance_declarations are of the form:

```
Instance DataType &Variable1 [, &Variable2. . .]
```

Where Constant_declarations are of the form:

```
Constant &Constant = {Number  | String | True | False | Null }
```

Description

Use the Class language construct to build application classes in an application package. All classes within a package must be uniquely named, however, classes contained in different packages do not have to be named uniquely.

External function declarations are allowed in application classes, in the global and component variable declarations, after the class declaration (after the end-class statement), and before the method definitions.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
classname	Specify the name of the class that you are creating. All classes within a package must be uniquely named, however, classes contained in different packages do not have to be named uniquely.
Extends | **Implements** *classname*	For a regular class, specify the name of the class that this class extends. If the class is an interface type of class, specify the name of the interface that this class implements. You must import the class to extend it.
Method_declarations	Specify the methods (and their signature) as used in this class.
Property_declarations	Specify the properties, and their usage, in this class.
Protected	Use this keyword to declare any methods, constants, or instance variables as protected to the class, that is, are only visible to the declaring class and subclasses.
Private	Use this keyword to declare any methods, constants, or instance variables as private to the class, that is, they can't be accessed by any other classes.
Instance_declarations	Specify any variables that should be in any instance (object) of the class.
Constant_declarations	Specify any constants that should be used with this class.

Returns

None.

Example

```
class Example extends ExampleBase
   method NumToStr(&Num as number) returns string;
   method AppendSlash();
   property number SlashCount get;
   property number ImportantDayOfWeek get set;
   property string SlashString readonly;
   property date ImportantDate;
private
   method NextDayOfWeek(&DoW as number) returns date;
   constant &Sunday = 1;
   instance number &BaseString;
end-class;
```

See Also

Chapter 6, "Application Classes," Method, page 190, Chapter 6, "Application Classes," Get, page 187 and Chapter 6, "Application Classes," Set, page 190.

Get

Syntax

Get *PropertyName*
StatementList

Copyright © 1988-2007, Oracle. All rights reserved.

```
End-Get
```

Description

Use the Get language construct when defining properties in an application class that are implemented by methods rather than an instance variable. All properties within an application class must be uniquely named.

Parameters

Parameter	Description
Get *PropertyName*	Specify the name of the property that you're implementing.
StatementList	Returns the value of the property. In other words, this is a method which has no parameters and must return the value of the property.

Returns

Depends on the assignment within *StatementList*

Example

```
Get FruitCount
  Return &MyFruit.Number();
End-Get;
```

See Also

Chapter 6, "Application Classes," Method, page 190 and Chapter 6, "Application Classes," Set, page 190.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Return

Import

Syntax

```
Import PackageName:[PackageName. . .:]{Classname | *}
```

Description

Before you can use a class name, you must first import it using an import declaration statement at the beginning of your program.

An import declaration statement names either all the classes in a package or a single application class.

Parameters

Parameter	Description
Import *PackageName*	Specify the name of the package that you want to import.
*	Import all the classes of the specified package.
Classname	Import only this class of the specified package. You won't have access to any other classes in the specified package, only this one.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

The following gives you access to all the classes in the package Fruit:

```
Import Fruit:*;
```

The following only gives you access to the class Banana in the package Fruit:

```
Import Fruit:Banana;
```

The following gives you access to the class Apple, in the package Drinks, which contains the package Fruit:

```
Import Drinks:Fruit:Apple;
```

See Also

Chapter 6, "Application Classes," Class, page 186

Interface

Syntax

```
Interface classname [{Extends  | Implements} Classname]
   [Method_declarations]
   [Property_declarations]
[Protected
   [Method_declarations]
   [Instance_declarations]
   [Constant_declaration]]
]
End-Interface
```

Where Method_declarations are of the form:

```
Method methodname([MethodParam1 [, MethodParam2. . .] [Returns Datatype]] [abstract])
```

Where Property_declarations are of the form:

```
Property DataType PropertyName {{[Get] | [Set]} | [abstract] | [readonly]}
```

Where Instance_declarations are of the form:

```
Instance DataType &Variable1 [, &Variable2. . .]
```

Where Constant_declarations are of the form:

```
Constant &Constant = {Number  | String | True | False | Null }
```

Description

Use the Interface language construct to create classes of type interface in an application package. An interface class is a purely abstract class.

Parameters

An interface does not have Private methods or properties, but all the other parameters are identical to the Class language construct.

See Chapter 6, "Application Classes," Class, page 186.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 6, “Application Classes,” Class, page 186

Method

Syntax

```
Method MethodName
StatementList
End-Method
```

Description

Use the Method statement to define the methods of your application class.

Parameters

Parameter	Description
Method *methodname*	Specify the name of the method that you are creating.
StatementList	Specify the program of the method, what it does.

Returns

Depends on the method.

Example

```
method NumToStr
   return String(&Num);
end-method;

method AppendSlash
   &SlashString = &SlashString | "/";
end-method;
```

See Also

Chapter 6, “Application Classes,” Get, page 187 and Chapter 6, “Application Classes,” Set, page 190.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” Return

Set

Syntax

```
SetPropertyName
   StatementList
End-Set
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the Set language construct when implementing properties in an application class. All properties within an application class must be uniquely named. The new value for the property is available in the &NewValue parameter of the Set method.

Note. You cannot create a set-only property. You can create only get-set properties.

Parameters

Parameter	Description
Set *PropertyName*	Specify the name of the property that you're implementing.
StatementList	Change the value of the property to be &NewValue.

Returns

None.

Example

```
Set FruitCount
  &Fruitcount = &MyFruit[&NewValue].ActiveRowCount();
End-Set;
```

See Also

Chapter 6, "Application Classes," Get, page 187 and Chapter 6, "Application Classes," Method, page 190.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 7

API Repository

This chapter provides an overview of the PeopleSoft API Repository and discusses:

- Repository properties.
- Bindings collection properties.
- Bindings collection methods.
- Bindings properties.
- Bindings methods.
- Namespaces collection properties.
- Namespaces collection methods.
- Namespaces properties.
- Namespaces methods.
- ClassInfo collection properties.
- ClassInfo collection methods.
- ClassInfo properties.
- MethodInfo collection properties.
- MethodInfo collection methods.
- MethodInfo properties.
- PropertyInfo collection properties.
- PropertyInfo collection methods.
- PropertyInfo properties.
- Summary of repository methods and properties.

Understanding the PeopleSoft API Repository

This section discusses:

- The PeopleSoft API Repository.
- Accessing the repository by using PeopleCode.
- Accessing the repository by using Visual Basic.

Copyright © 1988-2007, Oracle. All rights reserved.

The PeopleSoft API Repository

The PeopleSoft API Repository enables PeopleCode and third-party integrators to discover the internally available classes, methods, and properties that are provided by PeopleSoft for integration. The repository is useful to third-party integrators who integrate in a generic fashion: middleware providers, testing tool providers, and automated documentation providers.

The PeopleSoft API Repository is *not* a necessary interface for integrators who integrate at the business-rule level, such as integration with an expense report, and so on. Those integrators should use component interfaces or business interlinks.

The repository describes available PeopleSoft APIs and provides mechanisms to determine the classes that are available in the API, the properties of each class, the methods of a class (along with the required parameters), and information concerning which group a class belongs to (known as a namespace).

The process of determining information about the API is known as *discovery.* Third-party integrators use information found through discovery to drive generic integration tools.

The repository is divided into namespaces. Each namespace contains a collection of related classes. Example namespaces include "PeopleSoft," "ComponentInterface," "Trees," and "BusinessInterlinks".

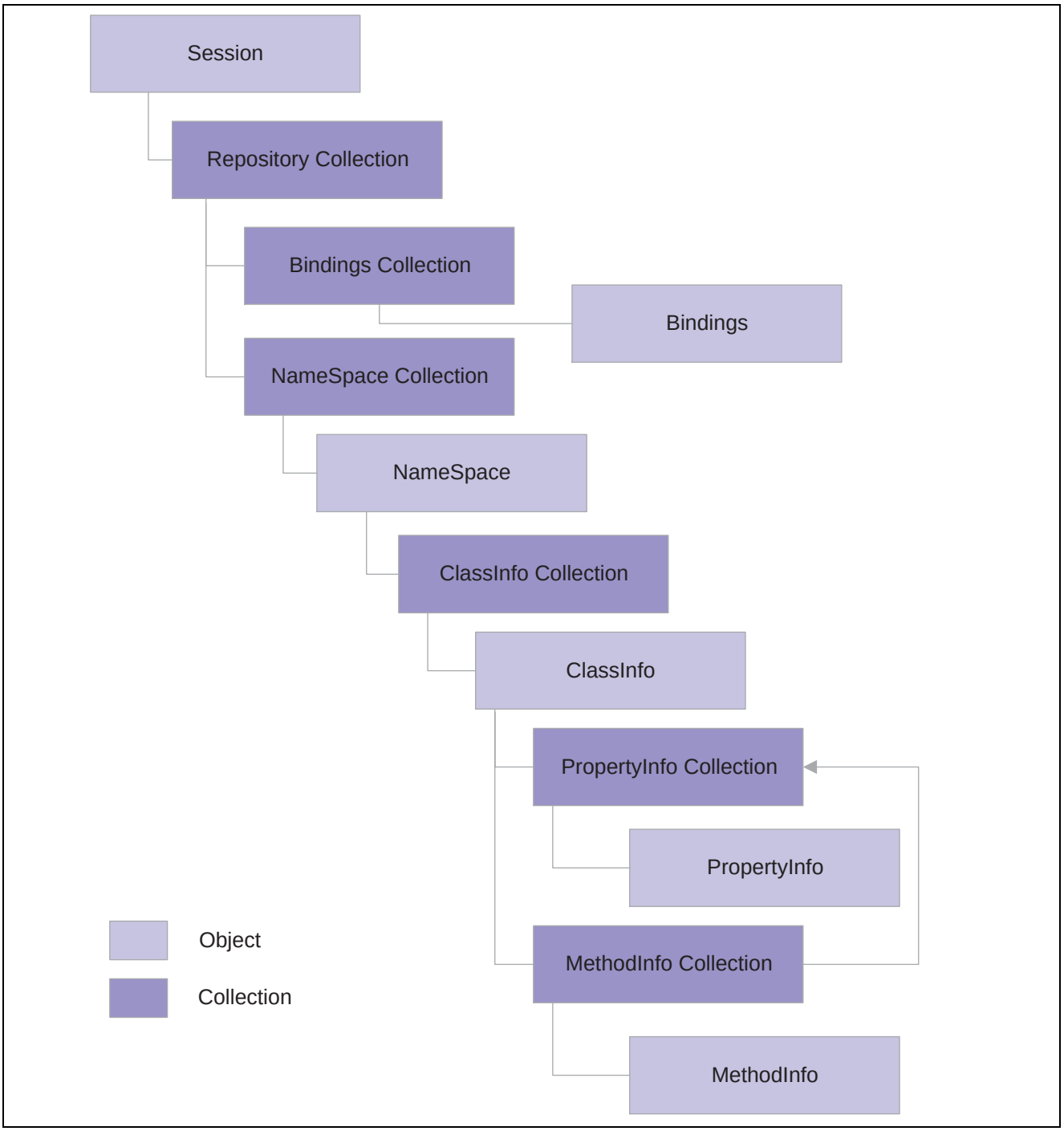
A *class* defines a related set of methods and properties. Using the repository, you can determine the methods and properties that are available and that can be used on any object returned by a call to the PeopleSoft API. An instance of a class is known as an object.

A *property* is a data item of an object that has both a name and type (string, number, Boolean, and so on are examples of types). Some properties are used for inputting data to a class, some are used for getting data from a class, and some are used for both. Whether a property is used for input or output or both is known as usage.

A *method* is a function that you can call on an object. Methods have a name and a return type (string, number, Boolean, and so on). Methods also have a collection of arguments that must be set prior to invoking the method. Methods arguments have identical attributes to properties.

This diagram shows the different types of objects and collections instantiated from the repository:

 Copyright © 1988-2007, Oracle. All rights reserved.



Repository class hierarchy

Example of Accessing the Repository Using PeopleCode

This example gets information for the class ABS_HIST from the Namespace component interface and writes it to the file BC.TXT:

This is the complete code sample, followed by the flat file. The next section presents steps that explain each line.

```
Local ApiObject &MYSESSION;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local ApiObject &MYCI;
Local string &OutTEXT;
Local File &MYFILE;

&MYSESSION = %Session;

&MYFILE = GetFile("CI.txt", "A");

&NAMESPACES = &MYSESSION.Repository.Namespaces;
&NAMESPACE = &NAMESPACES.ItemByName("CompIntfc");

&OutTEXT = "Namespace = " | &NAMESPACE.Name;
&MYFILE.WriteLine(&OutTEXT);

&CLASSES = &NAMESPACE.classes;
&CLASS = &CLASSES.ItemByName("ABS_HIST");

&OutTEXT = "  Class: " | &CLASS.Name;
&MYFILE.WriteLine(&OutTEXT);

&OutTEXT = "     Methods";
&MYFILE.WriteLine(&OutTEXT);

&METHODS = &CLASS.methods;
For &K = 0 To &METHODS.Count - 1
   &METHOD = &METHODS.item(&K);
   &OutTEXT = "             " | &METHOD.name | ":  " | &METHOD.Type

   &MYFILE.WriteLine(&OutTEXT);
   &ARGUMENTS = &METHOD.arguments;
   For &M = 0 To &ARGUMENTS.count - 1
      &ARGUMENT = &ARGUMENTS.item(&M);
      &OutTEXT = "           " | &ARGUMENT.name | ": " | &ARGUMENT.type;
      &MYFILE.WriteLine(&OutTEXT);
   End-For;
End-For;

&OutTEXT = "     Properties";
&MYFILE.WriteLine(&OutTEXT);

&PROPERTIES = &CLASS.properties;
For &I = 0 To &PROPERTIES.count - 1
   &PROPERTY = &PROPERTIES.item(&I);
   &OutTEXT = "              " | &PROPERTY.name | ": " | &PROPERTY.type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
&MYFILE.Close();
```

The previous code produces the following flat file:

```
Namespace = CompIntfc
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
Class: ABS_HIST
   Methods
           Get:  Boolean
           Save:  Boolean
           Cancel:  Boolean
           Find:  ABS_HIST
           GetPropertyByName:  Variant
         Name: String
           SetPropertyByName:  Number
         Name: String
         Value: Variant
           GetPropertyInfoByName:  CompIntfcPropertyInfo
         Name: String
   Properties
            EMPLID: String
            LAST_NAME_SRCH: String
            NAME: String
            ABSENCE_HIST: ABS_HIST_ABSENCE_HISTCollection
            interactiveMode: Boolean
            getHistoryItems: Boolean
            componentName: String
            compIntfcName: String
            stopOnFirstError: Boolean
            propertyInfoCollection: CompIntfcPropertyInfoCollection
            createKeyInfoCollection: CompIntfcPropertyInfoCollection
            getKeyInfoCollection: CompIntfcPropertyInfoCollection
            findKeyInfoCollection: CompIntfcPropertyInfoCollection
```

The PeopleCode Example Explanation

This procedure goes through the PeopleCode example line by line.

To retrieve information from the API Repository:

1. Get a session object.

   Before you can access the PeopleSoft API Repository, you have to get a session object. The session controls access to PeopleSoft, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Open the file.

   As this text will be written to a flat file, the next step is to open the file. If the file is already created, the new text is appended to the end of it. If the file hasn't been created, the GetFile built-in function creates the file.

   ```
   &MYFILE = GetFile("CI.txt", "A");
   ```

3. Get the namespace you want.

Copyright © 1988-2007, Oracle. All rights reserved.

Use the Namespaces property on the repository object to get a collection of available namespaces. We want to discover information about a component interface, so we specify CompIntfc in the ItemByName method to get that namespace. With ItemByName, you must specify a namespace that already exists. You'll receive a runtime error if you specify one that doesn't exist.

```
&NAMESPACES = &MYSESSION.Repository.Namespaces;
&NAMESPACE = &NAMESPACES.ItemByName("CompIntfc");
```

4. Write the text to the file.

   Because all of the information discovered is being written to a file, the next step is to write text to the file. This code writes the string "Namespace", followed by the name of the namespace, to the file.

```
&OutTEXT = "Namespace = " | &NAMESPACE.Name;
&MYFILE.WriteLine(&OutTEXT);
```

5. Get the class that you want and write text to the file.

   Use the Classes property on the Namespace object to get a collection of all the available classes. We want to discover information about the component interface named ABS_HIST, so we specify that using ItemByName. Then we write that information to the file.

```
&CLASSES = &NAMESPACE.classes;
&CLASS = &CLASSES.ItemByName("ABS_HIST");

&OutTEXT = "  Class: " | &CLASS.Name;
&MYFILE.WriteLine(&OutTEXT);
```

6. Get the methods and arguments, and write the information to the file.

   Use the Methods property on the Class object to get a collection of all the available methods. After you get each method and write the information to the file, loop through and find all of the arguments for the method, then write that information to the file.

```
&OutTEXT = "     Methods";
&MYFILE.WriteLine(&OutTEXT);

&METHODS = &CLASS.methods;
For &K = 0 To &METHODS.Count - 1
   &METHOD = &METHODS.item(&K);
   &OutTEXT = "             " | &METHOD.name | ":  " | &METHOD.Type;
   &MYFILE.WriteLine(&OutTEXT);
   &ARGUMENTS = &METHOD.arguments;
   For &M = 0 To &ARGUMENTS.count - 1
      &ARGUMENT = &ARGUMENTS.item(&M);
      &OutTEXT = "           " | &ARGUMENT.name | ": " | &ARGUMENT.type;
      &MYFILE.WriteLine(&OutTEXT);
   End-For;
End-For;
```

7. Get the properties and write the information to the file.

   Use the Properties property on the Class object to get a collection of all the available properties. Write each property, with its type, to the file. At the end of the program, close the file.

```
&OutTEXT = "     Properties";
&MYFILE.WriteLine(&OutTEXT);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&PROPERTIES = &CLASS.properties;
For &I = 0 To &PROPERTIES.count - 1
   &PROPERTY = &PROPERTIES.item(&I);
   &OutTEXT = "              " | &PROPERTY.name | ": " | &PROPERTY.type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
&MYFILE.Close();
```

Example of Accessing the Repository by Using Visual Basic

This example gets information for the class ABS_HIST from the Namespace component interface.

```
Private Sub Command1_Click()
    '************************************************************
    '* TacDemo: Example Repository Usage from Visual Basic
    '*
    '* Copyright (c) 1999 PeopleSoft, Inc.  All rights reserved

    '************************************************************

    ' Declare variables
    Dim oSession As New PeopleSoft_PeopleSoft.Session
    Dim oPSMessages As PSMessageCollection
    Dim oPSMessage As PSMessage

    ' Establish a PeopleSoft Session
    nStatus = oSession.Connect(1, "//PSOFTO070698:9001", "PTDMO", "PTDMO", 0)

    ' Enable error-handler
    On Error GoTo ErrorHandler

    ' Get a Component Interface "shell"
    Dim oNamespaces As NamespaceCollection
    Dim oNamespace As Namespace
    Dim oClasses As ClassInfoCollection
    Dim oClass As ClassInfo
    Dim oMethods As MethodInfoCollection
    Dim oMethod As MethodInfo
    Dim oArguments As PropertyInfoCollection
    Dim oArgument As PropertyInfo
    Dim oProperties As PropertyInfoCollection
    Dim oProperty As PropertyInfo

    Set oNamespaces = oSession.Repository.namespaces
    Set oNamespace = oNamespaces.ItemByName("ComponentInterface")

    Dim outText As String

    outText = "Namespace = " & oNamespace.Name & vbNewLine
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
    Set oClasses = oNamespace.classes
    Set oClass = oClasses.ItemByName("ABS_HIST")

    outText = outText & "   Class: " & oClass.Name & vbNewLine

    outText = outText & "       Methods" & vbNewLine

    Set oMethods = oClass.methods
    For k = 0 To oMethods.Count - 1
        Set oMethod = oMethods.Item(k)        outText = outText & "           "
& oMethod.Name & ":  " & oMethod.Type & vbNewLine
        Set oArguments = oMethod.arguments
        For m = 0 To oArguments.Count - 1
            Set oArgument = oArguments.Item(m)
            outText = outText & "                "
& oArgument.Name & ":  " & oArgument.Type & vbNewLine
        Next
    Next

    outText = outText & "       Properties" & vbNewLine

    Set oProperties = oClass.properties
    For k = 0 To oProperties.Count - 1
        Set oProperty = oProperties.Item(k)
        outText = outText & "            " & oProperty.Name & ":  "
& oProperty.Type & vbNewLine
    Next

 txtResults = outText

' Leave before we encounter the error handler
Exit Sub

ErrorHandler:
    If Err.Number = 1001 Then               ' PeopleSoft Error
        Set oPSMessages = oSession.PSMessages
        If oPSMessages.Count > 0 Then
            For i = 1 To oPSMessages.Count
               Set oPSMessage = oPSMessages.Item(i)
               MsgBox (oPSMessage.Text)
            Next i
            oPSMessages.DeleteAll
        Else
            MsgBox ("PS Api Error. No additional information
available from Session log")
        End If
    Else                                    ' VB Error
        MsgBox ("VB Error: " & Err.Description)
    End If
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
End Sub
```

Repository Properties

This section discusses the Repository properties in alphabetical order.

Bindings

Description

The Bindings property returns a reference to a Bindings collection.

This property is read-only.

Namespaces

Description

The Namespaces property returns a reference to a Namespaces collection.

This property is read-only.

Bindings Collection Properties

This section discusses the Bindings collection properties in alphabetical order.

Count

Description

This property returns the number of Bindings Properties objects in the Bindings collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &BINDINGS.Count;
```

See Also

Chapter 7, “API Repository,” Bindings Properties, page 202

Copyright © 1988-2007, Oracle. All rights reserved.

Bindings Collection Methods

This section discusses the Bindings collection methods in alphabetical order.

Item

Syntax

Item(*number*)

Description

The Item method returns a Bindings object that exists at the number position in the Bindings collection executing the method

Parameters

Parameter	Description
number	Specify the position number in the collection of the Bindings object that you want returned.

Returns

A reference to a Bindings object or NULL.

Example

```
For &N = 0 to &BINDINGS.Count - 1
   &BINDING = &BINDINGS.Item(&N);
   /* do processing */
End-For;
```

Bindings Properties

This section discusses the Bindings properties in alphabetical order.

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Bindings Methods

This section discusses the Bindings methods in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Generate

Syntax

Generate()

Description

This method is a reserved internal function and shouldn't be used at this time.

Namespaces Collection Properties

This section discusses the Namespaces collection properties in alphabetical order.

Count

Description

This property returns the number of Namespaces Properties objects in the Namespaces collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &NameC.Count;
```

See Also

Chapter 7, "API Repository," Namespaces Properties, page 204

Namespaces Collection Methods

This section discusses the Namespaces collection methods in alphabetical order.

Item

Syntax

Item(*number*)

Description

The Item method returns a Namespaces object that exists at the *number* position in the Namespaces collection executing the method.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
number	Specify the position number in the collection of the Namespaces object that you want returned.

Returns

A reference to a Namespaces object or NULL.

Example

```
For &N = 0 to &NAMESPACES.Count - 1
   &NAMESPACE = &NAMESPACES.Item(&N);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*name*)

Description

The ItemByName method returns the item specified by *name. Name* is not case-sensitive.

Parameters

Parameter	Description
name	Specify the name of the Namespaces object that you want returned. This parameter takes a string value.

Returns

A reference to a Namespaces object or NULL.

Example

```
&NAMESPACE = &NAMESPACES.ItemByName("BusinessComponent");
```

Namespaces Properties

This section discusses the Namespaces properties in alphabetical order.

Classes

Description

This property returns a reference to a ClassInfo collection object.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

```
&CLASSC = &NAME.Classes;
```

See Also

Chapter 7, “API Repository,” ClassInfo Collection Properties, page 205

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Namespaces Methods

This section discusses the Namespaces methods in alphabetical order.

CreateObject

Syntax

CreateObject(*classname*)

Description

This method is a reserved internal function and shouldn’t be used at this time.

ClassInfo Collection Properties

This section discusses the ClassInfo collection properties in alphabetical order.

Count

Description

This property returns the number of ClassInfo Properties objects in the ClassInfo collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &InfoC.Count;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 7, "API Repository," ClassInfo Properties, page 207

ClassInfo Collection Methods

This section discusses the ClassInfo collection methods in alphabetical order.

Item

Syntax
Item(*number*)

Description

The Item method returns a ClassInfo object that exists at the *number* position in the ClassInfo collection executing the method.

Parameters

Parameter	Description
number	Specify the position number in the collection of the ClassInfo object that you want returned.

Returns

A reference to a ClassInfo object or NULL.

Example

```
For &N = 0 to &CLASSES.Count - 1
   &CLASS = &CLASSES.Item(&N);
   /* do processing */
End-For;
```

ItemByName

Syntax
ItemByName(*name*)

Description

The ItemByName method returns the item specified by *name*. *Name* is not case-sensitive.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
name	Specify the name of the ClassInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a ClassInfo object or NULL.

Example

```
&CLASS = &CLASSES.ItemByName("ABS_HIST");
```

ClassInfo Properties

This section discusses the ClassInfo properties in alphabetical order.

Documentation

Description

This property doesn't actually return all the documentation for the class, just a brief description of the class as a string.

This property is read-only.

Methods

Description

This property returns a reference to a MethodInfo collection object.

This property is read-only.

See Also

Chapter 7, "API Repository," MethodInfo Collection Methods, page 208

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Properties

Description

This property returns a reference to a PropertyInfo collection object.

This property is read-only.

See Also

Chapter 7, "API Repository," PropertyInfo Collection Methods, page 211

MethodInfo Collection Properties

This section discusses the MethodInfo collection properties in alphabetical order.

Count

Description

This property returns the number of MethodInfo Properties objects in the MethodInfo collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &MethC.Count;
```

See Also

Chapter 7, "API Repository," MethodInfo Properties, page 209

MethodInfo Collection Methods

This section discusses the MethodInfo collection methods in alphabetical order.

Item

Syntax

Item(*number*)

Description

The Item method returns a MethodInfo object that exists at the *number* position in the MethodInfo collection executing the method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
number	Specify the position number in the collection of the MethodInfo object that you want returned.

Returns

A reference to a MethodInfo object or NULL.

Example

```
For &K = 0 To &METHODS.Count - 1
   &METHOD = &METHODS.item(&K);
   &OutTEXT = "             " | &METHOD.name | ":  " | &METHOD.Type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
```

ItemByName

Syntax

ItemByName(*name*)

Description

The ItemByName method returns the item specified by *name*. *Name* is not case-sensitive.

Parameters

Parameter	Description
name	Specify the name of the MethodInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a MethodInfo object or NULL.

Example

```
&METHOD = &METHODS.ItemByName("Save");
```

MethodInfo Properties

This section discusses the MethodInfo properties in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

Arguments

Description

This property returns a reference to a PropertyInfo collection object.

This property is read-only.

See Also

Chapter 7, "API Repository," PropertyInfo Collection Methods, page 211

Documentation

Description

This property doesn't actually return all the documentation for the class, just a brief description of the class, as a string.

This property is read-only.

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Type

Description

This property returns the type of the method. Values include:

- Bool (Boolean).
- Number.
- Float.
- String.
- Variant.
- Blob (binary large object).
- Any API class name.

This property is read-only.

PropertyInfo Collection Properties

This section discusses the PropertyInfo collection properties in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of PropertyInfo Properties objects in the PropertyInfo collection object.

Note. All repository counts begin at zero, not one.

This property is read-only.

Example

```
&COUNT = &PropC.Count;
```

See Also

Chapter 7, "API Repository," PropertyInfo Properties, page 212

PropertyInfo Collection Methods

This section discusses the PropertyInfo collection methods in alphabetical order.

Item

Syntax

Item(*number*)

Description

The Item method returns a PropertyInfo object that exists at the *number* position in the PropertyInfo collection executing the method.

Parameters

Parameter	Description
number	Specify the position number in the collection of the PropertyInfo object that you want returned.

Returns

A reference to a PropertyInfo object or NULL.

Example

```
For &K = 0 To &PROPERTIES.Count - 1
   &PROPERTY = &PROPERTIES.item(&K);
   &OutTEXT = "             " | &PROPERTY.name | ":  " | &PROPERTY.Type;
   &MYFILE.WriteLine(&OutTEXT);
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

ItemByName

Syntax

ItemByName(*name*)

Description

The ItemByName method returns the item specified by *name. Name* is not case-sensitive.

Parameters

Parameter	Description
name	Specify the name of the PropertyInfo object that you want returned. This parameter takes a string value.

Returns

A reference to a PropertyInfo object or NULL.

Example

```
&PROPERTY = &PROPERTIES.ItemByName(&ldquo;GetHistoryItems&rdquo;);
```

PropertyInfo Properties

This section discusses the PropertyInfo properties in alphabetical order.

Documentation

Description

This property doesn't actually return all the documentation for the class, just a brief description of the class, as a string. This property is read-only.

Name

Description

This property returns the name of the object as a string.

This property is read-only.

Type

Description

This property returns the data type. Values are:

- Bool (Boolean).
- Number.

 Copyright © 1988-2007, Oracle. All rights reserved.

- Float.
- String.
- Variant.
- Blob (binary large object).
- Any API class name.

This property is read-only.

Usage

Description

This property returns a number that describes in which direction the specified property (or argument) can be passed. The following table describes the valid values.

Value	Description
0	Can be passed into PeopleSoft API.
1	Can be passed out of PeopleSoft API.
2	Can be passed either into or out of PeopleSoft API.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 8

Array Class

This chapter provides an overview of arrays and discusses how to:

- Create arrays.
- Populate an array.
- Remove items from an array.
- Create empty arrays.
- Create and populate multi-dimensional arrays.
- Use flattening and promotion.
- Declare array objects.
- Understand the scope of an array object.
- Use array class built-in functions.
- Use array class methods
- Use array class properties

Understanding Arrays

An array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an element of the array. When you create an array, you don't have to declare the size of the array at declaration time. Arrays grow and shrink dynamically as you add (or remove) data. The size of an array is limited by the available memory. You can't access past the end of an array, but you can assign outside the existing boundaries, thereby growing the array.

Creating Arrays

Arrays are declared by using the Array type name, optionally followed by "of" and the type of the elements. If the element type is omitted, it is set by default to ANY.

```
Local Array of Number &MYARRAY;
Local Array &ARRAYANY;
```

Arrays can be composed of any valid PeopleCode data type, such as string, record, number, date, and so on.

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleSoft recommends you declare every object you use in PeopleCode. This provides some syntax checking when you save PeopleCode. It's better to find out that you misspelled the name of a method or property at design time, rather than at runtime!

Arrays can be declared as Local, Global, or Component, just like any other PeopleTools object.

Arrays can be created with one or more dimensions. An array with more than one dimension is called an array of arrays. The *dimension* of an array is the number of Array type names in the declaration. This is also called the *depth* of an array. The maximum depth of a PeopleCode array is 15 dimensions.

In the following example, &MYARRAY has three dimensions, and &MYA2 has two dimensions.

```
Local Array of Array of Array of Number &MYARRAY;
Local Array of Array &MYA2;
```

An array must always have a consistent dimension. This means that in a one-dimensional array *none* of the elements can be an array, in a two-dimensional array *all* of the elements must be one-dimensional arrays, and so on.

After you declare an array, use one of the built-in array functions to instantiate it and return an object reference to it. For example, the following creates an array containing one element of type &TEMP, whatever data type &TEMP may be.

```
&MYARRAY = CreateArray(&TEMP);
```

Or you can use the CreateArrayRept function to instantiate an array. The Rept stands for *repeat*. CreateArrayRept creates an array that contains the number of copies you specify of a particular value. The following code creates an array with three copies of the string &MYSTRING. This does *not* create a three-dimensional array, but rather creates an array that's already populated with three elements of data (`Len = 3`), each of which contain the same string (&MYSTRING).

```
&MYARRAY = CreateArrayRept(&MYSTRING, 3);
```

An array *object* can be assigned to an array *variable*. Array objects can be passed from and returned to any kind of PeopleCode function:

```
ANewFunc (&myarray);
MyFunc (&myarray);
&MyArray = YourFunc("something");
```

For example, the ReturnToServer function returns an array of nodes to which a message can be published.

Elements in an array are specified by providing a bracketed subscript after the array object reference:

```
&MyArray[1] = 123;
&temp = &memory[1][2][3];
&temp = &memory[1, 2, 3]; /* Same as preceding line.  */
MyFunc(&MyArray[7]);
MyFunc(10)[15] = "a string";
```

To access data in a two-dimensional array, you must specify both indexes. The following accesses the second item in the first subarray:

```
&VALUE  = &DOUBLE[1][2];
```

You receive an error if you use a zero or negative index in an array. Accessing an array element whose index is larger than the last array element is also an error, but storing to such an index extends the array. Any intervening elements between the former last element and the new last element are assigned a value based on the element type of the array. This is the same value as an unassigned variable of that type.

An array is an object, which means that assignments to an array are the same as for any other object. An array variable can be assigned the distinguished value NULL, which indicates the absence of any array value.

 Copyright © 1988-2007, Oracle. All rights reserved.

Array variables are supported for all scopes. This means that you can have local, global, and Component array variables.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReturnToServer

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Populating an Array

There are several ways to populate an array. The example code following each of these methods creates the exact same array, with the same elements:

- Use CreateArray when you initially create the array:

```
Local Array of Number &MyArray;

&MyArray = CreateArray(100, 200, 300);
```

- Assign values to the elements of the array:

```
Local Array of Any &MYARRAY;

&MYARRAY = CreateArray();
&MYARRAY[1] = 100;
&MYARRAY[2] = 200;
&MYARRAY[3] = 300;
```

Note. Using CreateArray without any parameters creates an array of Any.

- Use the Push method to add items to the end of the array:

```
Local Array of Number &MYARRAY;
Local Number &MYNUM;

&MYARRAY = CreateArrayRept(&MYNUM, 0);
/* this creates an empty array of number */
&MYARRAY.Push(100);
&MYARRAY.Push(200);
&MYARRAY.Push(300);
```

- Use the Unshift method to add items to the beginning of the array:

```
Local Array of Number &MYARRAY;
Local Number &MYNUM;

&MYARRAY = CreateArrayRept(&MYNUM, 0);
/* this creates an empty array of number */
&MYARRAY.Unshift(300);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYARRAY.Unshift(200);
&MYARRAY.Unshift(100);
```

You can also use CreateArrayRept (repeat) when you initially create the array. The following example creates an array with three elements: all three elements have the same data, that is, 100:

```
Local Array of Number &MYARRAY;

&MYARRAY = CreateArrayRept(100, 3);
```

Removing Items From an Array

You can remove elements from either the start or the end of the array:

- Use the POP method to select and remove an element from the end of an array

```
Local Array of Number &MYARRAY;

&MYARRAY = CreateArray();
&MYARRAY[1] = 100;
&MYARRAY[2] = 200;
&MYARRAY[3] = 300;

&ANSWER = &MYARRAY.Pop();
```

&ANSWER will equal 300.

- Use the SHIFT method to select and remove an element from the beginning of an array

```
Local Array of Number &MYARRAY;

&MYARRAY = CreateArray();
&MYARRAY[1] = 100;
&MYARRAY[2] = 200;
&MYARRAY[3] = 300;

&ANSWER = &MYARRAY.Shift();
```

&ANSWER equals 100.

Creating Empty Arrays

If you create an array using CreateArray of any data type other than ANY, the new array is *not* empty: it contains one item. If you need to create a completely empty array that contains 0 elements, use one of the following:

```
Local Array of Number &AN;
Local Array of String &AS;
Local Array of Record &AR;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local Array of Array of Number &AAN;
Local Record &REC;

&AN = CreateArrayRept(0,0); /* creates an empty array of number */
&AS = CreateArrayRept("", 0); /* creates an empty array of string */
&AR = CreateArrayRept(&REC, 0); /*create empty array of records */
&AAN = CreateArrayRept(&AN, 0); /*creates empty array of array of number */
&BOTH = CreateArray(CreateArrayRept("", 0), CreateArrayRept("", 0)); /* creates an⇒
 empty array of array of string */
```

Creating and Populating Multi-Dimensional Arrays

You can create arrays with more than one dimension. Each element in a multi-dimensional array is itself an array. For example, a two-dimensional array is really an array of arrays. Each subarray has to be created and populated as an array.

Each subarray in a multi-dimensional array must be of the same type. For example, you can't create a two dimensional array that has one subarray of type string and a second subarray of type number.

The following example creates an array of array of string, then reads two files, one into each "column" of the array. The CreateArrayRept function call creates an empty array of string (that is, the Len property is 0) but with two dimensions (that is, with two subarrays, Dimension is 2). The first Push method adds elements into the first subarray, so at the end of that WHILE loop in the example, &BOTH has Len larger than 0. The other Push methods add elements to the second subarray.

```
Local array of array of string &BOTH;
Local File &MYFILE;
Local string &HOLDER;

/* Create empty &BOTH array */
&BOTH = CreateArrayRept(CreateArrayRept("", 0), 0);

/* Read first file into first column */

&MYFILE = GetFile("names.txt", "R");
While &MYFILE.ReadLine(&HOLDER);
   &BOTH.Push(&HOLDER);
End-While;

/* read second file into second column */

&MYFILE = GetFile("numbers.txt", "R");
&LINENO = 1;
While &MYFILE.ReadLine(&HOLDER);
   If &LINENO > &BOTH.Len Then
      /* more number lines than names, use a null name */
      &BOTH.Push(CreateArray("", &HOLDER));
   Else
      &BOTH[&LINENO].Push(&HOLDER);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   End-If;
   &LINENO = &LINENO + 1;
End-While;

/* if more names than numbers, add null numbers */
for &LINENO = &LINENO to &BOTH.Len
      &BOTH[&LINENO].Push("");
End-For;
```

Local Name	Local Value
⊟ &BOTH	Array
└ Dimension	2
└ Len	3
⊟ [1]	
└ Dimension	1
└ Len	2
└ [1]	Pete
└ [2]	100
⊟ [2]	
└ Dimension	1
└ Len	2
└ [1]	Sue
└ [2]	200
⊟ [3]	
└ Dimension	1
└ Len	2
└ [1]	Laszlo
└ [2]	400
⊞ &MYFILE	File
&HOLDER	400
&LINENO	4

&BOTH array expanded in PeopleCode debugger at program end

The following code reads from a two-dimensional array and writes the data from the each subarray into a separate file.

```
Local File &MYFILE1, &MYFILE2;
Local string &STRING1, &STRING2;
Local array of array of string &BOTH;
.
/* code to load data into array would be here */
.
/* open files to be written to */

&MYFILE1 = GetFile("names.txt", "A");
&MYFILE2 = GetFile("numbers.txt", "A");

/* loop through array and write to files */

For &I = 1 To &BOTH.Len
   &J = 1;
   &STRING1 = &BOTH[&I][&J];
   &MYFILE1.writeline(&STRING1);
   &J = &J + 1;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   &STRING2 = &BOTH[&I][&J];
   &MYFILE2.writeline(&STRING2);
End-For;

&MYFILE1.Close();
&MYFILE2.Close();
```

The following example populates a multi-dimensional string array using SQL. This could be used for reading small tables.

```
Component array of array of string &ArrRunStatus;

&ArrRunStatus = CreateArrayRept(CreateArrayRept("", 0), 0);
&ArrRunStatusDescr = CreateArrayRept("", 0);

&SQL = CreateSQL("SELECT FIELDVALUE, XLATSHORTNAME FROM XLATTABLE WHERE FIELDNAME =⇒
 'RUNSTATUS'");

&LineNo = 1;
While &SQL.Fetch(&FieldValue, &XlatShortName)
      &ArrRunStatus.Push(&FieldValue);
      &ArrRunStatus[&LineNo].Push(&XlatShortName);
      &LineNo = &LineNo + 1;
End-While;
```

To search for a particular element in this array, use the following:

```
&iIndex = &ArrRunStatus.Find(&RunStatusToGet);
&RunStatusDescr = &ArrRunStatus[&iIndex][2];
```

The following example shows how to create a two-dimension array using CreateArrayRept and Push. In addition, it shows how to randomly assigns values to the cells in a two-dimension array.

```
Local array of array of string &ValueArray;

&Dim1 = 10;
&Dim2 = 5;
&ValueArray = CreateArrayRept(CreateArrayRept("", 0), 0);
For &I = 1 To &Dim1
   &ValueArray.Push(CreateArrayRept("", &Dim2));
End-For;

&ValueArray[1][1] = "V11";
&ValueArray[2][1] = "V21";

WinMessage("&ValueArray[1][1] = " | &ValueArray[1][1] | " " | "&ValueArray[2][1] =⇒
 " | &ValueArray[2][1], 0);
```

Using Flattening and Promotion

Several of the functions and methods that support arrays in PeopleCode use flattening and promotion to convert their operands to the correct dimension for the array.

Copyright © 1988-2007, Oracle. All rights reserved.

Flattening converts an array into its elements. For example, the CreateArray built-in function constructs an array from its parameters. If it is constructing a one-dimensional array and is given an array as a parameter, then it flattens that array into its elements and adds each of them to the array that it is building, rather than adding a reference to the array (which would be a dimension error) or reporting an error.

Likewise, for functions that operate on multiple-dimension arrays, if they are given a non-array parameter, they use *promotion* to convert it into an array of suitable dimension. For example, the Push method appends elements onto the end of an array. If it is operating with a two-dimensional array of Array of Number, and is given a numeric argument, it will convert the argument into a one-dimensional array of Number with the given number as its only element, and then append that to the two-dimensional array.

An array value can only be assigned to an array variable if the value and variable have both the same dimension and base type. This means you cannot assign an Array of Any to an Array of Number variable or vice-versa. You can, however, assign an Array of Number to an Any variable, as long as you do not break the rule that the base element of an array cannot be an array reference value.

Declaring Array Objects

Arrays are declared by using the Array type name, optionally followed by "of" and the type of the elements. If the element type is omitted, it is set by default to ANY.

```
Local Array of Number &MYARRAY;
Local Array &ARRAYANY;
```

Arrays can be composed of any valid PeopleCode data type, such as string, record, number, date, and so on.

Understanding the Scope of an Array Object

An array object can only be instantiated from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

Array Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateArray

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateArrayAny

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateArrayRept

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” Split

 Copyright © 1988-2007, Oracle. All rights reserved.

Array Class Methods

In this section, we discuss the array class methods.

Clone

Syntax

Clone()

Description

The Clone method returns a reference to a new array, which is a copy of the given array. It copies all levels of the array, meaning, if the array contains elements which are themselves arrays (actually references to arrays), then the copy contains references to copies of the subarrays. Furthermore, if the array contains elements that are references to the same subarray, then the copy contains references to *different* subarrays (which of course have the same value).

Parameters

None. The array object that the clone method is executed against is the array to be cloned. Assigning the result of this method assigns a reference to the new array.

Returns

An array object copied from the original.

Example

In the following example, &AAN2 contains the three elements like &AAN, but they are distinct arrays. The last line changes only &AAN2[1][1], *not* &AAN[1][1].

```
Local Array of Array of String &AAN, &AAN2;

&AAN = CreateArray(CreateArray("A", "B"), CreateArray("C", "D"), "E");
&AAN2 = &AAN.Clone();
&AAN2[1][1] = "Z";
```

After the following example, &AAN contains three elements: two references to the subarray that was &AAN[2] (with elements C and D), and a reference to a subarray with element E.

```
&AAN[1] = &AAN[2];
```

After the following example, &AAN2 contains three elements: references to two different subarrays both with elements C and D, and a subarray with element E.

```
&AAN2 = &AAN.Clone();
```

See Also

Chapter 8, “Array Class,” Subarray, page 236

Find

Syntax

Find(*value*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

For a one-dimensional array, the Find method returns the lowest index of an element in the array that is equal to the given value. If the value is not found in the array, it returns zero.

For a two-dimensional array, the Find method returns the lowest index of a subarray which has its first element equal to the given value. If such a subarray is not found in the array, it returns zero.

Note. This method works with arrays that have only one or two dimensions. You receive a runtime error if you try to use this method with an array that has more than two dimensions.

Parameters

Parameter	Description
value	The string or subarray to search for.

Returns

An index of an element or zero.

Example

Given an array &AS containing (A, B, C, D, E), the following code sets &IND to the index of D, that is, &IND has the value 4:

```
Local array of string &AS;

&AS = CreateArrayRept("", 0);
&AS.Push("A");
&AS.Push("B");
&AS.Push("C");
&AS.Push("D");
&AS.Push("E");
&IND = &AS.Find("D");
```

Given an array of array of string &AABYNAME containing (("John", "July"), ("Jane", "June"), ("Norm", "November"), the following code sets &IND to the index of the subarray starting with "Jane", that is, &IND has the value 2:

```
&NAME = "Jane";
&IND = &AABYNAME.Find(&NAME);
```

See Also

Chapter 8, “Array Class,” Replace, page 231 and Chapter 8, “Array Class,” Substitute, page 237.

Get

Syntax

Get(*index*)

Description

Use the Get method to return the *index* element of an array. This method is used with the Java PeopleCode functions, instead of using subscripts (which aren’t available in Java.)

 Copyright © 1988-2007, Oracle. All rights reserved.

Using this method is the same as using a subscript to return an item of an array. In the following example, the two lines of code are identical:

```
&Value = &MyArray[8];
&value = &MyArray.Get(8);
```

Parameters

Parameter	Description
index	The array element to be accessed.

Returns

An element in an array.

See Also

Chapter 8, "Array Class," Set, page 233

Chapter 19, "Java Class," page 615

Join

Syntax

Join([*separator* [, *arraystart*, *arrayend* [,*stringsizehint*]]])

Description

The Join method *converts* the array that is executing the method into a string by converting each element into a string and *joining* these strings together, separated by *separator*.

Note. Join does *not* join two arrays together.

Each array or subarray to be joined is preceded by the string given by *arraystart* and followed by the string given by *arrayend*. If the given array is multi-dimensional, then (logically) each subarray is first joined, then the resulting strings are joined together.

Parameters

Parameter	Description
separator	Specifies what the elements in the resulting string should be separated with in the resulting string. *Separator* is set by default to a comma (",").
arraystart	Specifies what each array or subarray to be joined should be preceded with in the resulting string. *arraystart* is set by default to a left parenthesis ("(").
arrayend	Specifies what each array or subarray to be joined should be followed by in the resulting string. a*rrayend* is set by default to a right parenthesis (")").
stringsizehint	Specify a hint to the Join method about the resulting size of the string. This can improve performance if your application is concatenating a large number of string. See the Example section below.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A string containing the converted elements of the array.

Example

The following example:

```
Local array of array of number &AAN;

&AAN = CreateArray(CreateArray(1, 2), CreateArray(3, 4), 5);
&STR = &AAN.Join(", ");
```

produces in &STR the string:

```
((1, 2), (3, 4), 5)
```

The following example makes use of the *stringsizehint* parameter. The following application class passes the resulting string size hint in the Value property.

```
class StringBuffer
   method StringBuffer(&InitialValue As string, &MaxSize As integer);
   method Append(&New As string);
   method Reset();
   property string Value get set;
   property integer Length readonly;
private
   instance array of string &Pieces;
   instance integer &MaxLength;
end-class;

method StringBuffer
   /+ &InitialValue as String, +/
   /+ &MaxSize as Integer +/
   &Pieces = CreateArray(&InitialValue);
   &MaxLength = &MaxSize;
   &Length = 0;
end-method;

method Reset
   &Pieces = CreateArrayRept("", 0);
   &Length = 0;
end-method;

method Append
   /+ &New as String +/
   Local integer &TempLength = &Length + Len(&New);
   If &Length > &MaxLength Then
      throw CreateException(0, 0, "Maximum size of StringBuffer exceeded(" | &Max⇒
Length | ")");
   End-If;
   &Length = &TempLength;
   &Pieces.Push(&New);
end-method;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
get Value
   /+ Returns String +/
   Local string &Temp = &Pieces.Join("", "", "", &Length);
   /* collapse array now */
   &Pieces = CreateArrayRept("", 0);
   &Pieces.Push(&Temp); /* start out with this combo string */
   Return &Temp;
end-get;

set Value
   /+ &NewValue as String +/
   /* Ditch our current value */
   %This.Reset();
   &Pieces.Push(&NewValue);
end-set;
```

The following code concatenates strings.

```
While &file.ReadLine(&line)
   &S.Append(&line);
   &S.Append(&separator);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Split

Next

Syntax

Next(&*index*)

Description

The Next method increments the given index variable. It returns true if and only if the resulting index variable refers to an existing element of the array. Next is typically used in the condition of a WHILE clause to process a series of array elements up to the end of the array.

&index must be a variable of type integer, or of type Any initialized to an integer, as Next attempts to update it.

If you want to start from the first element of the array, start Next with an index variable with the value zero. The first thing Next does is to increment the value by one.

Parameters

Parameter	Description
&index	The array element where processing should start. *&index* must be a variable of type integer, or of type Any initialized to an integer, as Next attempts to update it.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

True if the resulting index refers to an existing element of the array, False otherwise.

Example

Next can be used in a While loop to iterate through an array in the following manner:

```
&INDEX = 0;
While &A.Next(&INDEX)
   /* Process &A[&INDEX] */
End-While;
```

In the following code example, &BOTH is a two-dimensional array. This example writes the data from each subarray in &BOTH into a different file.

```
&I = 0;
While &BOTH.Next(&I)
   &J = 1;
   &STRING1 = &BOTH[&I][&J];
   &MYFILE1.writeline(&STRING1);
   &J = &J + 1;
   &STRING2 = &BOTH[&I][&J];
   &MYFILE2.writeline(&STRING2);
End-While;
```

See Also

Chapter 8, “Array Class,” Len, page 239

Pop

Syntax

Pop()

Description

The Pop method removes the last element from the array and returns its value.

Parameters

None.

Returns

The value of the last element of the array. If the last element is a subarray, the subarray is returned.

Example

The Pop method can be used with the Push method to use an array as a stack. To put values on the end of the array, use Push. To take the values back off the end of the array, use Pop.

 Copyright © 1988-2007, Oracle. All rights reserved.

Suppose we have a two-dimensional array &SUBPARTS which gives the subparts of each part of some assemblies. Each row (subarray) of &SUBPARTS starts with the name of the part, and then has the names of the subparts. Assuming there are no "loops" in this data, the following code puts all the subparts, subsubparts, and so on, of the part given by &PNAME into the array &ALLSUBPARTS, in "depth first order" (that is, subpart1, subparts of subpart1, …, subpart2, subparts of subpart2, …). We stack the indexes into &SUBPARTS when we want to go down to the subsubparts of the current subpart.

```
Local array of array of string &SUBPARTS;
Local array of string &ALLSUBPARTS;
Local array of array of number &STACK;
Local array of number &CUR;
Local string &SUBNAME;

/* Set the ALLSUBPARTS array to an empty array of string.  */

&ALLSUBPARTS = CreateArrayRept("dummy", 0);

/* Start with the part name.  */

&STACK = CreateArray(CreateArray(&SUBPARTS.Find(&PNAME), 2));
While &STACK.Len > 0
   &CUR = &STACK.Pop();
   If &CUR[1] <> 0 And
         &CUR[2] <= &SUBPARTS[&CUR[1]].Len Then

      /* There is a subpart here.  Add it.  */
      &SUBNAME = &SUBPARTS[&CUR[1], &CUR[2]];
      &ALLSUBPARTS.Push(&SUBNAME);

      /* Tour its fellow subparts later.  */
      &STACK.Push(CreateArray(&CUR[1], &CUR[2] + 1));

      /* Now tour its subsubparts.  */
      &STACK.Push(CreateArray(&SUBPARTS.Find(&SUBNAME), 2));

   End-If;
End-While;
```

See Also

Chapter 8, “Array Class,” Push, page 229, Chapter 8, “Array Class,” Replace, page 231, Chapter 8, “Array Class,” Shift, page 234 and Chapter 8, “Array Class,” Unshift, page 238.

Push

Syntax

Push(*paramlist*)

Where *paramlist* is an arbitrary-length list of values in the form:

value1 [, *value2*] ...

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Push method adds the values in *paramlist* onto the end of the array executing the method. If a value is not the correct dimension, it is flattened or promoted to the correct dimension first, then the resulting values are added to the end of the array.

Considerations Using Arrays With Object References

This method only adds an element to the end of an array. It does not clone or otherwise deep-copy the parameters. For example, if you are adding a reference to an object, Push just adds a reference to the object at the end of the array. This is similar to an assignment. It is *not* making a copy of the object. The following code snippet only puts a reference to the same record onto the end of the array.

```
While &SQL.Fetch(&Rec);
&MYARRAY.Push(&Rec);
...
End-While;
```

Even though the array is growing, all the elements point to the same record. You have only as many standalone record objects as you create. The following code snippet creates new standalone records, so each element in the array points to a new object:

```
local Record &FetchedRec = CreateRecord(Record.PERSONAL_DATA);

While &SQL.Fetch(&FetchedRec)
     &MYARRAY.Push(&FetchedRec);
     &FetchedRec = CreateRecord(Record.PERSONAL_DATA);
End-While;
```

Parameters

Parameter	Description
paramlist	An arbitrary-length list of values, separated by commas.

Returns

None.

Example

The following example loads an array with data from a database table.

```
Local array of record &MYARRAY;
Local SQL &SQL;

&I = 1;
&SQL = CreateSQL("Select(:1) from %Table(:1) where EMPLID like &lsquo;8%&rsquo;",⇒
 &REC);
While &SQL.Fetch(&REC);
   &MYARRAY.Push(CreateRecord(RECORD.PERSONAL_DATA));
   &I = &I + 1;
   &REC.CopyFieldsTo(&MYARRAY[&I]);
End-While;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 8, “Array Class,” Pop, page 228, Chapter 8, “Array Class,” Replace, page 231, Chapter 8, “Array Class,” Shift, page 234 and Chapter 8, “Array Class,” Unshift, page 238.

Chapter 8, “Array Class,” Using Flattening and Promotion, page 221

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, “Understanding Objects and Classes in PeopleCode,” Assigning Objects

Replace

Syntax

```
Replace(start, length[, paramlist])
```

Where *paramlist* is an arbitrary-length list of values in the form:

```
value1 [, value2] ...
```

Description

Replace replaces the *length* elements starting at *start* with the given values, if any. If *length* is zero, the insertion takes place *before* the element indicated by *start*. Otherwise, the replacement starts with *start* and continues up to and including *length*, replacing the existing values with *paramlist*.

If a negative number is used for *start*, it indicates the starting position relative to the last element in the array, such that −1 indicates the position just *after* the end of the array. To insert at the end of the array (equivalent to the Push method), use a *start* of −1 and a *length* of 0.

If a negative number is used for *length*, it indicates a length measuring downward to lower indexes. Both flattening and promotion can be applied to change the dimension of the supplied parameters to match the elements of the given array.

Similar to how the built-in function Replace is used to update a string, the Replace method is a general way to update an array, and can cause the array to grow or shrink.

See Chapter 8, “Array Class,” Using Flattening and Promotion, page 221.

Using Replace to Remove an Element

You can use the Replace method to remove an element from an array. Just specify the item you want replaces, with *length* equal to one.

The following example removes the item from &Index:

```
&Array.Replace(&Index, 1);
```

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
start	Specifies where to start replacing the given elements in the array. If a negative number is used for *start*, it indicates the starting position relative to the last element in the array.
length	Specifies the number of elements in the array to be replaced.
paramlist	Specifies values to be used to replace existing values in the array. This parameter is optional.

Returns

None.

Example

For example, given the following array:

```
Local array of string &AS;

&AS = CreateArray("AA", "BB", "CC");
```

After executing the next code, the array &AN will contain four elements, ZZ, YY, BB, CC:

```
&AS.Replace(1, 1, "ZZ", "YY");
```

After executing the next code, the array &AN will contain three elements, ZZ, MM, CC:

```
&AS.Replace(2, 2, "MM");
```

After executing the next code, the array &AN will contain three elements, ZZ, OO, CC.

```
&AS.Replace( - 2, - 1, "OO");
```

Local Name	Local Value
⊟ &AS	Array
└ Dimension	1
└ Len	3
└ [1]	ZZ
└ [2]	OO
└ [3]	CC

&AS expanded in PeopleCode debugger

See Also

Chapter 8, “Array Class,” Substitute, page 237 and Chapter 8, “Array Class,” Find, page 223.

Reverse

Syntax

```
Reverse()
```

Description

The Reverse method reverses the order of the elements in the array.

If the array is composed of subarrays, the Reverse method reverses only the elements in the super-array, it doesn’t reverse all the elements in the subarrays. For example, the following:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&AN = CreateArray(CreateArray(1, 2), CreateArray(3, 4), CreateArray(5, 6)).reverse⇒
();
```

results in &AN containing:

```
((5,6), (3,4), (1,2)
```

Parameters

None.

Returns

None.

Example

Suppose you had the following array.

```
Local Array of Sting &AS;
&AS = CreateArray("R", "O", "S", "E");
```

If you executed the Reverse method on this array, the elements would be ESOR.

See Also

Chapter 8, “Array Class,” Sort, page 234

Set

Syntax

Set(*index*)

Description

Use the Set method to set the value of the *index* element of an array. This method is used with the Java PeopleCode functions, instead of using subscripts (which aren’t available in Java.)

Using this method is the same as using a subscript to reference an item of an array. In the following example, the two lines of code are identical:

```
&MyArray[8] = &MyValue;
&MyArray.Set(8) = &MyValue;
```

Parameters

Parameter	Description
index	The array element to be accessed.

Returns

None.

See Also

Chapter 8, “Array Class,” Get, page 224

Chapter 19, “Java Class,” page 615

Copyright © 1988-2007, Oracle. All rights reserved.

Shift

Syntax

Shift()

Description

Use the Shift method to remove the first element from the array and return it. Any following elements are "shifted" to an index of one less than they had.

Parameters

None.

Returns

Returns the value of the first element of the array. If the first element is a subarray, the subarray is returned.

Example

```
For &I = 1 to &ARRAY.Len;
   &ITEM = &ARRAY.Shift;
   /* do processing */
End-For;
```

See Also

Chapter 8, "Array Class," Pop, page 228, Chapter 8, "Array Class," Push, page 229, Chapter 8, "Array Class," Replace, page 231 and Chapter 8, "Array Class," Unshift, page 238.

Sort

Syntax

Sort(*order*)

Description

The Sort method rearranges the elements of the array executing the method into an order.

The type of sort done by this function, that is, whether it is a linguistic or binary sort, is determined by the Sort Order Option on the PeopleTools Options page.

If the array is one-dimensional, the elements are arranged in either ascending or descending order.

The type of the first element is used to determine the kind of comparison to be made. Any attempt to sort an array whose elements are not all of the same type results in an error.

If *order* is "A", the order is ascending; if it is "D", the order is descending. The comparison between elements is the same one as if done using the PeopleCode comparison operators (<, >, =, and so on.)

Note. If you execute this method on a server, the string sorting order is determined by the character set and localization of the server.

If the array is two-dimensional, the subarrays are arranged in order by the first element of each subarray. Sorting an array whose subarrays have different types of first elements will result in an error. The comparison is done by using the PeopleCode comparison operators (<, >, =, and so on.)

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. This method works with arrays that have only one or two dimensions. You receive a runtime error if you try to use this method with an array that has more than two dimensions.

Parameters

Parameter	Description
order	Specifies whether the array should be sorted in ascending or descending order. Values for *order* are:

Value	Description
A	Ascending
D	Descending

Returns

None.

Example

The following example changes the order of the elements in array &A to be ("Frank", "Harry", "John").

```
&A = CreateArray("John", "Frank", "Harry");
&A.Sort();
```

The following example changes the order of the elements in array &A to be (("Frank", 1957), ("Harry", 1928), ("John", 1952)).

```
&A = CreateArray(CreateArray("John", 1952), CreateArray("Frank", 1957), Create⇒
Array("Harry", 1928));
```

Local Name	Local Value
⊟ &A	Array
└ Dimension	2
└ Len	3
⊟ [1]	
└ Dimension	1
└ Len	2
└ [1]	John
└ [2]	1952
⊟ [2]	
└ Dimension	1
└ Len	2
└ [1]	Frank
└ [2]	1957
⊟ [3]	
└ Dimension	1
└ Len	2
└ [1]	Harry
└ [2]	1928

&A expanded in PeopleCode debugger

```
&A.Sort("A");
```

Copyright © 1988-2007, Oracle. All rights reserved.

Local Name	Local Value
⊟ &A	Array
└Dimension	2
└Len	3
⊟ [1]	
└Dimension	1
└Len	2
└[1]	Frank
└[2]	1957
⊟ [2]	
└Dimension	1
└Len	2
└[1]	Harry
└[2]	1928
⊟ [3]	
└Dimension	1
└Len	2
└[1]	John
└[2]	1952

&A expanded in PeopleCode debugger, showing code results

See Also

Chapter 8, "Array Class," Reverse, page 232

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Subarray

Syntax

```
Subarray(start, length)
```

Description

The Subarray method creates a new array from an existing one, taking the elements from *start* for a total of *length*. If *length* if omitted, all elements from *start* to the end of the array are used.

If the array is multi-dimensional, the subarrays of the created array are *references* to the same subarrays from the existing array. This means if you make changes to the original subarrays, the referenced subarrays are also changed. To make distinct subarrays, use the Clone method.

Parameters

Parameter	Description
start	Specifies where in the array to begin the subarray.
length	Specifies the number of elements in the array to be part of the subarray.

Returns

An array object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

To make a distinct array from a multi-dimensional array, use the following:

```
&A = &AAN.Subarray(1, 2).Clone();
```

See Also

Chapter 8, "Array Class," Clone, page 223

Substitute

Syntax

Substitute(*old_val*, *new_val*)

Description

The Substitute method replaces every occurrence of a value found in an array with a new value. To replace an element that occurs in a specific location in an array, use Replace.

If the array is one-dimensional, Substitute replaces every occurrence of the *old_val* in the array with *new_val*.

If the array is two-dimensional, Substitute replaces every subarray whose first element is equal to *old_val*, with the subarray given by *new_val*.

Note. This method works with arrays that have only one or two dimensions. You receive a runtime error if you try to use this method with an array that has more than two dimensions.

Parameters

Parameter	Description
old_val	Specifies the existing value in the array to be replaced.
new_val	Specifies the value with which to replace occurrences of *old_val*.

Returns

None

Example

The following example changes the array &A to be ("John", "Jane", "Hamilton").

```
&A = CreateArray();
&A[1] = "John";
&A[2] = "Jane";
&A[3] = "Henry";
&A.Substitute("Henry", "Hamilton");
```

The following example changes the array &A to be (("John", 1952), ("Jane", 1957), ("Hamilton", 1971), ("Frank", 1961)).

```
&A = CreateArray(CreateArray("John", 1952), CreateArray("Jane", 1957), CreateArray⇒
("Henry", 1928), CreateArray("Frank", 1961));
&A.Substitute("Henry", CreateArray("Hamilton", 1971));
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 8, “Array Class,” Find, page 223 and Chapter 8, “Array Class,” Replace, page 231.

Unshift

Syntax
Unshift(*paramlist*)

Where *paramlist* is an arbitrary-length list of values in the form:
value1 [, *value2*] ...

Description

The Unshift method adds the given elements to the *start* of the array. Any following elements are moved up to indexes that are larger by the number of values moved. Flattening and Promotion are used to change the dimension of the supplied parameters to be one less than that of the given array.

Parameters

Parameter	Description
paramlist	Specifies values to be added to the start of the array.

Returns

None.

Example

The following code changes &A to be ("x", "Y", "a", "B", "c").

```
&A = CreateArray("a", "B", "c");
&A.Unshift("x", "Y");
```

See Also
Chapter 8, “Array Class,” Pop, page 228, Chapter 8, “Array Class,” Push, page 229, Chapter 8, “Array Class,” Replace, page 231, Chapter 8, “Array Class,” Shift, page 234 and Chapter 8, “Array Class,” Using Flattening and Promotion, page 221.

Array Class Properties

In this section, we discuss the array class properties.

Dimension

Description

The Dimension property is the number of "Array" type names from the declaration of the array, also called subarrays. This property returns a number.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example sets &DIM to 2.

```
Local Array of Array of Number &AAN;
&DIM = &AAN.Dimension;
```

Len

Description

The Len property is the current number of elements in the array. This property can be updated. Setting it to a negative value results in an error.

If this property is set to a smaller (nonnegative) number than its current value, the array is truncated to that length, discarding any elements whose indexes are larger than the given new length.

If this property is set to a number larger than its current value, the array is extended to the new length. Any new elements are set to a default value based on the element type of the array.

This property is read-write.

Example

The following is a test of whether an array is empty:

```
If &ARR.Len = 0 then
/* &ARR is empty.  */
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 9

BPEL Classes

This chapter discusses the BPEL classes, as well as describes:

- Scope of the BPEL classes
- Data types of the BPEL classes
- How to import the BPEL classes
- How to create a BPEL object
- BPEL reference material

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Integrating with BPEL Process-Based Services"

Understanding the BPEL Classes

Use the Business Process Execution Language (BPEL) classes for launching, as well as controlling, a BPEL process instance. A BPEL process instance is associated with a service operation, that is, a message transaction, such as an outbound asynchronous message. You must have first created the service operation using PeopleSoft Pure Internet Architecture pages. When you define the service operation you specify the node used for connecting to the third-party system, as well as the routings used by the messages, and so on.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Integrating with BPEL Process-Based Services"

Scope of the BPEL Classes

The BPEL classes can be instantiated only from PeopleCode.

The BPEL classes can be called from a component, an internet script, Integration Broker, or an Application Engine program.

BPEL classes can be of Local, Global, or Component scope.

Copyright © 1988-2007, Oracle. All rights reserved.

Data Types of the BPEL Classes

Every BPEL class is its own data type. That is, the class of BPELUtil is declared as type BPELUtil, and so on.

The following are the data types of the BPEL classes:

- AsyncFFSend
- BPELUtil
- IBUtil

How to Import the BPEL Classes

The BPEL classes are *not* built-in classes, like rowset, field, record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the BPEL classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_BPEL contains the BPEL classes that you will use:

- AsyncFFSend
- BPELUtil
- IBUtil

The import statement you should use is as follows:

```
import PT_BPEL:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

See Also

Chapter 6, “Application Classes,” page 153

How to Create a BPEL Object

After you’ve imported the BPEL classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of the BPELUtil class, as the variable &MyBU, with local scope:

```
import PT_BPEL:*;

Local BPELUtil &MyBU = Create BPELUtil();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 9, “BPEL Classes,” BPEL Classes Constructors, page 243

BPEL Classes Constructors

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the BPEL classes.

AsyncFFSend

Syntax

```
AsyncFFSend()
```

Description

Use the AsyncFFSend constructor to instantiate an AsyncFFSend object.

Parameters

None.

Returns

A reference to an AsyncFFSend object.

See Also

Chapter 9, “BPEL Classes,” AsyncFFSend Class, page 244

BPELUtil

Syntax

```
BPELUtil()
```

Description

Use the BPELUtil constructor to instantiate a BPELUtil object.

Parameters

None.

Returns

A reference to a BPELUtil object.

See Also

Chapter 9, “BPEL Classes,” BPELUtil Class, page 245

Copyright © 1988-2007, Oracle. All rights reserved.

IBUtil

Syntax

```
IBUtil()
```

Description

Use the IBUtil constructor to instantiate an IBUtil object.

Parameters

None.

Returns

A reference to an IBUtil object.

See Also

Chapter 9, "BPEL Classes," IBUtil Class, page 248

AsyncFFSend Class

This class provides the application handler used when an asynchronous one way operation is called from the local system. This class extends the integration broker ISend class OnRequestSend method.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

AsyncFFSend Class Method

This section describes the AsyncFFSend class method OnRequestSend.

OnRequestSend

Syntax

```
OnRequestSend(&Msg)
```

Description

Use the OnRequestSend method to implement the integration broker ISend class OnRequestSend method.

This method is generally used to associate the WSA_MessageID when making an asynchronous one way request to the BPEL process so that it can be tracked.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&Msg	Specify an already populated and instantiated message object.

Returns

A message object.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Understanding Sending and Receiving Messages

BPELUtil Class

This class provides the utility methods to launch and monitor BPEL processes.

BPELUtil Class Methods

This section discusses the BPELUtil class methods, in alphabetical order.

GetASyncBPELProcessInstanceUrl

Syntax

GetASyncBPELProcessInstanceUrl(*TransactionID*)

Description

Use the GetASyncBPELProcessInstanceUrl method to return the complete URL that can be used to monitor an asynchronous BPEL process instance. The URL can be used to access the BPEL monitor for this particular BPEL process instance.

Parameters

Parameter	Description
TransactionID	Specify the transaction ID for the asynchronous BPEL process instance that you want to monitor.

Returns

A string containing a complete URL if successful, a Null or invalid string otherwise.

See Also

Chapter 9, "BPEL Classes," GetSyncBPELProcessInstanceUrl, page 246

Copyright © 1988-2007, Oracle. All rights reserved.

GetBPELProcessBrowserUrl

Syntax

GetBPELProcessBrowserUrl(*NodeName*)

Description

Use the GetBPELProcessBrowserUrl to return a complete URL of the WSIL process browser. It can be used to discover all the web services that are being hosted by the given BPEL domain.

Parameters

Parameter	Description
NodeName	Specify the name of the node associated with the BPEL process that you want to access, as a string.

Returns

A string containing the complete URL if successful, an invalid URL string otherwise.

GetOperationType

Syntax

GetOperationType(*OperationName*)

Description

Use the GetOperationType method to return the type of the specified operation.

Parameters

Parameter	Description
OperationName	Specify the name of the operation for which you want to determine the type, as a string.

Returns

A string containing the operation type. Valid values are:

Value	Description
S	The operation type is synchronous.
A	The operation type is asynchronous.
Null	There is an error in the method execution.

GetSyncBPELProcessInstanceUrl

Syntax

GetSyncBPELProcessInstanceUrl(*NodeName, MessageID*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetSyncBPELProcessInstanceUrl method to return the complete URL that can be used to monitor the specified synchronous BPEL process instance.

Parameters

Parameter	Description
NodeName	Specify the node name associated with the synchronous BPEL process instance, as a string.
MessageID	Specify the message ID associated with the external system for this BPEL process instance, as a string.

Returns

A string containing a complete URL if successful, an invalid URL otherwise.

See Also

Chapter 9, "BPEL Classes," GetASyncBPELProcessInstanceUrl, page 245

LaunchASyncBPELProcess

Syntax

```
LaunchASyncBPELProcess(operationName, &Msg, Username, Password)
```

Description

Use the LaunchASyncBPELProcess method to invoke an asynchronous BPEL process.

Parameters

Parameter	Description
OperationName	Specify the name of the operation associated with the asynchronous BPEL process that you want to start, as a string.
&Msg	Specify an already instantiated and populated message object that is created from the message definition that is associated with the asynchronous BPEL process.
Username	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.
Password	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.

Returns

A string that contains the transaction ID of the published message if successful, a Null string otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 9, "BPEL Classes," LaunchSyncBPELProcess, page 248

LaunchSyncBPELProcess

Syntax

LaunchSyncBPELProcess(*operationName*, *&Msg*, *Username*, *Password*)

Description

Use the LaunchSyncBPELProcess method to invoke a synchronous BPEL process.

Parameters

Parameter	Description
OperationName	Specify the name of the operation associated with the synchronous BPEL process that you want to start, as a string.
&Msg	Specify an already instantiated and populated message object that is created from the message definition that is associated with the synchronous BPEL process.
Username	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.
Password	*Username* and *Password* are used to override the security credentials that are sent by default by Integration Broker when invoking a secure BPEL process (a web service.) If you specify an empty string, the Integration Broker credentials are used. If you specify a value, it's sent as part of the security credentials of the BPEL process.

Returns

A message object populated with the reply message if successful, a Null object otherwise.

See Also

Chapter 9, "BPEL Classes," LaunchASyncBPELProcess, page 247

IBUtil Class

The IBUtil class provides the utility methods to access integration broker metadata.

IBUtil Class Methods

This section discusses the BPELUtil class methods, in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

GetBPELConsoleUrl

Syntax

GetBPELConsoleUrl(*NodeName*)

Description

Use the GetBPELConsoleUrl method to return the BPEL console URL.

A transaction ID is an internal identifier for the message that is published. A message ID is the external ID that is used by the external system for identifying the transaction. The transaction ID is appended to the URL generated by this method. This concatenated URL is used to monitor the BPEL process instance.

Use the GetMessageId method to return the message ID.

Parameters

Parameter	Description
NodeName	Specify the name of the node associated with the BPEL process that you want to monitor.

Returns

A string containing a complete URL if successful, Null otherwise.

See Also

Chapter 9, "BPEL Classes," GetMessageId, page 249

GetBPELDomain

Syntax

GetBPELDomain(*Nodename*)

Description

Use the GetBPELDomain method to determine the name of the domain of the specified node. This is used to construct the URL for accessing a BPEL process instance.

Parameters

Parameter	Description
Nodename	Specify the name of the node that you want to determine the domain for, as a string.

Returns

A string containing the domain if successful, Null otherwise.

GetMessageId

Syntax

GetMessageId(*TransactionId*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetmessageId method to return the message ID used to monitor a BPEL process instance.

A transaction ID is an internal identifier for the message that is published. A message ID is the external ID that is used by the external system for identifying the transaction.

Parameters

Parameter	Description
TransactionId	Specify the transaction ID associated with the BPEL process instance that you want to monitor, as a string.

Returns

A string containing the message ID, Null otherwise.

See Also

Chapter 9, "BPEL Classes," GetBPELConsoleUrl, page 249

GetNode

Syntax

```
GetNode(MessageId)
```

Description

Use the GetNode method to return the node name of the node involved in processing of the message.

Parameters

Parameter	Description
MessageId	Specify the message ID associated with the BPEL process instance that you want to get the node for, as a string.

Returns

A string containing the node name if successful, Null otherwise.

GetNumberOfRoutings

Syntax

```
GetNumberOfRoutings(ServiceOperationName)
```

Description

Use the GetNumberOfRoutings method to return the number of routings for the specified service operation.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ServiceOperationName	Specify the name of the service operation that you want to discover the number of routings for, as a string.

Returns

A number.

GetStatus

Syntax

```
GetStatus(TransactionId)
```

Description

Use the GetStatus method to return the status of the Integration Broker process specified by the transaction Id.

Parameters

Parameter	Description
TransactionId	Specify the transaction ID for the BPEL process instance that you want to check the status of, as a string.

Returns

A string containing the status, Null otherwise.

GetTransactionId

Syntax

```
GetTransactionId(MessageID)
```

Description

Use the GetTransactionId method to return the transaction ID from the message. You must use the Publish or SyncRequest method on the message before you use this method.

Parameters

Parameter	Description
MessageID	Specify the message ID associated with the BPEL process instance, as a string.

Returns

A string containing the transaction ID associated with the published message, Null otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 10

Business Interlink Class

This chapter provides an overview of Business Interlink class and discusses the following topics:

- Using the Interlink object
- Deciding which methods to use
- Using the incoming Business Interlink methods and properties
- Understanding the state of an Interlink object
- Using Business Interlink with Application Engine
- Using the Data Type of a Business Interlink object
- Understanding the scope of a Business Interlink object
- Business Interlink class built-in function
- Business Interlink class reference
- Configuration parameters

Understanding Business Interlink Class

The PeopleSoft Business Interlink framework provides a gateway for PeopleSoft applications to the services of any external system. This framework enables any PeopleSoft component (that is, a page, an Application Engine program, and so on) to integrate with any external system in near real-time and batch modes.

This framework enables a PeopleCode program to map the input and outputs of a Business Interlink definition to PeopleCode variables and record fields, then, using an Interlink object, call the Interlink plug-in, which in turn calls the external system, passes the information to the external system, and returns values to the PeopleCode program.

This documentation describes only the Interlink object portion of the PeopleSoft Business Interlink framework.

Each Interlink object is based on a Business Interlink definition, which is created in PeopleSoft Application Designer. An Interlink object can be based on only a single Business Interlink definition.

See Also

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

Copyright © 1988-2007, Oracle. All rights reserved.

Using the Interlink Object

After you instantiate an Interlink object, use the Interlink object to:

- Populate the input buffer
- Send the data to the Interlink plug-in (by using the Execute method)
- Fetch the outputs from the buffer
- Check for status and error messages

After the Business Interlink object is instantiated, you can assign values from constants, PeopleSoft variables, or record fields to the inputs of that Business Interlink object.

When you execute the Business Interlink object, it loads the appropriate Business Interlink Plug-in and passes itself to that Business Interlink Plug-in. The Business Interlink Plug-in processes the input data, passing the input values of the Business Interlink object to the external system and then fills the output values of the Business Interlink object (if there are outputs).

Deciding Which Methods to Use

After you create your Business Interlink definition, you must use PeopleCode to instantiate an Interlink object and execute the Business Interlink plug-in. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the Business Interlink definition from the PeopleSoft Application Designer Project View into an open PeopleCode edit pane. PeopleSoft Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify.

In this section, we discuss how to:

- Execute the Business Interlink object.
- Support batch input and output.
- Support rowsets.
- Use the flat table methods.
- Support dynamic output.
- Use hierarchical data (BIDocs).

See Also

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

Executing the Business Interlink Object

In most cases, you must use the Execute method to execute the Business Interlink object. However, for bulk input, you can use the BulkExecute method instead.

The Execute and BulkExecute methods return a value you can use for status and error checking.

 Copyright © 1988-2007, Oracle. All rights reserved.

Supporting Batch Input and Output

The methods discussed in this section, except for BulkExecute, add input values to the Business Interlink object one set, or row, at a time. The call to the Business Interlink Plug-in occurs only once. All the input data is passed with the single Execute. All output is returned as batch as well. The methods then get the output values one set, or row, at a time.

If you're sending a large amount of data to the input buffers, instead of adding one input row at a time, you might write the data to a staging table, then use the BulkExecute method. This method automatically executes; that is, you don't have to use the Execute method. It also automatically fills the output record specified with the method with all the output values in every row in the output buffer if you've specified an output record.

Supporting Rowsets

If your data is mapped into rowsets, you may want to use the InputRowset method. This method takes a standard rowset object to populate the inputs for the Business Interlink object. You can use the FetchIntoRowset method to repopulate the rowset with new data.

Using the Flat Table Methods

If your data is in a flat table structure, you can use the flat table methods. AddInputRow adds rows of input to the Business Interlink object; FetchNextRow fetches rows of output from the Business Interlink object.

Supporting Dynamic Output

A Business Interlink can have dynamic output, meaning that the outputs for a Business Interlink object are changeable in data type or number of outputs.

Business Interlinks supplies a set of methods to support dynamic output. These methods enable you to interrogate the output buffer programmatically to determine the number of fields (columns), their types, and their values.

The following methods support dynamic output:

- GetFieldCount
- GetFieldType
- GetFieldValue
- MoveFirst
- MoveNext

Use the MoveFirst method to move to the first column, first row of the output buffer, and within a loop, use the MoveNext method to move to each row on the output buffer.

Within a MoveFirst (or MoveNext) loop, use the GetFieldCount method to get the number of columns in the output buffer, which is also the number of outputs for this Business Interlink object. Then you can extract the outputs from the buffer in a loop.

Within the GetFieldCount loop, use GetFieldName to get the name of the output, GetFieldValue to get the value of the output (which will be returned as a string), and GetFieldType to get the type of the output (if the output is not a string type, you will convert it to this type).

See Chapter 10, "Business Interlink Class," GetFieldCount, page 285; Chapter 10, "Business Interlink Class," GetFieldType, page 286; Chapter 10, "Business Interlink Class," GetFieldValue, page 287; Chapter 10,

Copyright © 1988-2007, Oracle. All rights reserved.

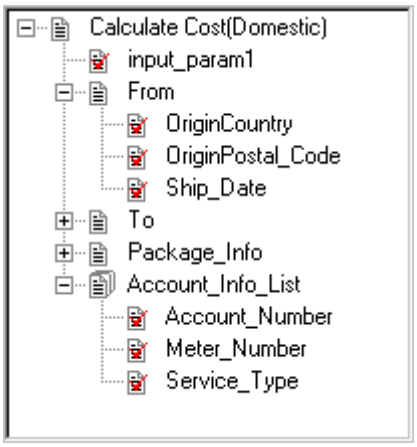
“Business Interlink Class,” MoveFirst, page 299 and Chapter 10, “Business Interlink Class,” MoveNext, page 300.

Using Hierarchical Data (BIDocs)

A Business Interlink can have hierarchical data. Think of the structure as a tree, with a root doc, node docs, and values. The hierarchical data methods and objects are also referred to as BIDocs.

Input Structures

The following shows an example of an input structure:



Example input structure

The *root doc* is Calculate Cost(Domestic). A root doc can contain both values and node docs.

The *node docs* are From, To, Package_Info and Account_Info_List. Each node can contain both values and child nodes. Node docs can be further described as either:

- *Simple node docs* have only one set of values for a single instance of a root doc. From, To, Package_Info are all simple node docs.
- *List node docs* can contain more than one set of values for a single instance of a root doc. Account_Info_List is a list node doc.

The values in the From node are OriginCountry, OriginPostal_Code, and Ship_Date. The value within the root node is input_param1. Notice that there are values both within the root doc and within node docs.

Business Interlinks support hierarchical input structures with the following methods:

- GetInputDocs
- AddDoc
- AddValue
- AddNextDoc

The GetInputDocs method returns a reference to the root doc of an input structure. From the previous example, it returns a reference to Calculate Cost(Domestic).

Use the AddDoc method to access the node docs. From the previous example, you would use AddDoc to access the From, To, Package_Info, and Account_Info_List node docs. If any of these nodes contained nodes, you could use AddDoc to access those as well.

 Copyright © 1988-2007, Oracle. All rights reserved.

Use the AddValue method to set values. From the previous example, you would use AddValue to set the value for input_param1, OriginCountry, OriginPosta_Code, and Ship_date. You must call AddDoc on a node before you can call AddValue for its values.

Use the AddNextDoc method to access the following:

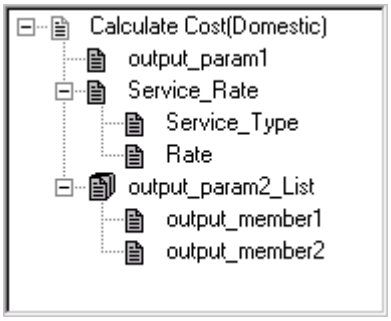
- If a node doc is a list, that is, it can contain more than one set of values, use AddNextDoc to reference the next set of values.
- To add another copy of the entire input structure, use AddNextDoc to return a reference to the next root doc.

The following code example sets values for the node doc From, which is a simple node doc. It also sets the values for Account_Info_List, which is a list node doc.

```
&Calc_Input = &QE_COST.GetInputDocs("");

&FromDoc = &Calc_Input.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

&Account_Doc = &Calc_Input.AddDoc("Account_Info_List");
&ret = &Account_Doc.AddValue("Account_Number", "CT-8001");
&ret = &Account_Doc.AddValue("Meter_Number", &METER);
&ret = & Account_Doc.AddValue("Service_Type", &MODE);

/* add next set of values in list */

&ret = &Calc_Input.AddNextDoc();
&ret = &Account_Doc.AddValue("Account_Number", "CT-8002");
&ret = &Account_Doc.AddValue("Meter_Number", &METER);
&ret = &Account_Doc.AddValue("Service_Type", &MODE);
```

Output Structures

The following shows an example of an output structure:



Example output structure

The *root doc* is Calculate Cost(Domestic).

The *node docs* are Service_Rate and output_param2_List. Each node can contain both values and child nodes. Node docs can be further be described as either:

- *Simple node docs* have only one set of values for a single instance of a root doc. Service_Rate is a simple node doc.

Copyright © 1988-2007, Oracle. All rights reserved.

- *List node docs* can contain more than one set of values for a single instance of a root doc. output_param2_List is a list node doc.

The values in the Service_Rate node are Service_Type, Rate, and in the output_param2_List node, output_member1 and output_member2, and within the root node, output_param1.

Business Interlinks support hierarchical output structures with the following methods:

- GetOutputDocs
- GetDoc
- GetNextDoc
- GetPreviousDoc
- GetStatus
- GetValue
- GetCount
- MoveToDoc

The GetOutputDocs method returns a reference to the root doc of an output structure. From the previous example, it returns a reference to Calculate Cost(Domestic).

Use the GetDoc method to access node docs. From the previous example, you would use GetDoc to access the Service_Rate or output_param2_List node docs. If any of these nodes contain nodes, you use GetDoc to access those as well.

Use the GetValue method to retrieve the values. From the previous example, you would use GetValue to retrieve the values for output_param1, and for the Service_Rate node, to get the values Service_Type, Rate, output_member1, and output_member2. You must call GetDoc on a node before you can call GetValue for its values.

Use the GetNextDoc (or GetPreviousDoc) method to access the following:

- A reference to the next (or previous, respectively) root doc.
- If a node doc is a list, that is, can contain more than one set of values, use GetNextDoc to reference the next node doc in the list (or GetPreviousDoc to access the previous node doc in the list.)

Use the GetCount method to return either the number of docs in a list node doc, or the number of root docs. In the example, you can count the number of Calculate Cost(Domestic) nodes or the number of output_param2_list nodes.

Use the MoveToDoc method to move to a particular doc at the root level or to a list node doc. In the example, you can move to a Calculate Cost(Domestic) node or to an output_param2_list_node.

The following code example gets values for the node docs Service Rate, which is a simple node doc. It also sets the values for output_param2_List, which is a list node doc.

```
&Calc_Input = &QE_COST.GetOutputDocs("");

&Service_Rate_Doc = &Calc_Input.GetDoc("Service_Rate");
&ret = &Service_Rate_Doc.GetValue("Service_Type", "Overnight");
&ret = &Service_Rate_Doc.GetValue("Rate", "50.00");

&Out_Param_Doc = &Calc_Input.GetDoc("output_param2_List");
&ret = &Out_Param_Doc.GetValue("output_member1", "value1");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&ret = &Out_Param_Doc.GetValue("output_member2", "value2");

/* get next set of values in list */

&Account_Doc = &Out_Param_Doc.AddNextDoc();
&ret = &Out_Param_Doc.GetValue("output_member1", "value3");
&ret = &Out_Param_Doc.GetValue("output_member2", "value4");
```

Using the Incoming Business Interlink Methods and Properties

The incoming Business Interlink methods enable you to parse an XML request and build an XML response.

The incoming Business Interlink uses a set of methods and functions.

The GetContentBody Request object method converts the request content into an XML string. You can then use this string with the GetBiDoc function to create a BiDocs structure from it that is the same shape and contains the same data as the XML document contained in the XML string.

After you have the BiDocs structure containing the XML request, you can:

- Use the GetNode method to get one of the XML elements.
- Use the NodeType method to get the type of the node (element, processing instruction, comment).
- Use the NodeName property to get the name of the element.
- Use the NodeValue property to get the value of the element.

You can also use the standard BiDocs methods (such as GetDoc, GetValue) to retrieve information from this BiDocs object.

When an XML element contains attributes, the AttributeCount property gets the number of attributes, the GetAttributeName method gets the name of an attribute, and the GetAttributeValue method gets the value of an attribute.

To build an XML response, you can use the GetBiDocs function to create a blank BiDocs structure. To create the XML structure within that BiDocs, use CreateElement to create an XML tag, AddComment to add an XML comment, AddAttribute to add an attribute to an XML tag, and AddProcessInstruction to add a processing instruction (the first tag of the XML response).

Use the GenXMLString method to create an XML string from the BiDocs structure. Then you can use the Write Response method to write the response to the HTTP response string.

See Also

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

Copyright © 1988-2007, Oracle. All rights reserved.

Understanding the State of an Interlink Object

PeopleSoft Business Interlink API that are run on the application server should be stateless, that is, if you want to save information from one call of the Interlink object to the next, you have to do it yourself by writing the relevant information to the database. If you use the Execute method more than once within a single PeopleCode event (that is, if you have the Execute method in some sort of loop) the state is preserved. After you leave the event, any state associated with the Interlink object is lost.

You should create only one Business Interlink object, that is, you should only use the GetInterlink function once. After that, you can load it with data, pass the data to the Interlink plug-in (using Execute) and fetch output data as many times as you need.

Using Business Interlink with Application Engine

In a regular PeopleCode program, you can only declare a Business Interlink object as local. However, in an Application Engine program, you can declare a Business Interlink object as global. Instantiating a Business Interlink object once as a global saves on the significant overhead of reinstantiating a local object for every iteration of PeopleCode called in a loop.

- Global Business Interlink objects can only be used in Application Engine PeopleCode programs because PeopleCode that runs on an application server must be stateless.
- When a restartable Application Engine program abends, global Business Interlink objects that were instantiated before the last checkpoint are automatically reinstantiated at restart. So the object will be available, even though no call has been made to GetInterlink in the restarted process. However, the associated Business Interlink data buffers are *not* recovered, so the Application Engine program must be written such that these buffers are empty whenever a checkpoint is taken.
- Business Interlink objects should *not* be declared as global unless they are used in several PeopleCode actions, or in a PeopleCode action that is called in a loop. Only in these instances is the overhead of doing a checkpoint worthwhile.

Using the Data Type of a Business Interlink Object

You should declare a Business Interlink object using the data type Interlink. For example,

```
Local Interlink &MYBI;
```

Declare hierarchical data as type BIDocs. For example:

```
Local BIDocs &OutlistDoc;
```

Note. BIDocs and Interlink objects used in PeopleCode programs run on the application server can be declared only as type Local. You can declare Interlinks as Global only in an Application Engine program.

Understanding the Scope of a Business Interlink Object

A Business Interlink object can be instantiated from PeopleCode.

 Copyright © 1988-2007, Oracle. All rights reserved.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Business Interlink Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetBiDoc

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

Business Interlink Class Methods

In this section, we discuss the Business Interlink class methods. The methods are discussed in alphabetical order.

AddDoc

Syntax

AddDoc(*docname*)

Description

The AddDoc method adds a document to an input structure. The added document is an input parameter for a Business Interlink object that is not of simple type (such as integer or string). You must add the document to the input structure before you can add values to its members with AddValue.

Parameters

Parameter	Description
docname	The name of the document that AddDoc adds to the structure.

Returns

The document added to the BIDocs input document.

Example

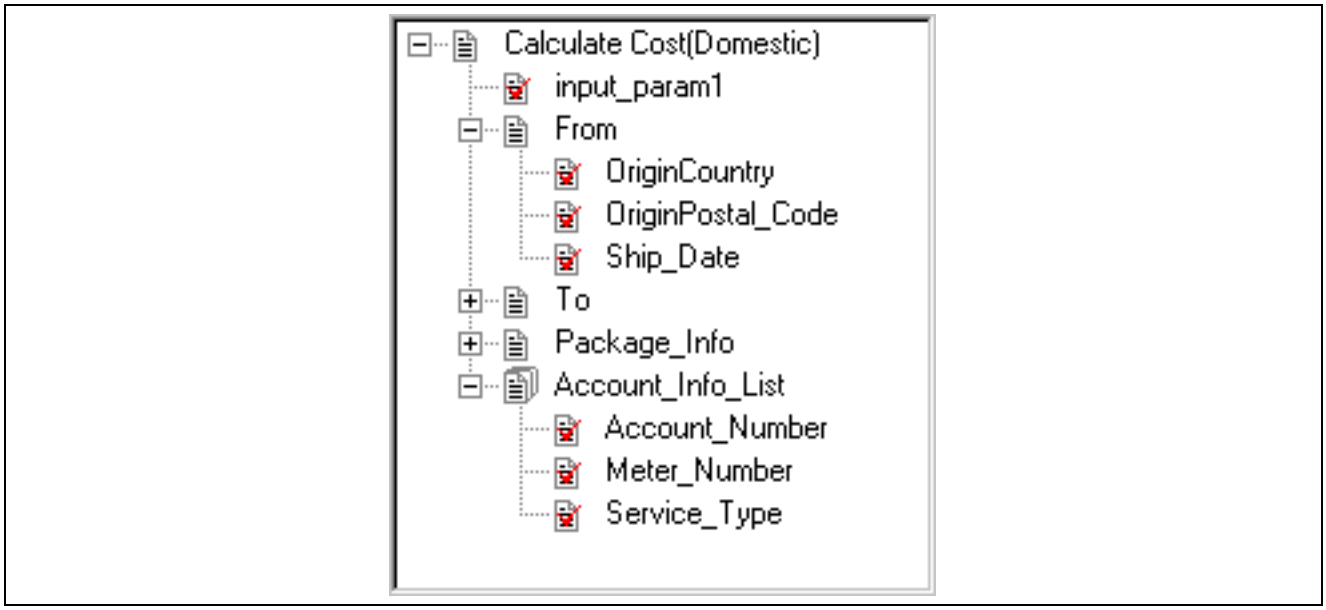
In the following example, the input structure for Calculate Cost, or the root level document, is created with the GetInputDocs method. (To create, or add, more input structures, use AddNextDoc.) The Calculate Cost input parameter From is a document, so the AddDoc method adds it to the input document.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret, &retinput;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostIn = &QE_COST.GetInputDocs("");

&ret = &CalcCostIn.AddValue("input_param1","value");
&FromDoc = &CalcCostIn.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

/* Call AddDoc and AddValue for To, Package_Info, and Account_Info_List (code not⇒
 shown) */
```

The following shows the input structure for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List. This is a version of the Federal Express plug-in that was modified for this example (input_param1 was added, Account_Info was modified to be a list).



Example input structure

See Also

Chapter 10, "Business Interlink Class," GetInputDocs, page 288, Chapter 10, "Business Interlink Class," AddValue, page 266 and Chapter 10, "Business Interlink Class," AddNextDoc, page 264.

AddInputRow

Syntax

AddInputRow(*inputname, value*)

where *inputname* and *value* are in matched pairs, in the form:

inputname1, *value1* [, *inputname2, value2*] . . .

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The AddInputRow method adds a row of input data (*value*) from PeopleCode variables or record fields for the Business Interlink object executing the method. These must be entered in matched pairs, that is, every input name must be followed by its matching value.

Note. The input *name*, not the input path, of the interface definition is used for this method.

There must be an input name for every input parameter defined in the interface definition used to instantiate the Business Interlink object.

If you specify a record field that is not part of the record the PeopleCode program is associated with, you must use *recname..fieldname* for that value.

You can specify default values for every input name in the interface definition (created in PeopleSoft Application Designer.) These values are used if you create a PeopleCode "template" by dragging the interface definition from the Project window in Application Designer to an open PeopleCode editor window.

Parameters

Parameter	Description
inputname	Specify the input name. There must be one *inputname* for every input name defined in the interface definition used to instantiate the Business Interlink object.
value	Specify the value for the input name. This can be a constant, a variable, or a record field. Each *value* must be paired with an *inputname*.

Returns

A Boolean value: True if the input values were successfully added. Otherwise, it returns False.

Example

In the following example, the Business Interlink object name is SRA_ALL_1, and the input name, such as ship_site_name, are being bound to record fields, such as QE_RP_SITENAME or VENDOR_INFO.QE_RP_PROMISEDATE, to variables like &PARTNAME, or to literals, like 10 for quantity.

```
&SRA_ALL_1.AddInputRow("ship_site_name", QE_RP_SITENAME,
   "promise_date", VENDOR_INFO.QE_RP_PROMISEDATE,
   "request_date", QE_RP_ORDERREQDATE,
   "subline_site_name", QE_RP_SITENAME,
   "quantity", 10,
   "part_name", &PARTNAME,
   "site_name", INV_LOCATIONS.QE_RP_SITENAME);
```

See Also

Chapter 10, “Business Interlink Class,” Execute, page 273

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetInterlink

Copyright © 1988-2007, Oracle. All rights reserved.

AddNextDoc

Syntax

AddNextDoc()

Description

The AddNextDoc method adds a document to one of the following levels:

- The root level of the input structure for a Business Interlink object. You create a reference to this level with the GetInputDocs method.
- When a document within the input structure is a list, AddNextDoc adds another document to the list. If you use AddNextDoc on a document that is not a list, AddNextDoc fails and returns an error.

Parameters

None.

Returns

This method returns an integer. The values are:

Value	Description
0	eNoError. The method succeeded.
1	eNO_DOCUMENT. The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED. Internal error.
4	eNOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE. You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA. You tried to retrieve data from a document that contained no data.

Example

In this example, the input structure for Calculate Cost, or the root level document, is accessed with the GetInputDocs method. The AddNextDoc method adds another BIDocs input document. The Calculate Cost input parameter Account_Info_List is a document, so the AddDoc method adds it to the BIDocs input document. Account_Info_List is also a document list, so AddNextDoc method adds another Account_Info_List document.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
```


Copyright © 1988-2007, Oracle. All rights reserved.

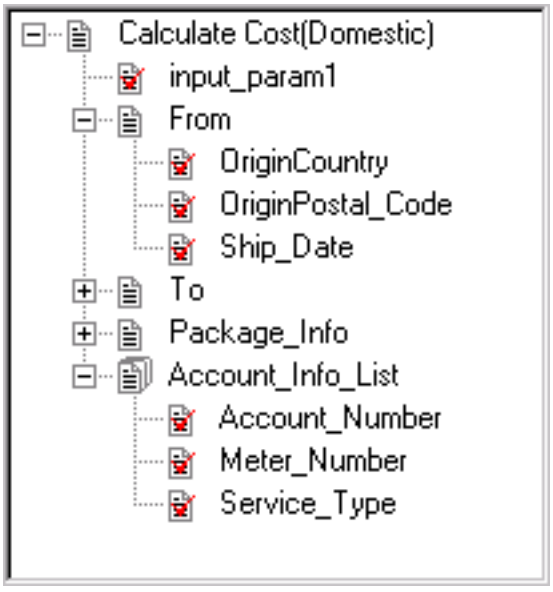
```
Local number &ret, &retinput;


&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostIn = &QE_COST.GetInputDocs("");

For &n = 1 to &number_of_input_sets
/* Get values for inputs, such as &ORIGIN (code not shown) */

   &ret = &CalcCostIn.AddValue("input_param1","value");
   &FromDoc = &CalcCostIn.AddDoc("From");
   &ret = &FromDoc.AddValue("OriginCountry", "United States");
   &ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
   &ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

   /* Call AddDoc and AddValue for To and Package_Info
      (code not shown) */

   /* Call AddDoc and AddNextDoc for the AccountDoc document list */
   &AccountDoc = &CalcCostIn.AddDoc("Account_Info_List");
   For &m = 1 to &number_of_AccountDocs
      &ret = &AccountDoc.AddValue("Account_Number", &ACCOUNT);
      &ret = &AccountDoc.AddValue("Meter_Number", &METER);
      &ret = &AccountDoc.AddValue("Service_Type", &SVCTYPE);
      &retinput = &AccountDoc.AddNextDoc();
   End-For;

   &retinput = &CalcCostIn.AddNextDoc();

End-For;
```

The following shows the input structure for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.

From, To, and Package_Info are not lists, so if you try to use AddNextDoc with these parameters, such as the following line of code, AddNextDoc will fail and return an error message.

```
&retinput = &To.AddNextDoc();
```

Copyright © 1988-2007, Oracle. All rights reserved.



Example input structure

See Also

Chapter 10, "Business Interlink Class," GetInputDocs, page 288, Chapter 10, "Business Interlink Class," AddDoc, page 261 and Chapter 10, "Business Interlink Class," AddValue, page 266.

AddValue

Syntax

AddValue(*paramname, value*)

Description

The AddValue method adds a value to a member of a document within the input structure for a Business Interlink object.

Parameters

Parameter	Description
paramname	The name of the member of the document that is having a value added to it.
value	The value that is added. This can be a constant, a variable, or a record field. The data type can be string, integer, double, float, time, date, or datetime.

Returns

This method returns an integer. The values are:

Value	Description
0	eNoError. The method succeeded.
1	eNO_DOCUMENT. The document referenced by this method does not exist.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
2	eDOCLIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED. Internal error.
4	eNOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE. You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA. You tried to retrieve data from a document that contained no data.

Example

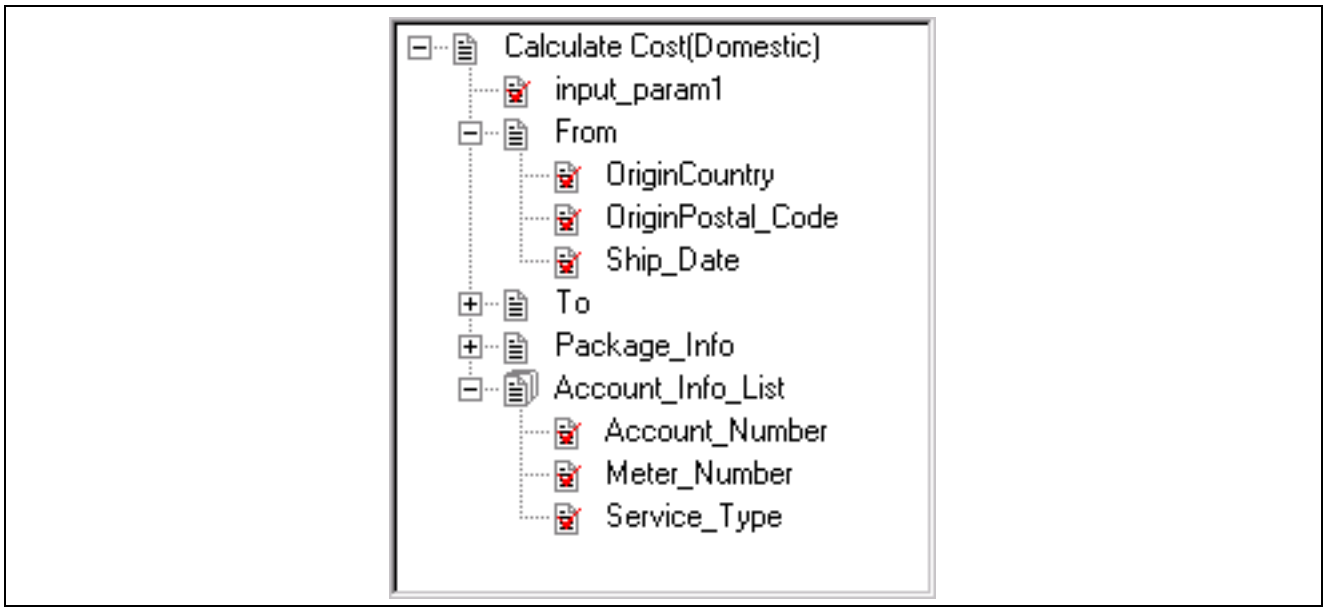
In the following example, the Business Interlink object name is QE_COST.

In the following example, the input structure for Calculate Cost, or the root level document, is accessed with the GetInputDocs method. (To create, or add, more input structures, use AddNextDoc.) The Calculate Cost input parameter From is a document, so the AddDoc method adds it to the input structure. Then the AddValue method adds values to each of the From document members.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn;
Local BIDocs &FromDoc, &ToDoc, &PackageDoc, &AccountDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

/* Get some values for input, such as &ORIGIN (code not shown) */

&CalcCostIn = &QE_COST.GetInputDocs("");
&ret = &CalcCostIn.AddValue("input_param1","value");

&FromDoc = &CalcCostIn.AddDoc("From");
&ret = &FromDoc.AddValue("OriginCountry", "United States");
&ret = &FromDoc.AddValue("OriginPostal_Code", &ORIGIN);
&ret = &FromDoc.AddValue("Ship_Date", &SHIPDATE);

/* Call AddDoc, AddValue for Package, Account_Info_List, and also call AddNextDoc⇒
 for Account_Info_List (code not shown) */
```

The following shows the input structure for this example. It contains five input parameters: input_param1, From, To, Package_Info, and Account_Info_List.

Copyright © 1988-2007, Oracle. All rights reserved.



Example input structure

See Also

Chapter 10, "Business Interlink Class," GetInputDocs, page 288, Chapter 10, "Business Interlink Class," AddDoc, page 261 and Chapter 10, "Business Interlink Class," AddNextDoc, page 264.

BulkExecute

Syntax

```
BulkExecute(RECORD.inputrecname [, RECORD.outputrecname] [, {user_
process_inst | user_operid}])
```

Description

The BulkExecute method uses the data in the specified record to populate the input buffer, copying like-named fields. Then the method executes, and, optionally, fills the record specified by outputrecname with data from the Business Interlink output buffer. BulkExecute results in significantly faster performance for transactions that process large amounts of data. Instead of adding one input row at a time, then fetching the values one at a time, you might write the data to a staging table, use the BulkExecute method and then read the data from the output table. This would be especially effective in Application Engine programs that process sets of data rather than individual rows.

This method assumes that the names of the fields in the record match the names of the inputs (or outputs) defined in the Business Interlink Definition. If there is no field in the *inputrecname* for a Business Interlink input parameter, the parameter's default value is used. If no default is specified, an empty string is passed. You must ensure that this default value is legitimate.

Fields in the output buffer are matched against the fields specified in the output record. You do not have to specify an output record. If you don't specify an output record, the output buffers are not be populated.

If you specify an output record, and if you have fields in the output record that are not specified as output parameters, they aren't populated with the BulkExecute method. In addition, they shouldn't be set as NOT NULL, otherwise inserts fail.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. Before you use this method, you should flush the record used for output and remove any residual data that might exist in it.

If you specify an output record, and you want to use the output record for error checking, you must add the following fields to your output record:

- RETURN_STATUS
- RETURN_STATUS_MSG

Note. The field names in the output record must match these names exactly.

Then you must mark one or more input parameters as "key fields" in the Business Interlink definition (using the BulkExecute ID check box.) The value of these key fields are copied to the output record, and you can use these fields to match error messages with input (or output) rows.

To order your input (and output) rows, you must add the following column to both the input and output table:

- BI_SEQ_NUM
-

Note. The field name in your input and output records must match this name exactly.

The input rows are read in the order of numbers in BI_SEQ_NUM, and the output rows are generated using the same order number.

This method automatically executes, so you don't need to use the Execute method with this method.

Whether this method halts on execution depends on the setting of the StopAtError configuration parameter. The default value is True, that is, stop if the error number returned is something other than a 1 or 2. This configuration parameter must be set before using the BulkExecute method.

Parameters

Parameter	Description
RECORD.*inputrecname*	Specify a record (SQL table) that contains the data to populate the input buffer of the Business Interlink.
RECORD .*outputrecname*	Specify a record (SQL table) that will hold the data that populates the output buffer of the Business Interlink. This is optional; it is often used to error check the input.
user_process_inst | *user_operid*	This is an optional parameter that allows either different Application Engine programs or different clients to populate the same **RECORD.** *outputrecname* at the same time.
	For *user_process_inst*, the parameter takes an integer. **RECORD.** *outputrecname* must have a PROCESS_INSTANCE field. The PROCESS_INSTANCE field is used to identify the Application Engine program that is using this Business Interlink. You can use the %PROCESS_INSTANCE variable to populate *user_process_inst*.
	For *user_operid*, the parameter takes a string. **RECORD.** *outputrecname* must have an OPERID field. The OPERID field is used to identify the client who is using this Business Interlink. You can use the %OPERID variable to populate *user_operid*.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

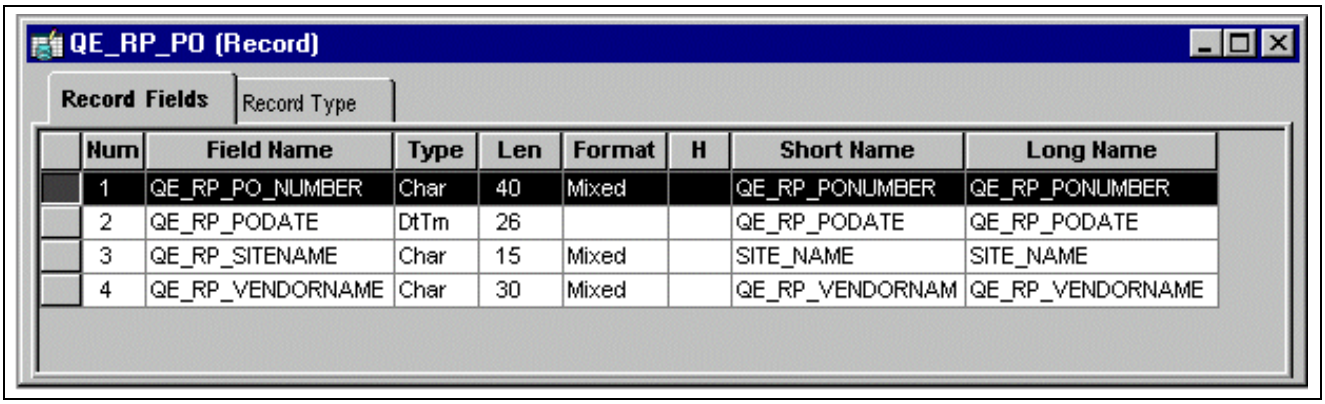
The following are the valid returns:

Value	Description
1	The Business Interlink object executes successfully
2	The Business Interlink object failed to execute
3	Transaction failed
4	Query failed
5	Missing criteria
6	Input mismatch
7	Output mismatch
8	No response from server
9	Missing parameter
10	Invalid username
11	Invalid password
12	Invalid server name
13	Connection error
14	Connection refused
15	Timeout reached
16	Unequal lists
17	No data for output
18	Output parameters empty
19	Driver not found
20	Internet connect error
21	XML parser error
22	XML deserialize

Example

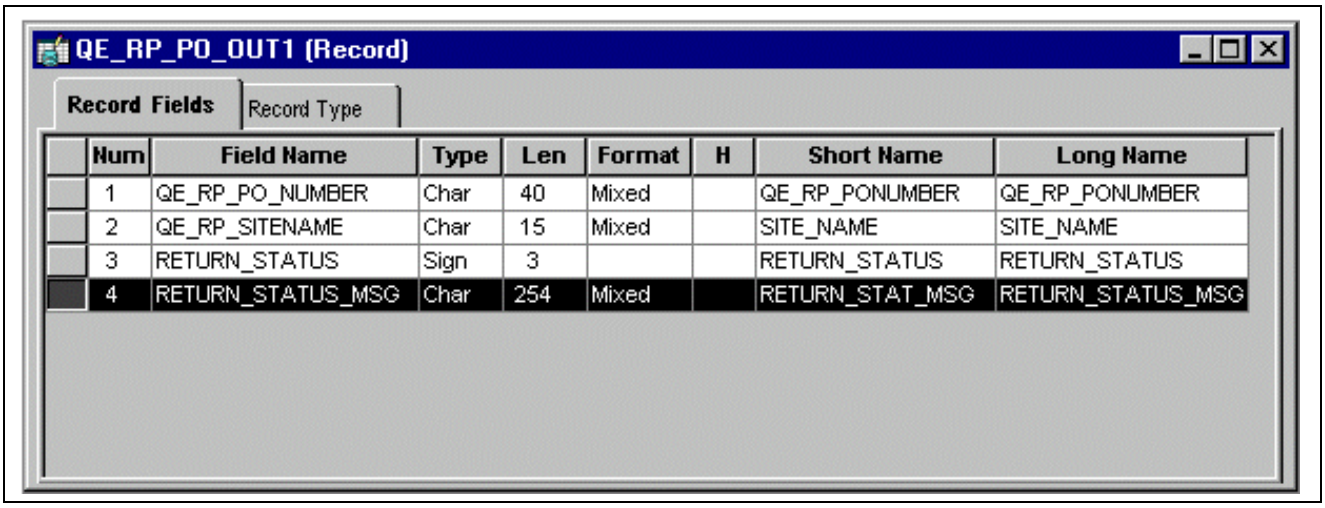
Here are two PeopleSoft records that could be used as input and output records for BulkExecute. The key fields in the input record, which need to have the BulkExecute ID check box checked in PeopleSoft Application Designer, are QE_RP_PO_NUMBER and QE_RP_SITENAME. These key fields are used both for input and output.

Copyright © 1988-2007, Oracle. All rights reserved.



Num	Field Name	Type	Len	Format	H	Short Name	Long Name
1	QE_RP_PO_NUMBER	Char	40	Mixed		QE_RP_PONUMBER	QE_RP_PONUMBER
2	QE_RP_PODATE	DtTm	26			QE_RP_PODATE	QE_RP_PODATE
3	QE_RP_SITENAME	Char	15	Mixed		SITE_NAME	SITE_NAME
4	QE_RP_VENDORNAME	Char	30	Mixed		QE_RP_VENDORNAM	QE_RP_VENDORNAME

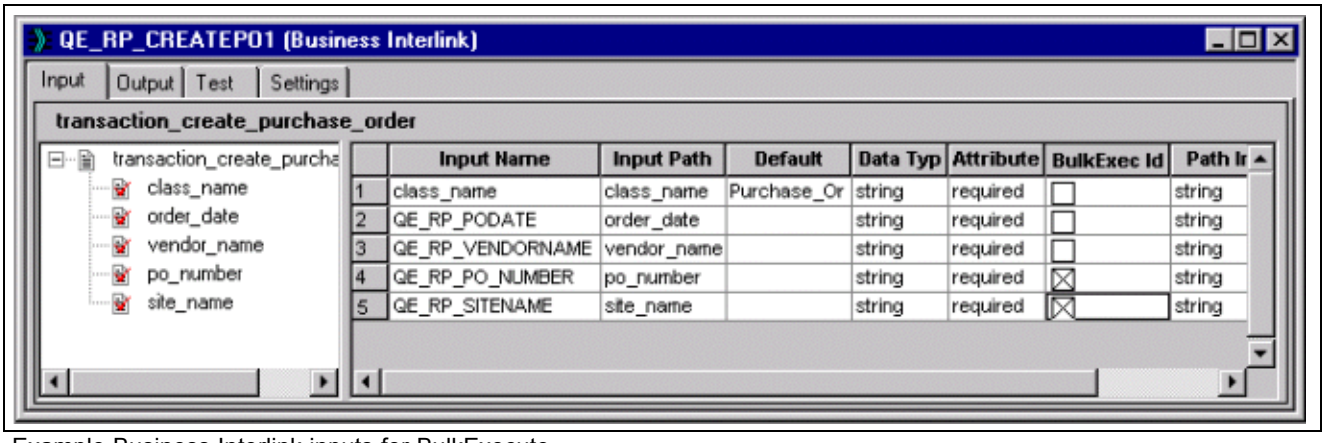
Example input record for BulkExecute



Num	Field Name	Type	Len	Format	H	Short Name	Long Name
1	QE_RP_PO_NUMBER	Char	40	Mixed		QE_RP_PONUMBER	QE_RP_PONUMBER
2	QE_RP_SITENAME	Char	15	Mixed		SITE_NAME	SITE_NAME
3	RETURN_STATUS	Sign	3			RETURN_STATUS	RETURN_STATUS
4	RETURN_STATUS_MSG	Char	254	Mixed		RETURN_STAT_MSG	RETURN_STATUS_MSG

Example output record for BulkExecute

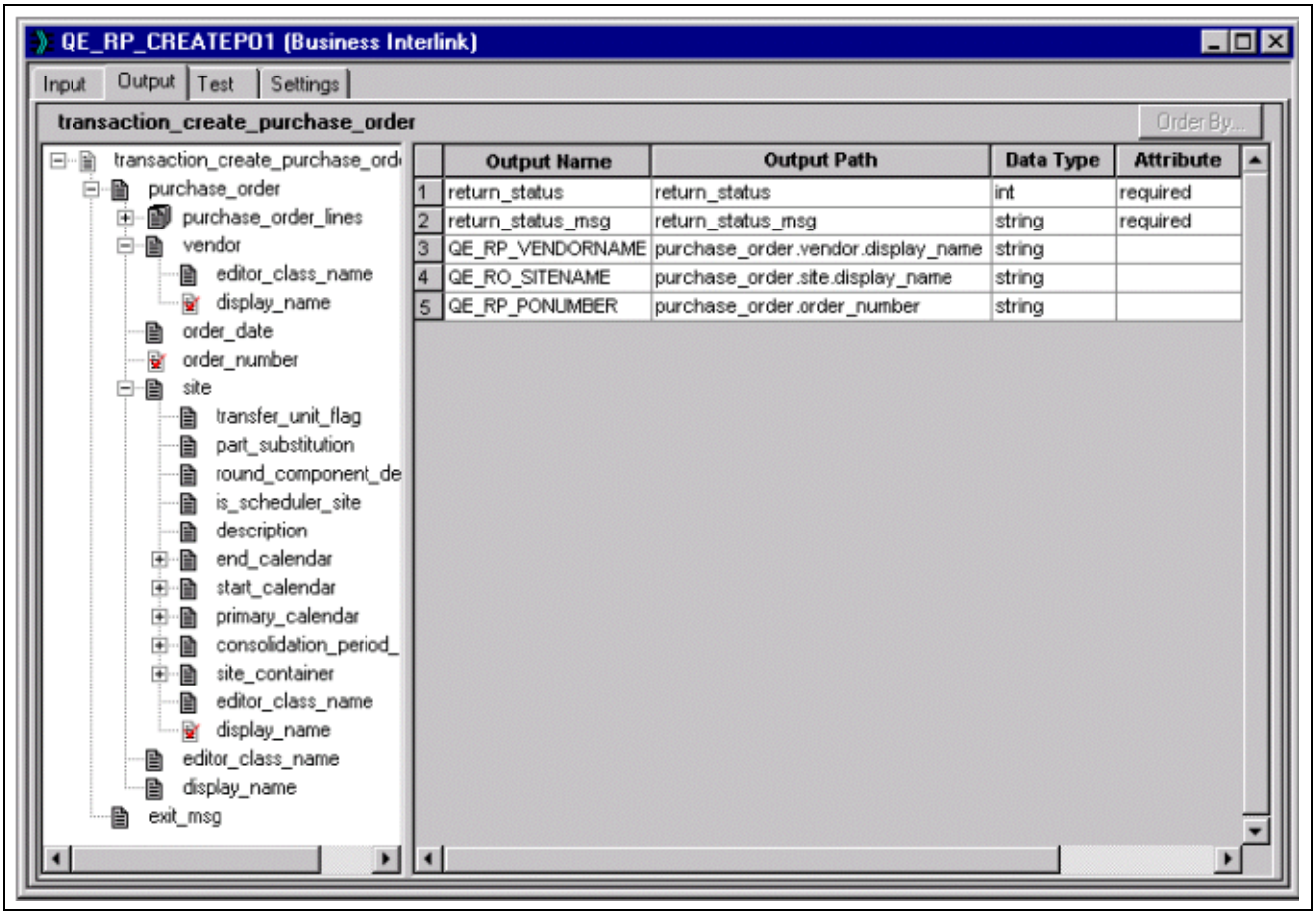
Here are the corresponding inputs and outputs for a Business Interlink that has had its inputs and outputs renamed to match the field names in the records. The inputs and outputs have been renamed where necessary to match the record field names.



	Input Name	Input Path	Default	Data Typ	Attribute	BulkExec Id	Path Ir
1	class_name	class_name	Purchase_Or	string	required	☐	string
2	QE_RP_PODATE	order_date		string	required	☐	string
3	QE_RP_VENDORNAME	vendor_name		string	required	☐	string
4	QE_RP_PO_NUMBER	po_number		string	required	☒	string
5	QE_RP_SITENAME	site_name		string	required	☒	string

Example Business Interlink inputs for BulkExecute

Copyright © 1988-2007, Oracle. All rights reserved.



Example Business Interlink inputs for BulkExecute

The PeopleCode program using these records could contain the following code to run BulkExecute.

```
Local Interlink &CREATE_PO;
Local number &EXECRSLT;

&CREATE_PO = GetInterlink(INTERLINK.QE_RP_CREATEPO1);

/* The next three lines set configuration parameters, which are not shown in the⇒
 previous examples. */
&CREATE_PO.SERVER_NAME = "bc1";
&CREATE_PO.RSERVER_HOST = "4.3.4.3";
&CREATE_PO.RSERVER_PORT = "2200";

&EXECRSLT = &CREATE_PO.BulkExecute(RECORD.QE_RP_PO, RECORD.QE_RP_PO_OUT1);
```

See Also

Chapter 10, “Business Interlink Class,” StopAtError, page 316

Clear

Syntax

Clear()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Clear method clears the input and output buffers. If you're using Business Interlinks in a batch program, every time after you use the Execute method, use the Clear method to flush out the input and output buffers.

Parameters

None.

Returns

None.

Example

```
For &n = 1 to x
   &myinterlink.AddInputRow("input1", value1, "input2", value2);
   &myinterlink.Execute();
   &myinterlink.FetchNextRow("output1", value1, "output2", value2);
      /* do processing on data */
   &myinterlink.Clear();
   &n = &n + 1;
End-for;
```

See Also

Chapter 10, "Business Interlink Class," Execute, page 273 and Chapter 10, "Business Interlink Class," BulkExecute, page 268.

Execute

Syntax

```
Execute()
```

Description

When an Interlink object executes the Execute method, the transaction associated with the Interlink object is executed.

If there is only one row, after appropriate substitution is made, the transaction is executed only once. Otherwise, the data is "batched up" and sent once. You have to call Execute only once to execute all the rows of the input buffer.

Generally, you would only use the Execute method after using the AddInputRow method. If you create a PeopleCode "template" by dragging the Business Interlink definition from the Project window in PeopleSoft Application Designer to an open PeopleCode editor window, the usual order of the execution of methods is shown in the template.

If you're using Business Interlinks in a batch program, every time after you use the Execute method, use the Clear method to flush out the input and output buffers.

Whether this method halts on execution depends on the setting of the StopAtError configuration parameter. The default value is True, that is, stop if the error number returned is something other than a 1 or 2. This configuration parameter must be set *before* using the Execute method.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

The following are the valid returns:

Value	Description
1	The Business Interlink object executes successfully
2	The Business Interlink object failed to execute
3	Transaction failed
4	Query failed
5	Missing criteria
6	Input mismatch
7	Output mismatch
8	No response from server
9	Missing parameter
10	Invalid username
11	Invalid password
12	Invalid server name
13	Connection error
14	Connection refused
15	Timeout reached
16	Unequal lists
17	No data for output
18	Output parameters empty
19	Driver not found
20	Internet connect error
21	XML parser error
22	XML deserialize

Example

```
Local number &EXECRSLT;
.  .  .
&EXECRSLT = &SRA_ALL_1.Execute();
If (&EXECRSLT <> 1) Then
/* The instance failed to execute */
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Do Error Processing */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," BulkExecute, page 268 and Chapter 10, "Business Interlink Class," Clear, page 272.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

FetchIntoRecord

Syntax

FetchIntoRecord(**RECORD.***recname*, [, {*user_process_inst* | *user_operid*}])

Description

The FetchIntoRecord method copies the data from the output buffer into the specified record (SQL table) copying *like-named* fields. This method assumes that the names of the fields in the record match the names of the outputs defined in the Business Interlink definition.

You can use the FetchIntoRecord method to perform a transaction on a large amount of data that you want to receive from your system. Instead of executing your Business Interlink and then fetching one output row at a time, you might execute your Business Interlink, then use the FetchIntoRecord method to write the data returned from the Business Interlink to a staging table.

Note. Before you use this method, you should flush the record used for output and remove any residual data that might exist in it.

If your data is hierarchical, that is, in a rowset, and you want to preserve the hierarchy, use FetchIntoRowset instead of this method.

To order your output rows, you must add the BI_SEQ_NUM column to the record.

This column is populated with a sequence number that corresponds to the order in which each row of the record was written to by the method.

Parameters

Parameter	Description
RECORD. *recname*	Specify a record (SQL table) that you want to populate with data from the output buffers.
user_process_inst | *user_operid*	This is an optional parameter that enables either different Application Engine programs or different clients to populate the same **RECORD.** *outputrecname* at the same time. For *user_process_inst*, the parameter takes an integer. **RECORD.** *outputrecname* must have a PROCESS_INSTANCE field. The PROCESS_INSTANCE field is used to identify the Application Engine program that is using this Business Interlink. You can use the %PROCESS_INSTANCE variable to populate *user_process_inst*. For *user_operid*, the parameter takes a string. **RECORD.** *outputrecname* must have an OPERID field. The OPERID field is used to identify the client who is using this Business Interlink. You can use the %OPERID variable to populate *user_operid*.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

Number of rows fetched if one or more non-empty rows are returned.

If an empty row is returned, that is, every if field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

Number	Meaning
0	No error
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor
10	CouldNotDestroySelectCursor
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError
201	FieldTooLarge

Copyright © 1988-2007, Oracle. All rights reserved.

Number	Meaning
202	IgnoreSignNumber
203	ConvertFloatToInt

Example

```
&RSLT = &QE_SM_CONCAT.FetchIntoRecord(RECORD.PERSONAL__VENDR_DATA);
If &RSLT = 0 Then
/* no rows fetched */
Else
   /* do processing */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," BulkExecute, page 268

FetchIntoRowset

Syntax

FetchIntoRowset(*&Rowset*)

Description

The FetchIntoRowset method copies the data from the output buffer into the specified rowset, copying *like-named* fields. This method assumes that the names of the fields in the rowset match the names of the outputs defined in the Business Interlink definition, and that the structure is the same.

Note. Before you use this method, you should flush the rowset used for output and remove any residual data that might exist in it.

Use this method only if you have a hierarchical data structure and you want to preserve the hierarchy. Otherwise, use BulkExecute or FetchIntoRecord.

Parameters

Parameter	Description
&Rowset	Specify rowset object that you want to populate with data from the output buffers. This must be an existing, instantiated rowset.

Returns

Number of rows fetched if one or more non-empty rows are returned.

If an empty row is returned, that is, every if field is empty except the return_status and return_status_msg fields, this method returns one of the following error messages:

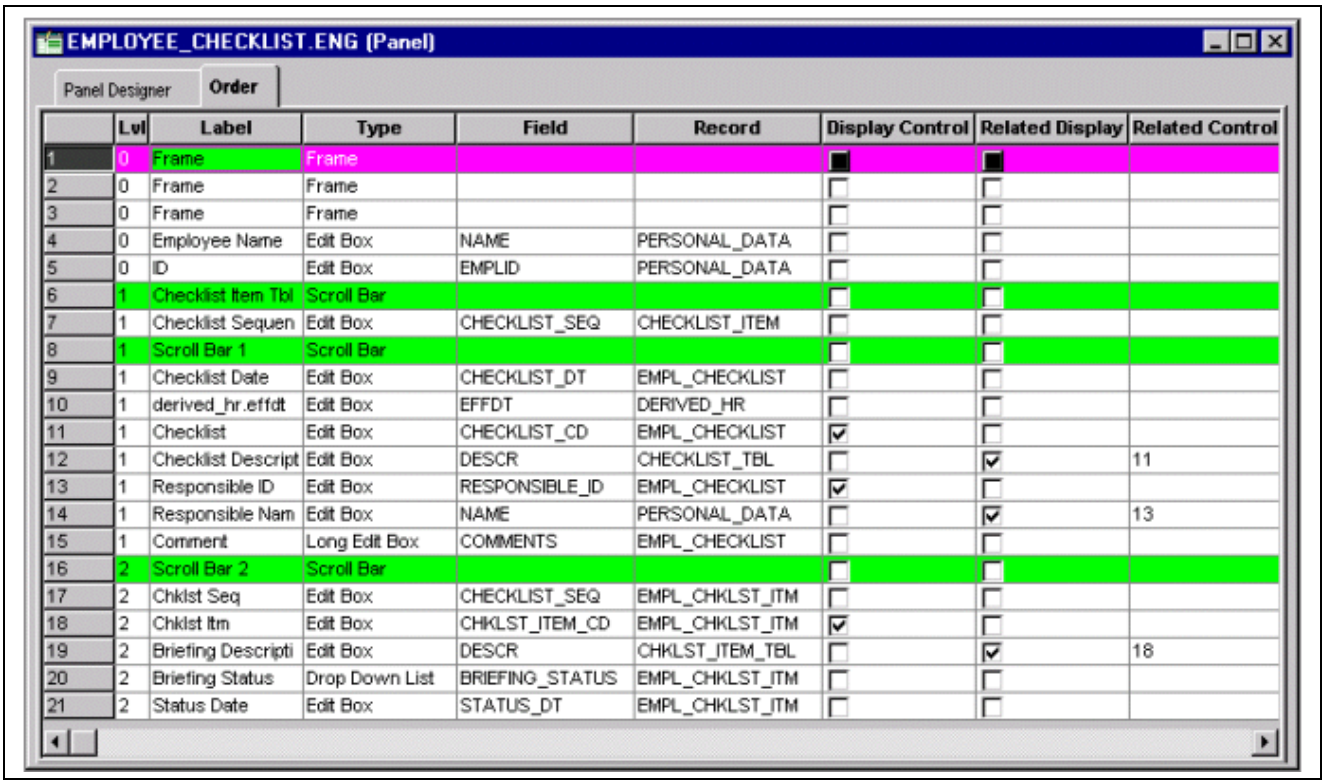
Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
0	No error
1	NoInterfaceObject
2	ParamCountError
3	IncorrectParameterType
4	NoColumnForOutputParm
5	NoColumnForInputParm
6	BulkInsertStartFailed
7	BulkInsertStopFailed
8	CouldNotCreateSelectCursor
9	CouldNotCreateInsertCursor
10	CouldNotDestroySelectCursor
11	CouldNotDestroyInsertCursor
12	InputRecordDoesNotExist
13	OutputRecordDoesNotExist
14	ContainEmptyRow
15	SQLExecError
201	FieldTooLarge
202	IgnoreSignNumber
203	ConvertFloatToInt

Example

The example uses the rowset on level one from the EMPLOYEE_CHECKLIST page. A PeopleCode program running in a field on level zero in that page can access the level one (child rowset).

 Copyright © 1988-2007, Oracle. All rights reserved.



	Lvl	Label	Type	Field	Record	Display Control	Related Display	Related Control
1	0	Frame	Frame			■	■	
2	0	Frame	Frame			☐	☐	
3	0	Frame	Frame			☐	☐	
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐	
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐	
6	1	Checklist Item Tbl	Scroll Bar			☐	☐	
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐	
8	1	Scroll Bar 1	Scroll Bar			☐	☐	
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐	
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐	
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐	
12	1	Checklist Descript	Edit Box	DESCR	CHECKLIST_TBL	☐	☑	11
13	1	Responsible ID	Edit Box	RESPONSIBLE_ID	EMPL_CHECKLIST	☑	☐	
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑	13
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐	
16	2	Scroll Bar 2	Scroll Bar			☐	☐	
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐	
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_CD	EMPL_CHKLST_ITM	☑	☐	
19	2	Briefing Descripti	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑	18
20	2	Briefing Status	Drop Down List	BRIEFING_STATUS	EMPL_CHKLST_ITM	☐	☐	
21	2	Status Date	Edit Box	STATUS_DT	EMPL_CHKLST_ITM	☐	☐	

EMPLOYEE_CHECKLIST Page order

EMPL_CHECKLIST is the primary database record for the level one scrollbar on the EMPLOYEE_CHECKLIST page. The following PeopleCode accesses the level one rowset using EMPL_CHECKLIST. The Business Interlink object name is QE_BI_EMPL_CHECKLIST. This Business Interlink object uses the level one rowset as its input and its output.

The InputRowset method uses this rowset as input for QE_BI_EMPL_CHECKLIST. Then a blank duplicate of the rowset is created with CreateRowset, and then the output of QE_BI_EMPL_CHECKLIST is fetched into the blank rowset with FetchIntoRowset.

```
&MYROWSET = GetRowset(SCROLL.EMPL_CHECKLIST);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.InputRowset(&MYROWSET);
&RSLT = &QE_BI_EMPL_CHECKLIST.Execute();
/* do some error processing */
&WorkRowset = CreateRowset(&MYROWSET);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.FetchIntoRowset(&WorkRowset);

If &ROWCOUNT = 0 Then
/* do some error processing */
Else
   /* Process the rowset from QE_BI_EMPL_CHECKLIST. Check it for errors. */
   For &I = 1 to &WorkRowset.RowCount
      For &K = 1 to &WorkRowset(&I).RecordCount
         &REC = &WorkRowset(&I).GetRecord(&K);
         &REC.ExecuteEdits();
         For &M = 1 to &REC.FieldCount
            If &REC.GetField(&M).EditError Then
            /* there are errors */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
            /* do other processing */
            End-If;
         End-For;
      End-For;
   End-for;
End-if;
```

See Also

Chapter 10, "Business Interlink Class," InputRowset, page 297

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

FetchNextRow

Syntax

FetchNextRow(*outputname, value*)

where *outputname* and *value* are in matched pairs, in the form:

outputname1, value1 [*, outputname2, value2*] . . .

Description

After the Business Interlink object executes the method Execute, the FetchNextRow method can be used to retrieve a row of output and store the values of the output name (*outputname*) *to* PeopleCode variables or record fields (*value*). These must be entered in matched pairs, that is, every output name must be followed by its matching value.

Note. The output *name,* not the output path of the interface definition is used for this method.

There must be an *outputname* for every output name defined in the interface definition used to instantiate the Business Interlink object.

If you specify a record field that is not part of the record the PeopleCode program is associated with, you must use *recname.fieldname* for that *value*.

You can specify default values for every output name in the interface definition (created in PeopleSoft Application Designer.) These values are used if you create a PeopleCode "template" by dragging the interface definition from the Project window in PeopleSoft Application Designer to an open PeopleCode Editor window.

Parameters

Parameter	Description
outputname	Specify the output name. There must be one *outputname* for every output name defined in the interface definition used to instantiate the Business Interlink object.
value	Specify the value for the output name. This can be a constant, a variable, or a record field. Each *value* must be paired with an *outputname*.

Returns

A Boolean value: True if the row of output parameters was fetched. Otherwise, it returns False.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

In the following example, the Business Interlink object name is SRA_ALL_1, and the output names, such as costs, are being bound to record fields, such as STR_COST.

```
&RSLT = &SRA_ALL_1.FetchNextRow("costs", &STR_COST,
          "unit_costs", &STR_UNIT_COST,
          "customer_ship_dates", &STR_SHIP_DATE,
          "quantities", &STR_QUANTITY,
          "so_numbers", &STR_SO_NUM,
          "so_names", &STR_SO_NAME,
          "line_numbers", &STR_LINE_NUM,
          "ship_sets", &STR_SHIP_SET,
          "customer_receipt_dates", &STR_RECPT_DATE);
```

See Also

Chapter 10, "Business Interlink Class," Execute, page 273

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetInterlink

GetCount

Syntax

GetCount(*docname*)

Description

The GetCount method returns the number of documents within a document list contained within an output structure for a Business Interlink object.

Parameters

Parameter	Description
docname	The name of the document list.

Returns

The number of documents in the list.

Example

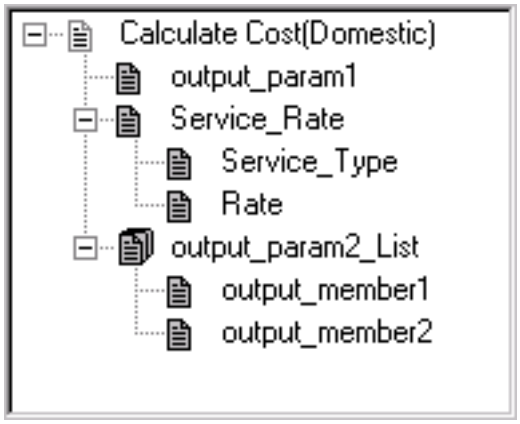
In the following example, the output structure for Calculate Cost, or the root level document, is accessed with the GetOutputDocs method. The Calculate Cost output parameter output_param2_List is a document, so the GetDoc method gets it from the output structure. Because output_param2_List is a document list, GetCount gets the number of documents in the list.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&QE_COST = GetInterlink(Interlink.QE_COST_EX);
/* Get inputs, execute. (code not shown) */

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* Call GetValue for output_param1, call GetDoc, GetValue for
Service_Rate (code not shown) */

&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");

&I = 0;
While (&I < &count)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I + 1;
      &retoutput = &OutlistDoc.GetNextDoc();
   End-If;
End-While;
```

The following shows the output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example output structure

GetDoc

Syntax

GetDoc(*docname*)

Description

The GetDoc method gets a document from an output structure. The document is an output parameter for a Business Interlink object that is not of simple type (such as integer or string). You must get the document from the output structure before you can get values from its members with GetValue.

 Copyright © 1988-2007, Oracle. All rights reserved.

You can call GetDoc using dot notation, to access documents that are "nested" within other documents. For example, the following accesses a document nested three levels deep:

```
&Docs3 = &CalcCostOut.GetDoc("Doc1.Doc2.Doc3");
```

Parameters

Parameter	Description
docname	The name of the document that GetDoc should get from the output structure.

Returns

A reference to the document received from the output structure.

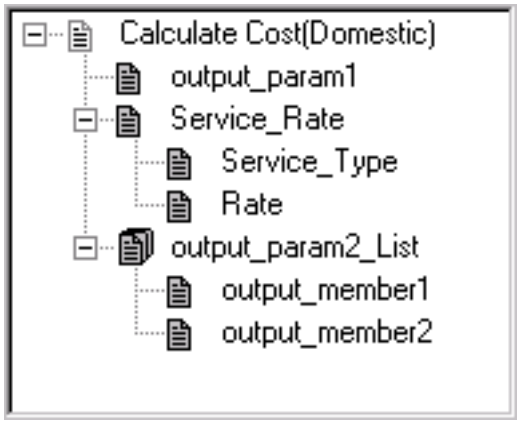
Example

In the following example, the Output structure for Calculate Cost(Domestic), or the root level document, is created with the GetOutputDocs method. (To create, or get, more Output structures, call GetNextDoc.) The Calculate Cost output parameter Service_Rate is a document, so the GetDoc method gets it from the Output structure.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &ServiceRateDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&ret = &CalcCostOut.GetValue("output_param1",&VALUE);
&ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
&ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
&ret = &ServiceRateDoc.GetValue("Rate", &RATE);
/* Call GetDoc, GetValue, GetNextDoc for output_param2_List (code not shown) */
If &ret = 0 Then
   /* Process output values */
End-If;
```

The following illustration shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Federal Express plug-in that was modified for this example (output_param1 and output_param2_List were added).

Copyright © 1988-2007, Oracle. All rights reserved.



Example output structure

In the following example, GetDoc is used to access a document that is nested more deeply. To access a document that is more deeply nested, you can either call GetDoc for each nesting, or you can call GetDoc once using the nesting feature.

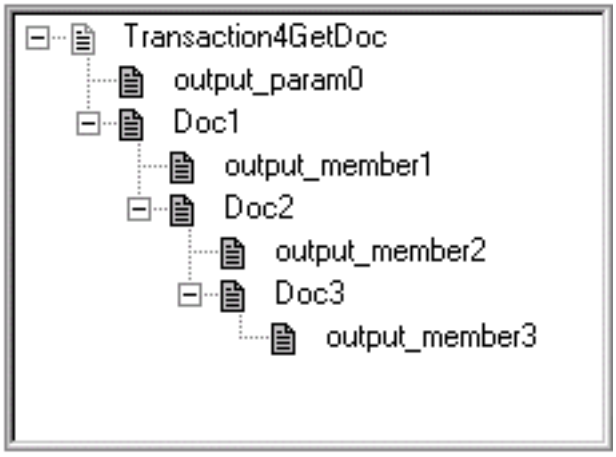
Calling GetDoc with the nesting feature:

```
Local Interlink &QE_4GETDOC;
Local BIDocs &CalcCostOut, &Docs3;
Local number &ret;

&QE_4GETDOC = GetInterlink(Interlink.QE_4GETDOC_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_4GETDOC.GetOutputDocs("");
&Docs3 = &CalcCostOut.GetDoc("Doc1.Doc2.Doc3");
&ret = &Docs3.GetValue("output_member3", &VALUE);
```

Calling GetDoc without the nesting feature:

```
Local Interlink &QE_4GETDOC;
Local BIDocs &CalcCostOut, &Docs1, &Docs2, &Docs3;
Local number &ret;

&QE_4GETDOC = GetInterlink(Interlink.QE_4GETDOC_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_4GETDOC.GetOutputDocs("");
&Docs1 = &CalcCostOut.GetDoc("Doc1");
&Docs2 = &Docs1.GetDoc("Doc2");
&Docs3 = &Docs2.GetDoc("Doc3");
&ret = &Docs3.GetValue("output_member3", &VALUE);
```

 Copyright © 1988-2007, Oracle. All rights reserved.



Example output structure with nested documents

GetFieldCount

Syntax

GetFieldCount()

Description

Use the GetFieldCount method to support dynamic output. It returns the number of columns in the output buffer, which is the same as the number of outputs in each row of the output buffer.

Note. This method can also be used with hierarchical doc data.

Parameters

None.

Returns

The number of columns in the output buffer, which is the same as the number of outputs in each row of the output buffer.

Example

The following example moves to the first column, first field of the output buffer. The Repeat loop goes through every row in the output buffer. The For loop processes every field in every row.

```
If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            /* do processing */
         .  .  .
         End-Evaluate;
      End-For;
   Until Not (&MYBI.MoveNext());
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Else
   /* do error processing - output buffer not returned */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldType, page 286, Chapter 10, "Business Interlink Class," GetFieldValue, page 287, Chapter 10, "Business Interlink Class," MoveFirst, page 299, Chapter 10, "Business Interlink Class," MoveNext, page 300 and Chapter 10, "Business Interlink Class," Using Hierarchical Data (BIDocs), page 256.

GetFieldType

Syntax

GetFieldType(*index*)

Description

Use the GetFieldType method to support dynamic output. It returns the type of field specified by *index*. You can use this method only after you have used the MoveFirst method: otherwise the system doesn't know where to start.

Note. This method can also be used with hierarchical doc data.

Parameters

Parameter	Description
index	Specify the number of the field you want to find the type of.

Returns

A number indicating the type of the field. The values are:

Value	Description
1	String
2	Integer
3	Float
4	Boolean
5	Date
6	Time
7	DateTime
8	Binary
9	Object

Copyright © 1988-2007, Oracle. All rights reserved.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row. Within the For loop, GetFieldType gets the type of the field data (string, integer, and so on.)

```
/* Add inputs to the Business Interlink object, then call Execute
to execute the Business Interlink object.
You are then ready to get the outputs using the following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 285, Chapter 10, "Business Interlink Class," GetFieldValue, page 287, Chapter 10, "Business Interlink Class," MoveFirst, page 299 and Chapter 10, "Business Interlink Class," MoveNext, page 300.

GetFieldValue

Syntax

GetFieldValue(*index*)

Description

Use the GetFieldValue method to support dynamic output. It returns the value of the field specified by index. You can use this method only after you have used the MoveFirst method: otherwise the system doesn't know where to start.

Note. This method can also be used with hierarchical doc data.

Parameters

Parameter	Description
index	Specify the number of the field you want to find the value of.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A string containing the value of the field.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row. Within the For loop, GetFieldValue gets the value of the field data.

```
/* Add inputs to the Business Interlink object, then call Execute to execute
the Business Interlink object. You are then ready to get the
outputs using the following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 285, Chapter 10, "Business Interlink Class," GetFieldType, page 286, Chapter 10, "Business Interlink Class," MoveFirst, page 299 and Chapter 10, "Business Interlink Class," MoveNext, page 300.

GetInputDocs

Syntax

GetInputDocs("")

Description

The GetInputDocs method gets the top input document at the root level for a Business Interlink object. This is a hierarchical structure that contains the values for the inputs for this Business Interlink object. The methods that you use to put the input values into this document are the hierarchical data methods.

Parameters

A null string.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

An input document. This is the document at the top of the root level of the input document for a Business Interlink object.

Example

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut, &CalcCostIn;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
&CalcCostIn = &QE_COST.GetInputDocs("");

/* You can now insert the input values and execute the BI object */
```

See Also

Chapter 10, "Business Interlink Class," Using Hierarchical Data (BIDocs), page 256

GetNextDoc

Syntax

GetNextDoc()

Description

The GetNextDoc method gets a document from one of the following levels:

- The root level of the Output structure for a Business Interlink object. This level was accessed with the GetOutputDocs method.
- When a document within the Output structure is a list, GetNextDoc gets another document from the list. If you use GetNextDoc on a document that is not a list, GetNextDoc fails and returns an error.

Parameters

None.

Returns

The following are the valid returns:

Number	Enum Value	Description
0	eNoError	The method succeeded.
1	eNO_DOCUMENT	The document referenced by this method does not exist.

Copyright © 1988-2007, Oracle. All rights reserved.

Number	Enum Value	Description
2	eDOCLIST_OUT_RANGE	You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED	Internal error
4	eNOT_DOCUMENTTYPE	You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE	You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE	You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA	You tried to retrieve data from a document that contained no data.

Example

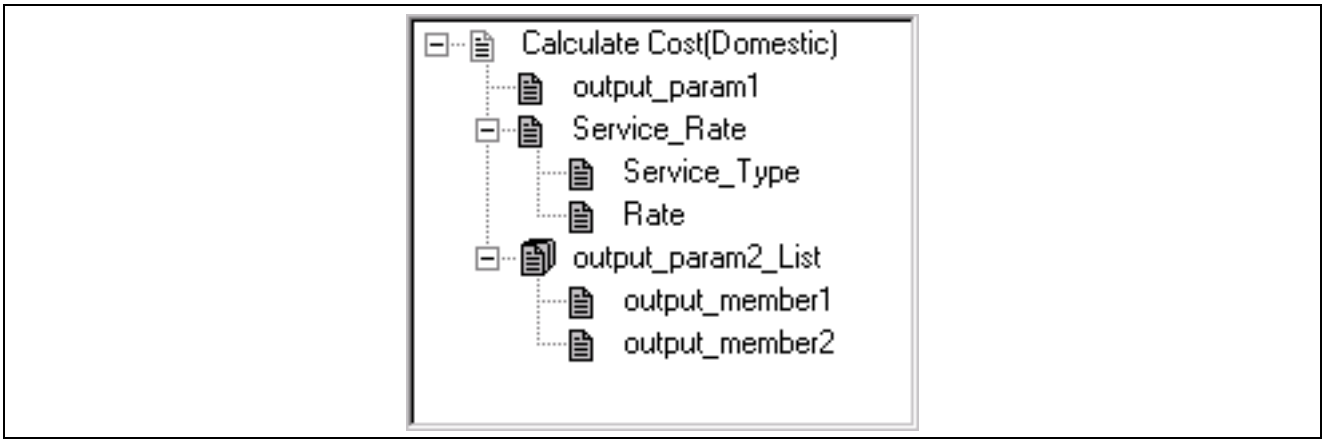
In this example, the Output structure for Calculate Cost, or the root level document, is accessed with the GetOutputDocs method. The GetNextDoc method gets another Output structure, assuming that there is more than one of them. The Calculate Cost output parameter output_param2_List is a document, so the GetDoc method gets it to the Output structure. output_param2_List is also a document list, so GetNextDoc method gets the next output_param2_List document.

```
Local Interlink &QE_COST;
Local number &count1, &count2;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret1, &ret2;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

&ret1 = 0;
While (&ret1)
   &ret1 = &CalcCostOut.GetValue("output_param1",&VALUE);
   &ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
   &ret1 = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
   &ret1 = &ServiceRateDoc.GetValue("Rate", &RATE);
   /* Process output values */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   &OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
   &count2 = &CalcCostOut.GetCount("output_param2_List");
   While (&I < &count2)
      &ret2 = &OutlistDoc.GetValue("output_member1", &VALUE1);
      &ret2 = &OutlistDoc.GetValue("output_member2", &VALUE2);
      If &ret2 = 0 Then
         /* Process output values */
         &I = &I + 1;
         &ret2 = &OutlistDoc.GetNextDoc();
      End-If;
   &ret1 = &CalcCostOut.GetNextDoc();
End-While;
```

The following shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example output structure

GetOutputDocs

Syntax

GetOutputDocs("")

Description

The GetOutputDocs method gets the top output document at the root level for a Business Interlink object. This is a hierarchical structure that contains the values for the outputs for this Business Interlink object. The methods that you use to get output values from this document are the hierarchical data methods.

Parameters

A null string.

Returns

An output document. This is the document at the top of the root level of the output document for a Business Interlink object.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostIn, &CalcCostOut;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* You can now execute the Business Interlink object and get the output values. */
```

See Also

Chapter 10, "Business Interlink Class," Using Hierarchical Data (BIDocs), page 256

GetPreviousDoc

Syntax

```
GetPreviousDoc()
```

Description

The GetPreviousDoc method gets the previous document from either the root level of the Output structure for a Business Interlink object, or from a document within the Output structure. When these documents are in a list, GetPreviousDoc gets the previous document in the list. You must get the document before you can get its values with GetValue.

Parameters

None.

Returns

The following are the possible returns:

Number	Enum Value	Description
0	eNoError	The method succeeded
1	eNO_DOCUMENT	The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE	You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED	Internal error
4	eNOT_DOCUMENTTYPE	You tried to perform an operation upon a parameter that is not a document type.

Copyright © 1988-2007, Oracle. All rights reserved.

Number	Enum Value	Description
5	eNOT_LISTTYPE	You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE	You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA	You tried to retrieve data from a document that contained no data.

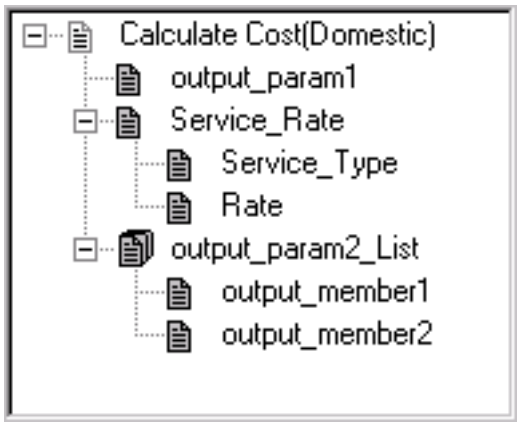
Example

In this example, the Output structure for Calculate Cost, or the root level document, is accessed with the GetOutputDocs method. It contains one output parameter, output_param2_List, which is also a list. The GetCount and MoveToDoc methods point to the last output_param2_List document in the list. The GetPreviousDoc method is used in a loop to cycle through the output_param2_List list, starting with the last and ending with the first in the list, and get each output_param2_List document and its values.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

/* Call GetValue for output_param1, call GetDoc, GetValue for Service_Rate (code⇒
 not shown) */

&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");
&ret = &OutlistDoc.MoveToDoc(&count-1);
&I = &count;
While (&I > 0)
   &ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
   &ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
   If &ret = 0 Then
      /* Process output values */
      &I = &I - 1;
      &retoutput = &OutlistDoc.GetPreviousDoc("");
   End-If;
End-While;
```

Copyright © 1988-2007, Oracle. All rights reserved.

The following shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List.



Example output structure

GetStatus

Syntax

```
GetStatus()
```

Description

The GetStatus method tests the BIDocs document. To find the status for any BIDocs method that does not return a status value, call GetStatus just after you call that BIDocs method.

Parameters

None.

Returns

The following are the possible returns:

Number	Enum Value	Description
0	NoError	The method succeeded.
1	NO_DOCUMENT	The document referenced by this method does not exist.
2	LIST_OUT_RANGE	You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	DOCUMENT_UNINITIALIZED	Internal error.

Copyright © 1988-2007, Oracle. All rights reserved.

Number	Enum Value	Description
4	NOT_DOCUMENTTYPE	You tried to perform an operation upon a parameter that is not a document type.
5	NOT_DOCUMENTLISTTYPE	You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
6	NOT_LISTTYPE	You tried to perform a list operation using GetValue, AddValue, on a non-list.
7	NOT_SINGLEBASICTYPE	You tried to perform a GetValue or AddValue upon a list that does not use a single basic type: integer, float, string, time, date, datetime.
9	NO_DATA	You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR	There was an error with the transaction.

GetValue

Syntax

GetValue(*paramname, value*)

Description

The GetValue method gets a value from an output parameter within a document of the Output structure for a Business Interlink object.

Parameters

Parameter	Description
Paramname	The name of the member of the document that is having a value retrieved from it.
Value	The value that is retrieved. This can be a variable or a record field. The data type can be string, integer, double, float, time, date, or datetime.

Returns

The following are the possible return values:

Copyright © 1988-2007, Oracle. All rights reserved.

Return Status for Integer

Number	Enum value and Meaning
0	NoError. The method succeeded.
1	NO_DOCUMENT. The document referenced by this method does not exist.
9	NO_DATA. You tried to retrieve data from a document that contained no data.
10	GENERIC_ERROR. There was an error with the transaction.

Example

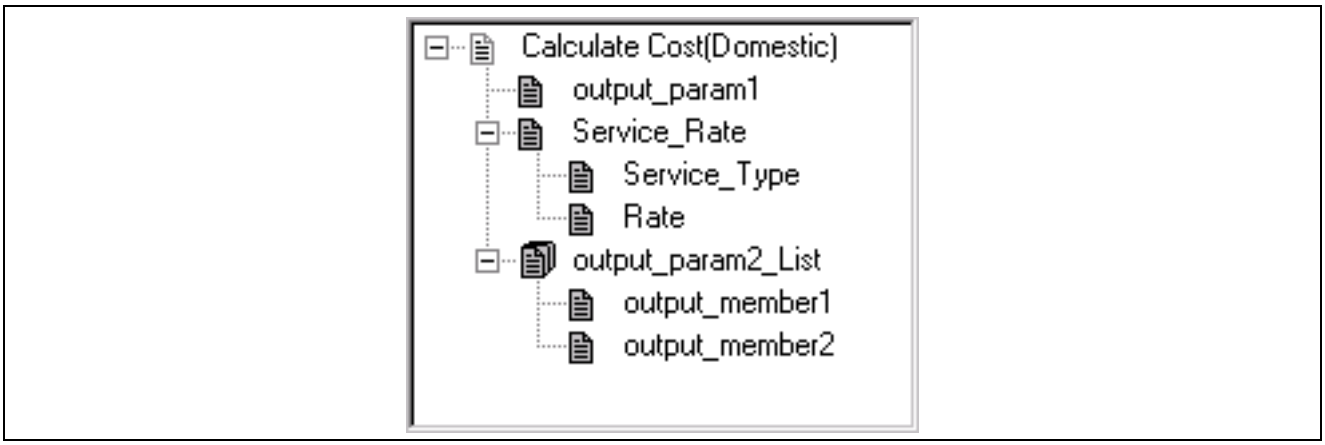
In the following example, the Business Interlink object name is QE_COST.

In the following example, the Output structure for Calculate Cost, or the root level document, is created with the GetOutputDocs method. (If you wanted to create, or get, more Output structures, you would call GetNextDoc.) The Calculate Cost output parameter Service_Rate is a document, so the GetDoc method gets it from the Output structure. Then the GetValue method gets values from each of the Service_Rate document members.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut;
Local BIDocs &ServiceRateDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&ret = &CalcCostOut.GetValue("output_param1",&PARAM1);

&ServiceRateDoc = &CalcCostOut.GetDoc("Service_Rate");
&ret = &ServiceRateDoc.GetValue("Service_Type", &SERVICE_TYPE);
&ret = &ServiceRateDoc.GetValue("Rate", &RATE);

/* Call GetDoc, GetValue, GetNextDoc for output_param2_List (code not shown) */
```

The following illustration shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Federal Express plug-in that was modified for this example (output_param1 and output_param2_List were added).

 Copyright © 1988-2007, Oracle. All rights reserved.



Example output structure

InputRowset

Syntax

InputRowset(*&Rowset*)

Description

The InputRowset method uses the data in the specified rowset to populate the input buffer, copying *like-named* fields in the appropriate structure. This method assumes that the names of the fields in the rowset match the names of the inputs defined in the Business Interlink definition, and that the structure of both rowsets are the same.

Use this method only if you have a hierarchical data structure and you want to preserve the hierarchy. Otherwise, use BulkExecute or AddInputRow.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object.

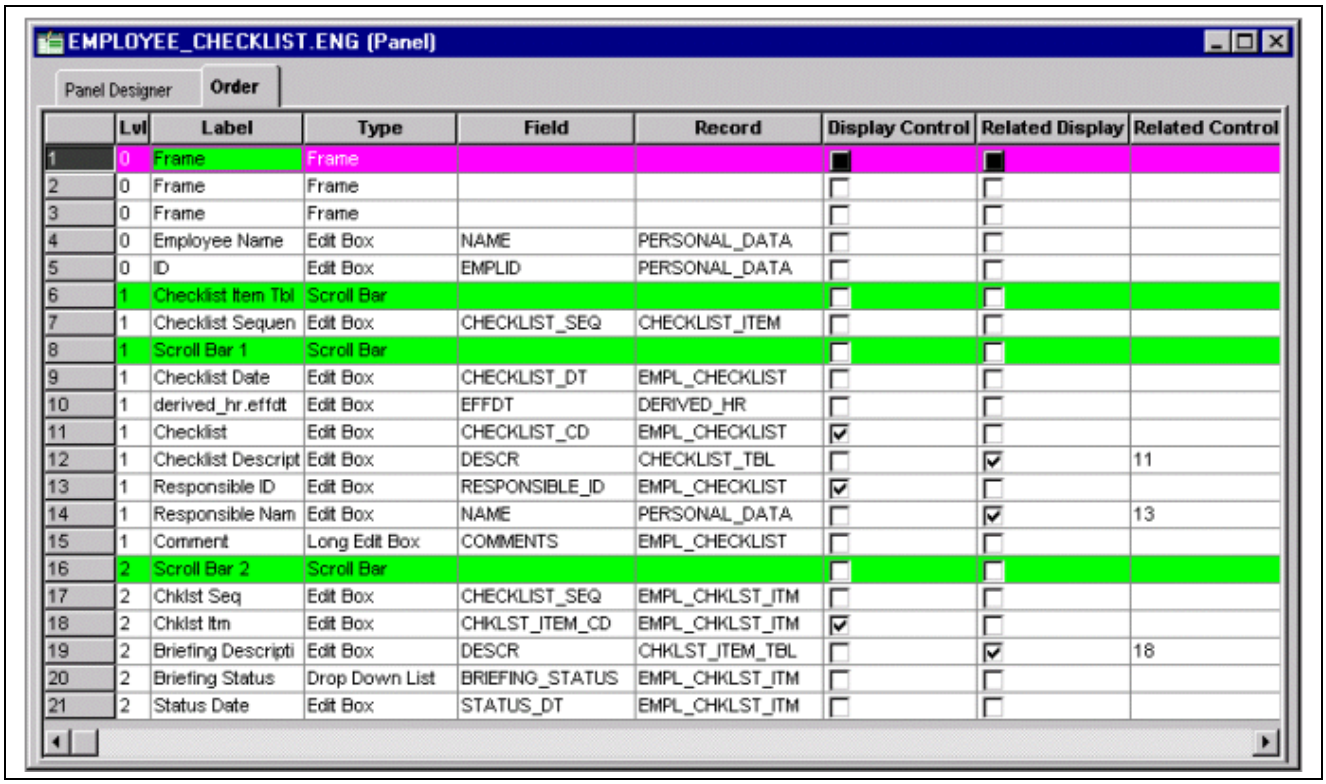
Returns

An optional value: the number of rows inserted into the output buffer.

Example

The example uses the rowset on level one from the EMPLOYEE_CHECKLIST page. A PeopleCode program running in a field on level zero in that panel can access the child rowset (level one), shown below from Scroll Bar 1 to Scroll Bar 2.

Copyright © 1988-2007, Oracle. All rights reserved.



	Lvl	Label	Type	Field	Record	Display Control	Related Display	Related Control
1	0	Frame	Frame			■	■	
2	0	Frame	Frame			☐	☐	
3	0	Frame	Frame			☐	☐	
4	0	Employee Name	Edit Box	NAME	PERSONAL_DATA	☐	☐	
5	0	ID	Edit Box	EMPLID	PERSONAL_DATA	☐	☐	
6	1	Checklist Item Tbl	Scroll Bar			☐	☐	
7	1	Checklist Sequen	Edit Box	CHECKLIST_SEQ	CHECKLIST_ITEM	☐	☐	
8	1	Scroll Bar 1	Scroll Bar			☐	☐	
9	1	Checklist Date	Edit Box	CHECKLIST_DT	EMPL_CHECKLIST	☐	☐	
10	1	derived_hr.effdt	Edit Box	EFFDT	DERIVED_HR	☐	☐	
11	1	Checklist	Edit Box	CHECKLIST_CD	EMPL_CHECKLIST	☑	☐	
12	1	Checklist Descript	Edit Box	DESCR	CHECKLIST_TBL	☐	☑	11
13	1	Responsible ID	Edit Box	RESPONSIBLE_ID	EMPL_CHECKLIST	☑	☐	
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑	13
15	1	Comment	Long Edit Box	COMMENTS	EMPL_CHECKLIST	☐	☐	
16	2	Scroll Bar 2	Scroll Bar			☐	☐	
17	2	Chklst Seq	Edit Box	CHECKLIST_SEQ	EMPL_CHKLST_ITM	☐	☐	
18	2	Chklst Itm	Edit Box	CHKLST_ITEM_CD	EMPL_CHKLST_ITM	☑	☐	
19	2	Briefing Descripti	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑	18
20	2	Briefing Status	Drop Down List	BRIEFING_STATUS	EMPL_CHKLST_ITM	☐	☐	
21	2	Status Date	Edit Box	STATUS_DT	EMPL_CHKLST_ITM	☐	☐	

EMPLOYEE_CHECKLIST Page structure

EMPL_CHECKLIST is the primary database record for the level one scrollbar on the EMPLOYEE_CHECKLIST page. The following PeopleCode access the level one rowset using EMPL_CHECKLIST. The Business Interlink object name is QE_BI_EMPL_CHECKLIST. This Business Interlink object uses the level one rowset as its input and its output.

The InputRowset method uses this rowset as input for QE_BI_EMPL_CHECKLIST. Then a blank duplicate of the rowset is created with CreateRowset, and then the output of QE_BI_EMPL_CHECKLIST is fetched into the blank rowset with FetchIntoRowset.

```
&MYROWSET = GetRowset(SCROLL.EMPL_CHECKLIST);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.InputRowset(&MYROWSET);
&RSLT = &QE_BI_EMPL_CHECKLIST.Execute();
/* do some error processing */
&WorkRowset = CreateRowset(&MYROWSET);
&ROWCOUNT = &QE_BI_EMPL_CHECKLIST.FetchIntoRowset(&WorkRowset);

If &ROWCOUNT = 0 Then
/* do some error processing */
Else
   /* Process the rowset from QE_BI_EMPL_CHECKLIST. Check it for errors. */
   For &I = 1 to &WorkRowset.RowCount
      For &K = 1 to &WorkRowset(&I).RecordCount
         &REC = &WorkRowset(&I).GetRecord(&K);
         &REC.ExecuteEdits();
         For &M = 1 to &REC.FieldCount
            If &REC.GetField(&M).EditError Then
            /* there are errors */
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
            /* do other processing */
            End-If;
         End-For;
      End-For;
   End-for;
End-if;
```

See Also

Chapter 10, "Business Interlink Class," FetchIntoRowset, page 277

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

MoveFirst

Syntax

MoveFirst()

Description

Use the MoveFirst method to support dynamic output. This method moves your cursor to the first column, first row of the output buffer. You must use this method before you try to access any data.

Parameters

None.

Returns

Boolean: True if successfully positioned cursor at the first column, first row of the output buffer, False otherwise.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row.

```
/* Add inputs to the Business Interlink object, then call Execute to execute the⇒
 Business Interlink object. You are then ready to get the outputs using the⇒
 following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Else
   /* Process error - no output buffer */
End-If;
```

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 285, Chapter 10, "Business Interlink Class," GetFieldType, page 286, Chapter 10, "Business Interlink Class," GetFieldValue, page 287 and Chapter 10, "Business Interlink Class," MoveNext, page 300.

MoveNext

Syntax

MoveNext()

Description

Use the MoveNext method to support dynamic output. This method moves your cursor to the first column of the next row of the output buffer. You can use this method only after you have used the MoveFirst method: otherwise, the system doesn't know where to start.

Parameters

None.

Returns

Boolean: True if successfully positioned cursor at the next row of the output buffer, False otherwise.

Example

In the following example, the Business Interlink object name is &MYBI. The example uses MoveFirst to move to the first row of the output buffer, and to the first column, or first field, in that row. The Repeat loop uses MoveNext to go through every row in the output buffer. The For loop processes every field in every row, using the GetFieldCount method to get the number of fields, or outputs, in the row.

```
/* Add inputs to the Business Interlink object, then call Execute to execute the⇒
 Business Interlink object. You are then ready to get the outputs using the⇒
 following code. */

If (&MYBI.MoveFirst()) Then
   Repeat
      For &I = 1 to &MYBI.GetFieldCount
         &TYPE = &MYBI.GetFieldType(&I);
         Evaluate &TYPE
         Where = 1
            &STRING_VARIABLE = &MYBI.GetFieldValue(&I);
         /* test for and process other field types */
         End-Evaluate;
      End-For;
   Until Not(&MYBI.MoveNext());
Else
   /* Process error - no output buffer */
End-If;
```


Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 10, "Business Interlink Class," GetFieldCount, page 285, Chapter 10, "Business Interlink Class," GetFieldType, page 286, Chapter 10, "Business Interlink Class," GetFieldValue, page 287 and Chapter 10, "Business Interlink Class," MoveFirst, page 299.

MoveToDoc

Syntax

MoveToDoc(*list_number*)

Description

Within a list of documents in the Output structure, the MoveToDoc method moves to the documents given by the parameter *list_number*. After using MoveToDoc, the GetValue method gets the values of the document that is in the *list_number*+1 location in the list.

Parameters

Parameter	Description
list_number	The number indicating the document that MoveToDoc moves to. After using MoveToDoc, the GetValue method gets the values of the document that is in the *list_number*+1 location in the list. For example, if *list_number* is zero, then MoveToDoc moves to the first document in the list.

Returns

This method returns an integer. The values are:

Value	Description
0	eNoError. The method succeeded.
1	eNO_DOCUMENT. The document referenced by this method does not exist.
2	eDOCLIST_OUT_RANGE. You tried to access a document in a list, but the document list is out of range. For example, if a document list contains five documents, and then you call GetDoc/GetOutputDocs once, you can call GetNextDoc four times; the fifth time results in this error.
3	eDOCUMENT_UNINITIALIZED. Internal error.
4	eNOT_DOCUMENTTYPE. You tried to perform an operation upon a parameter that is not a document type.
5	eNOT_LISTTYPE. You tried to perform a GetNextDoc or AddNextDoc upon a document that is not a list.
7	eNOT_BASICTYPE. You tried to perform a GetValue or AddValue upon a parameter that is not of basic type: integer, float, string, time, date, datetime.
8	eNO_DATA. You tried to retrieve data from a document that contained no data.

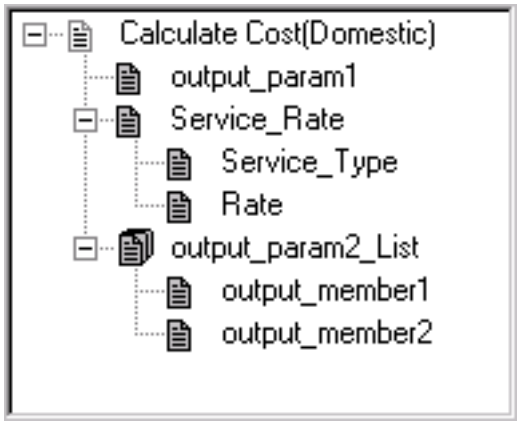
Example

The following example gets the values of the last document in the output_param2_List list. It uses GetCount to get the number of documents in the list, and then uses MoveToDoc to move to the last document in the list.

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local Interlink &QE_COST;
Local number &count;
Local BIDocs &CalcCostOut;
Local BIDocs &OutlistDoc;
Local number &ret;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);
// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");
&OutlistDoc = &CalcCostOut.GetDoc("output_param2_List");
&count = &CalcCostOut.GetCount("output_param2_List");
&ret = &OutlistDoc.MoveToDoc(&count-1);
&ret = &OutlistDoc.GetValue("output_member1", &VALUE1);
&ret = &OutlistDoc.GetValue("output_member2", &VALUE2);
```

The following illustration shows the Output structure for this example. It contains three output parameters: output_param1, Service_Rate, and output_param2_List. This is a version of the Federal Express plug-in that was modified for this example (output_param1 and output_param2_List were added).



Example output structure

ResetCursor

Syntax

ResetCursor()

Description

The ResetCursor method resets the cursor in the Output structure for a Business Interlink object to the top. After you call this method, the next time you call GetValue, you get an output value from the first document of the Output structures for a Business Interlink object.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

The following code uses ResetCursor to reset the cursor in the Output structure to the top.

```
Local Interlink &QE_COST;
Local BIDocs &CalcCostOut;

&QE_COST = GetInterlink(Interlink.QE_COST_EX);

// Get inputs, execute. (code not shown)

&CalcCostOut = &QE_COST.GetOutputDocs("");

// Perform actions on the output documents (code not shown)

&CalcCostOut.ResetCursor();
```

Incoming Business Interlink Methods

This section describes the PeopleCode methods you use with Incoming Business Interlinks.

AddAttribute

Syntax

AddAttribute(*attributename*, *attributevalue*)

Description

The AddAttribute method adds an attribute name and its value to an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
attributename	String. The name of the attribute.
attributevalue	String. The value of the attribute.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

Here is a set of XML response code.

Copyright © 1988-2007, Oracle. All rights reserved.

```
<?xml version="1.0"?>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?">
            </location>
         </user>
      </candidate>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &candidates, &location, &user;
Local number &ret;

&rootDoc = GetBiDoc("");
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&candidates = &postreqresponse.CreateElement("candidates");
&user = &candidates.CreateElement("user");
&location = &user.CreateElement("location");
&ret = &location.AddAttribute("scenery", "great");
&ret = &location.AddAttribute("density", "low");
&ret = &location.AddAttribute("blank", "eh?");
```

See Also

Chapter 10, “Business Interlink Class,” GetStatus, page 294

AddComment

Syntax

AddComment(*comment*)

Description

The AddComment method adds an XML comment after the beginning tag of an XML element referenced by a BiDoc object.

Parameters

Parameter	Description
comment	String. The comment.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

Here is a set of XML response code.

 Copyright © 1988-2007, Oracle. All rights reserved.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse, &error, &errorcode, &errortext;
Local string &blob;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 294

AddProcessInstruction

Syntax

AddProcessInstruction(*instruction*)

Description

The AddProcessInstruction method adds an XML processing instruction to a BiDocs object. Use this method at the root level of the BiDocs object for Incoming Business Interlinks before you add anything else to the BiDocs object.

Parameters

Parameter	Description
instruction	String. The processing instruction.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

See Also

Chapter 10, “Business Interlink Class,” GetStatus, page 294

AddText

Syntax

AddText(*text*)

Description

The AddText method adds text to an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
text	String. The text.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <!--this is a comment line-->
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;
&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&ret = &error.AddComment("this is a comment line");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

See Also

Chapter 10, “Business Interlink Class,” GetStatus, page 294

CreateElement

Syntax

CreateElement(*elementname*)

Description

The CreateElement method creates an XML element with the given name within a BiDoc object.

Parameters

Parameter	Description
elementname	String. The XML element name.

Returns

BiDocs. The reference to the created element.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
   <postreqresponse>
      <error>
         <errorcode>1</errorcode>
         <errortext></errortext>
      </error>
   </postreqresponse>
```

Here is the PeopleCode that builds it.

```
Local BIDocs &rootDoc, &postreqresponse;
Local BIDocs &error, &errorcode, &errortext;
Local number &ret;

&rootDoc = GetBiDoc("");

/* add a processing instruction*/
&ret = &rootDoc.AddProcessInstruction("<?xml version=""1.0""?>");
/* create an element and add text*/
&postreqresponse = &rootDoc.CreateElement("postreqresponse");
&error = &postreqresponse.CreateElement("error");
&errorcode = &error.CreateElement("errorcode");
&ret = &errorcode.AddText("1");
&errortext = &error.CreateElement("errortext");
```

GenXMLString

Syntax

GenXMLString()

Description

The GenXMLString method creates an XML string from a BiDocs object. The BiDocs object must contain the shape and data needed for an XML string. This is part of the Incoming Business Interlink functionality, which enables PeopleCode to receive an XML request and return an XML response.

Parameters

None.

Returns

String. The XML string containing the shape and data of the BiDocs object. For example, you can use this method to create an XML string containing an XML response.

Example

The following example takes a BiDocs structure that contains an XML response and puts that into a text string. After this is done, the %Response.Write function can send this as an XML response.

```
Local BIDocs &rootDoc;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local string &xmlString;

/* Create a BiDoc structure containing the data and shape of your XML response ⇒
(code not shown) */

/* Generate the XML string containing the data and shape of your XML response from⇒
 the BiDoc structure */
&xmlString = &rootDoc.GenXMLString();
%Response.Write(&xmlString);
```

GetAttributeName

Syntax

GetAttributeName(*attributenumber*)

Description

The GetAttributeName method gets the name of an attribute within an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
attributenumber	Number. The index number of the attribute.

Returns

String. The name of the attribute.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the name of the second attribute in the location node. &attrName is density.

```
Local BIDocs &rootInDoc, postreqDoc, &locationDoc;
Local string &blob, &attrName;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&attrName = &locationDoc.GetAttributeName(2);
```

GetAttributeValue

Syntax

GetAttributeValue({*attributename* | *attributenumber*})

Description

The GetAttributeValue method gets the value of an attribute within an XML element referenced by a BiDocs object.

Parameters

Parameter	Description
attributenumber | *attributename*	Specify the attribute that you want to get the value for. You can specify either the attribute number (1 for the first attribute, 2 for the second, and so on) or the attribute name (an XML tag.)

Returns

String. The value of the attribute.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the value of the second attribute in the location node. &attrValue is low.

```
Local BIDocs &rootInDoc, &postreqDoc, &locationDoc;
Local string &blob, &attrValue;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&attrValue = &locationDoc.GetAttributeValue(2);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

GetNode

Syntax

GetNode({*nodename* | *nodenumber*})

Description

The GetNode method returns a BiDocs reference to a child XML node (element or comment). Use the GetNode method upon a BiDocs reference to an XML element to access the child nodes for that element.

Parameters

Parameter	Description
nodenumber | *nodename*	Specify the node that you want to reference. You can specify either a node number (the first node is 1, the second 2, and so on) or a node name (that is, the XML tag.)

Returns

BiDocs. The returned XML element within a BiDocs object.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the postreqDoc element and the projtitle element.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&projtitleDoc = &postreqDoc.GetNode("projtitle");
```

Here is the PeopleCode that gets the postreqDoc element, the projtitle element, and the third projsubtitle element.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc, &projsubtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&projtitleDoc = &postreqDoc.GetNode(2);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&projsubtitleDoc = &projtitleDoc.GetNode(3);
```

To parse a list of elements like <projsubtitle>, where elements have the same name, you must call GetNode using an index number rather than the element name.

ParseXMLString

Syntax

ParseXMLString(*XMLstring* [, *DTDValidation*])

Description

The ParseXMLString methods fills an existing BiDocs object with the data and shape from an XML string. This is part of the Incoming Business Interlink functionality, which enables PeopleCode to receive an XML request and return an XML response.

Parameters

Parameter	Description
XMLstring	A string containing an XML document.
DTDValidation	Specify whether to validate a document type definition (DTD.) This parameter takes a boolean value. If you specify true, the DTD validation occurs if a DTD is provided. If you specify false, and if a DTD is provided, it is ignored and the XML isn't validated against the DTD. The default value for this parameter is false. In the case of application messaging, if a DTD is provided, it's always ignored and the XML isn't validated against the DTD. If the XML cannot be validated against a DTD, an error is thrown saying that there was an XML parse error.

Returns

Number. The return status. NoError, or 0, means the method succeeded.

Example

The following example gets an XML request, creates an empty BiDoc, and then fills the BiDoc with the data and shape contained in the XML string. After this is done, the GetDoc method and the GetValue method can get the value of the skills XML element, which is contained within the postreq element in the XML request.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the incoming xml(request)- Create a BiDoc and fill with the request*/
&rootInDoc = GetBiDoc("");
&ret = &rootInDoc.ParseXMLString(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 10, "Business Interlink Class," GetStatus, page 294

Incoming Business Interlink Properties

This section describes the PeopleCode properties you use with Incoming Business Interlinks.

AttributeCount

Description

The AttributeCount property gets the number of attributes within an XML element referenced by a BiDocs object.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <location scenery="great" density="low" blank="eh?">
         <city>San Rafael</city>
         <state>CA</state>
         <zip>94522</zip>
         <country>US</country>
      </location>
   </postreq>
```

Here is the PeopleCode that gets the number of attributes in the location XML element. &count should be 3, for scenery, density, and blank.

```
Local BIDocs &rootInDoc, &postreqDoc, &locationDoc;
Local string &blob;
Local number &count;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&locationDoc = &postreqDoc.GetNode("location");
&count = &locationDoc.AttributeCount;
```

ChildNodeCount

Description

The ChildNodeCount property returns the number of XML child nodes within the element referenced by the BiDocs object. Child nodes include XML elements, comments, and processing instructions.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <!--this is a comment line-->
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the XML code that gets the number of nodes within <postreq> and <projtitle>. &count1 is 2, for <email> and <projtitle>, and &count2 is 4, for the three <projsubtitle> nodes and the comment node.

```
Local BIDocs &rootInDoc, &projtitleDoc;
Local string &blob;
Local number &count1, &count2;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode("postreq");
&count1 = &postreqDoc.ChildNodeCount;
&projtitleDoc = &postreqDoc.GetNode("projtitle");
&count2 = &projtitleDoc.ChildNodeCount;
```

NodeName

Description

The NodeName property gets the name of an XML element referenced by a BiDocs object. Use this to get the name of an XML element when you used GetNode with an index number to retrieve it (meaning that you did not have the name of the XML element when you used GetNode).

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Here is the PeopleCode that gets the name of the <email> element, email.

```
Local BIDocs &rootInDoc, &postreqDoc, &emailDoc;
Local string &emailName, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&emailDoc = &postreqDoc.GetNode(1);
&emailName = &emailDoc.NodeName;
```

NodeType

Description

The NodeType property returns the type of an XML tag within a BiDocs object as an integer. The values are:

Value	Description
1	Element (a normal XML tag)
7	Processing instruction
8	Comment

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <!--this is a comment-->
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets types: &xmlprocType is 7 for processing instruction, postreqDoc is 1 for element, and commentType is 8 for comment.

```
Local BIDocs &rootInDoc, &postreqDoc, &commentDoc;
Local number &xmlprocType, &postreqType, &commentDoc;
Local string &blob;
&blob = %Request.GetContentBody();

/* <?xml version="1.0"?> */
&rootInDoc = GetBiDoc(&blob);
&xmlprocType = &rootInDoc.NodeType;

/* <postreq> */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&postreqDoc = &rootInDoc.GetNode(1);
&postreqType = &postreqDoc.NodeType;

/* <!--this is a comment--> */
&commentDoc = &postreqDoc.GetNode(2);
&commentType = &commentDoc.NodeType;
```

NodeValue

Description

The NodeValue property returns the value of a node within an XML document as a string.

This property is read-only.

Example

Here is a set of XML request code.

```
<?xml version="1.0"?>
   <postreq>
      <email>joe_blow@peoplesoft.com</email>
      <projtitle>
         <projsubtitle>first_subtitle</projsubtitle>
         <projsubtitle>second_subtitle</projsubtitle>
         <projsubtitle>third_subtitle</projsubtitle>
      </projtitle>
   </postreq>
```

Here is the PeopleCode that gets the value of the third <projsubtitle> element, third_subtitle.

```
Local BIDocs &rootInDoc, &postreqDoc, &projtitleDoc, &projsubtitleDoc;
Local string &name, &blob;
&blob = %Request.GetContentBody();

&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetNode(1);
&projtitleDoc = &postreqDoc.GetNode(2);
&projsubtitleDoc = &projtitleDoc.GetNode(3);
&projsubtitleName = &projsubtitleDoc.NodeValue;
```

Business Interlink Class Property

This section explains the StopAtError property.

StopAtError

Description

The StopAtError property specifies whether the execution of the PeopleCode program stops when there's an error, or if the PeopleCode program tries to capture the errors.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property takes a Boolean value: True to halt execution of your PeopleCode program at an error, False to continue executing. The default value is True.

This property is read-write.

Example

```
&QE_RP_SRAALL_1.StopAtError = False;
```

Configuration Parameters

There are two types of configuration parameters: the ones defined by the Interlink plug-in the Business Interlink definition is based on, and the ones that are standard, that is, defined for every Business Interlink object.

All configuration parameters must be set before you add any data to the input buffers.

In this section, we discuss the following parameters:

- Interlink plug-in configuration parameters
- URL configuration parameter
- BIDocValidate configuration parameter

Interlink Plug-in Configuration Parameters

The configuration parameters defined by the Interlink plug-in are accessed as if they were properties on the Business Interlink object. That is, in PeopleCode, you assign the value of a configuration parameter by using the Business Interlink object followed by a dot and the configuration parameter, in this format:

```
&MYINTERLINK.parameter = value;
```

You can also return the value of a configuration parameter:

```
&MYVALUE = &MYINTERLINK.parameter;
```

Each Business Interlink plug-in has its own set of configuration parameters. For example, the email project uses configuration parameters of SMTP_MAIL_SERVER, POP3MAIL_SERVER, LOGIN_NAME, PASSWORD, SENDERS_EMAIL_ADDRESS and REPLY_EMAIL_ADDRESS, while the Red Pepper transaction driver uses SERVER_NAME, RSERVER_HOST, RSERVER_PORT, and TIMEOUT.

You can specify default values for every configuration parameter in the Business Interlink definition (created in PeopleSoft Application Designer.) These values are used if you create a PeopleCode "template" by dragging the Business Interlink definition from the Project window in PeopleSoft Application Designer to an open PeopleCode editor window.

In the following example, the Interlink object name is QE_RP_SRAALL_1, and the driver is the Red Pepper driver:

```
&QE_RP_SRAALL_1.SERVER_NAME = "server";
&QE_RP_SRAALL_1.RSERVER_HOST = "pt-sun02.peoplesoft.com";
&QE_RP_SRAALL_1.RSERVER_PORT = "9884";
&QE_RP_SRAALL_1.TIMEOUT = 999;
&QE_RP_SRAALL_1.URL = "HTTP://www.PeopleSoft.com";
&QE_RP_SRAALL_1.StopAtError = False;
```

Copyright © 1988-2007, Oracle. All rights reserved.

URL Configuration Parameter

Specifies the location and name of the Business Interlink plug-in to be used to connect to the external system. This configuration parameter takes a string value.

If the plug-in is located on a web server, you have to specify the name of the web server.

If you specify a file, you can specify either a relative or an absolute path:

- If you specify an absolute path, you must specify the drive letter:

```
&MYBI.URL = "File://D:\TEMP.MYDLL";
```

- If you specify a relative path, just use the file name:

```
&MYBI.URL = "File://TEMP.MYDLL";
```

If you specify a relative path, the system firsts looks for the file in the Location directory (specified by the user when the Business Interlink was first created), then it looks in the directory where PeopleTools is installed, in the PSTOOLS/Interface Drivers directory.

BIDocValidate Configuration Parameter

Specifies whether the system should verify whether the hierarchical data object (BIDoc) exists before adding or getting values from it. This configuration parameter takes a Boolean value: True if the system should verify before accessing the object, False otherwise.

The default value is True.

If this configuration parameter is specified as True, and the object specified doesn't exist, the PeopleCode program halts execution and an error is displayed.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 11

Chart Class

This chapter provides an overview of the PeopleSoft Chart class and discusses the following topics:

- Chart terms.
- Chart types.
- Style sheets and charts.
- Font considerations for extended languages.
- Component processor considerations.
- Translation considerations.
- Label considerations.
- Bar chart considerations.
- iScripts and charts.
- Bubble chart considerations.
- Line chart considerations.
- Data series.
- Gantt charts.
- Error handling.
- Data type of a chart object.
- Scope of a chart object.
- Chart class reference.
- Chart class built-in functions.

Understanding the Chart Class

The PeopleSoft Chart class enables you to plot data in charts according to your specifications. You can create a chart by setting data using PeopleCode. You can create a chart based on a chart control on a page, or at runtime using an Internet Script (iScript).

Possible uses for the chart control include:

- charting dynamic data, such as current prices of selected commodities, from a rowset.
- plotting stored data, such as product prices, for graphic analysis of trends, from a record.

The chart class supports the following features:

Copyright © 1988-2007, Oracle. All rights reserved.

- Three-dimensional representation.
- Support for all major chart types.
- Control over the various elements of the chart such as the axes, legend, and titles.
- Interactive charts. The user can click on the different data elements of the chart to trigger further processing.

You can specify a rowset and have it graphed using a chart object with minimal PeopleCode. Within this rowset there must be a column containing data for the X axis and one for the Y axis. If more than one series of data is used, an additional column should indicate the series values. Color and popup text for each point could also be set in this rowset using additional columns.

The following distinguishable types of data can be involved in a chart:

- Primary data.
- Secondary or overlay data.

The overlay data may be quite different to the primary data so an additional rowset is needed.

There is only one object used with the chart class, that is, a chart object. There are no subobjects. All methods and properties are used with the chart object. You do not have to instantiate a subobject to access its properties and methods.

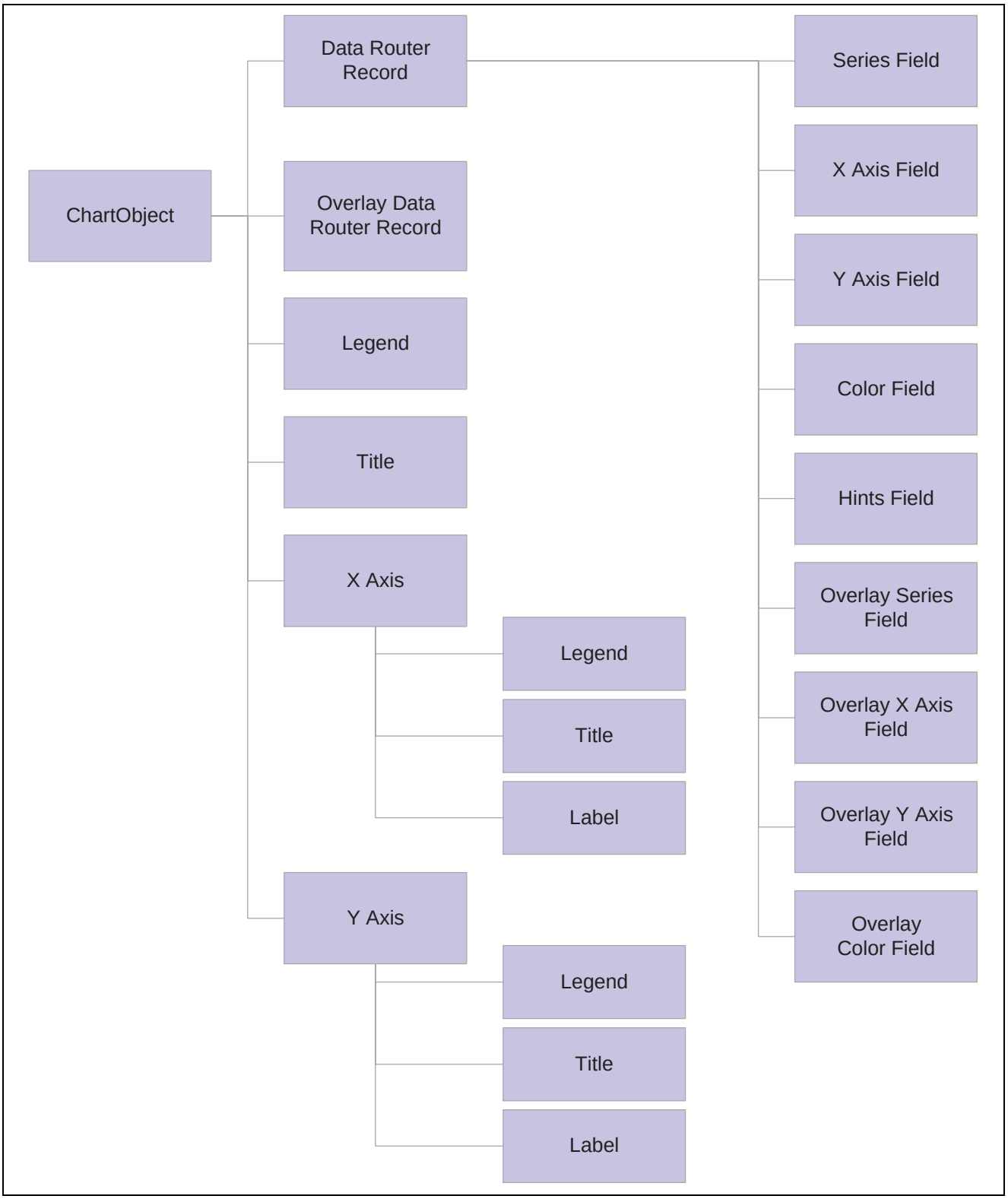
However, for a conceptual overview, it's easier to view the chart object in pieces. Just remember that these pieces do *not* represent subobjects.

A chart contains the following major parts:

- X Axis
- Y Axis
- Overlay
- Legend
- Data
- Title

The X, Y, and Overlay axes have access to titles and labels associated with their data. The following is a conceptual overview of the chart class.

 Copyright © 1988-2007, Oracle. All rights reserved.



Overview of chart class

Copyright © 1988-2007, Oracle. All rights reserved.

Chart Terms

Charts can be composed of many different elements, such as a legend, a title, a label, and so on. The following is a list of general chart terms and their description. Specific chart terminology (such as for Gantt charts) is found in those specific sections:

Series A group of related information. For example, if you were tracking sales for several divisions over many years, each division could be a series. Generally, every series in a chart has a distinct color. You can have more than one series in a chart for comparison.

Label Text that identifies the data on one of the axis.

Legend Text that identifies the different series in the chart.

Overlay Related data represented by a line drawn over the background chart. You can also specify a data series, with different line types for each overlay.

Title There are three titles: Main, X axis, and Y axis. These are text that identifies that portion of the chart.

TrueXY A type of line chart that uses a numeric X axis, in stead of a categorical X axis. The numeric X axis supports non-uniform X data, that is, each series need not have the same number of points and those points along the X axis need not match up across the series.

X axis The axis that data is measured against. For a Gantt chart, the X axis is the display of dates which items in the Y axis span.

Y axis The axis that contains the data. In most charts, this is the vertical axis. In a horizontal bar chart, this is the horizontal axis. For a Gantt chart, the Y axis is the display of tasks or activities, and is displayed as rows.

See Also

Chapter 11, “Chart Class,” Gantt Charts, page 338

Chapter 11, “Chart Class,” Understanding Gantt Chart Terms, page 338

Chart Types

The following chart types are available for PeopleSoft Chart:

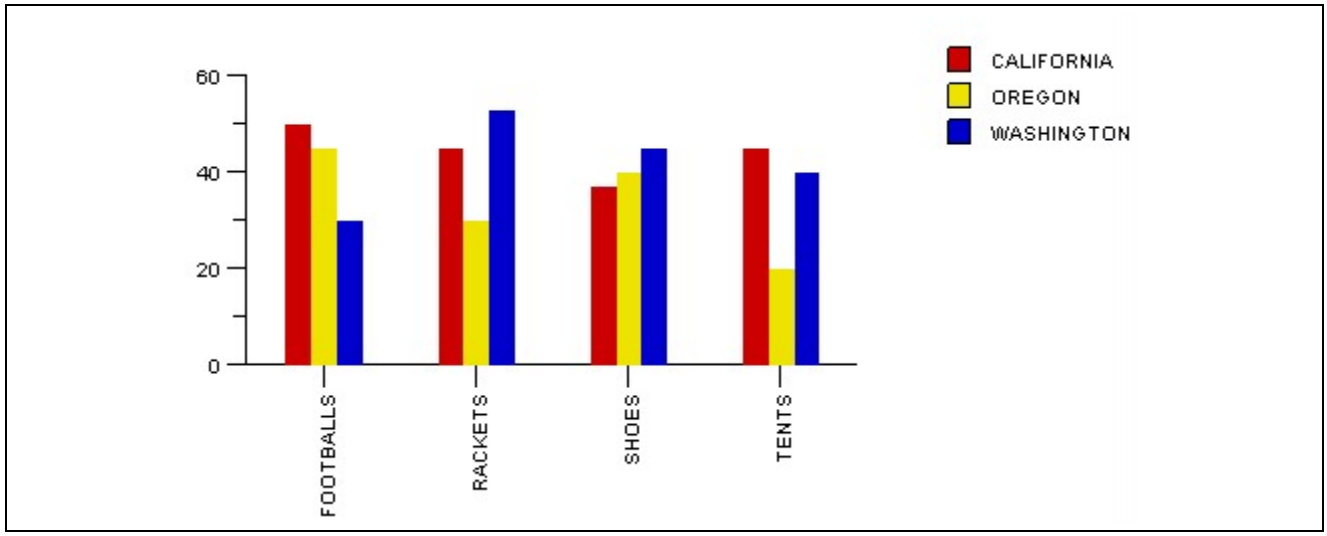
- 2D Bar.
- 2D Histogram.
- 2D Horizontal Bar.
- 2D Horizontal Percent Bar.
- 2D Horizontal Stacked Bar.
- 2D Line.

 Copyright © 1988-2007, Oracle. All rights reserved.

- 2D Percent Bar.
- 2D Pie.
- 2D Scatter.
- 2D Stacked Bar.
- 3D Bar.
- 3D Percent Bar.
- 3D Pie.
- 3D Stacked Bar.
- Gantt.

2D Bar

The following is an example of a two-dimensional bar chart.

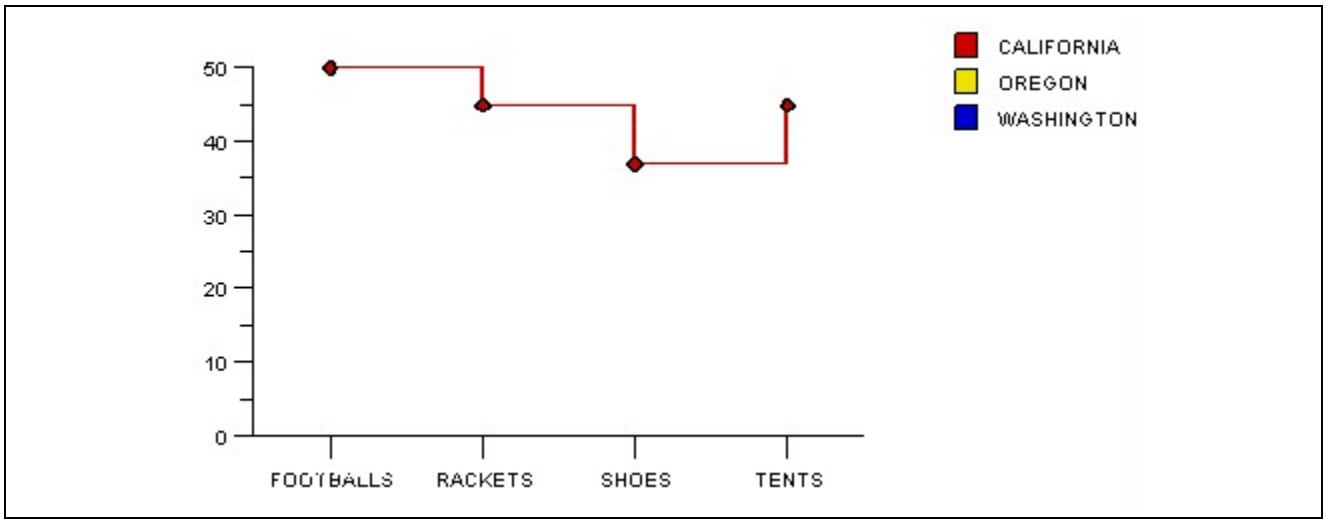


Type=%ChartType_2DBar

2D Histogram

The following is an example of a two-dimensional histogram chart.

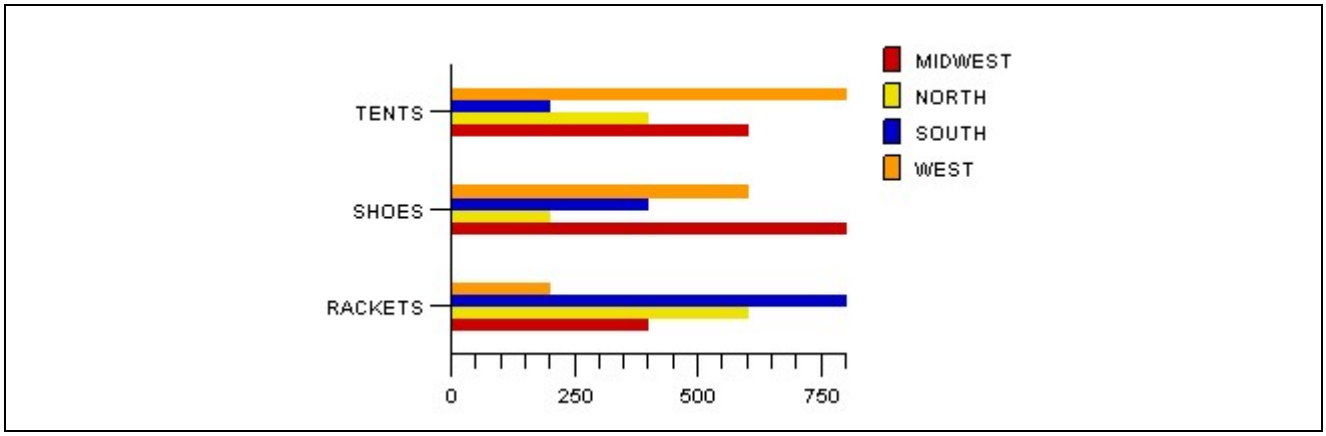
Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_2DHistogram

2D Horizontal Bar

The following is an example of a two-dimensional horizontal bar chart.

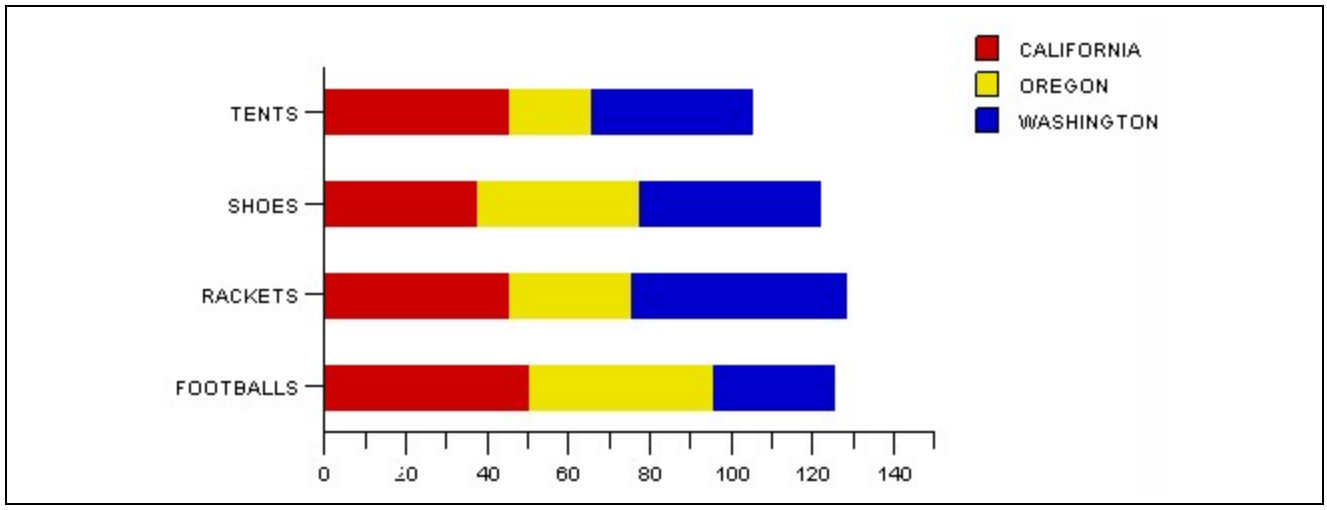


Type = %ChartType_2DHorizBar

2D Horizontal Stacked Bar

The following is an example of a two-dimensional horizontal stacked bar chart.

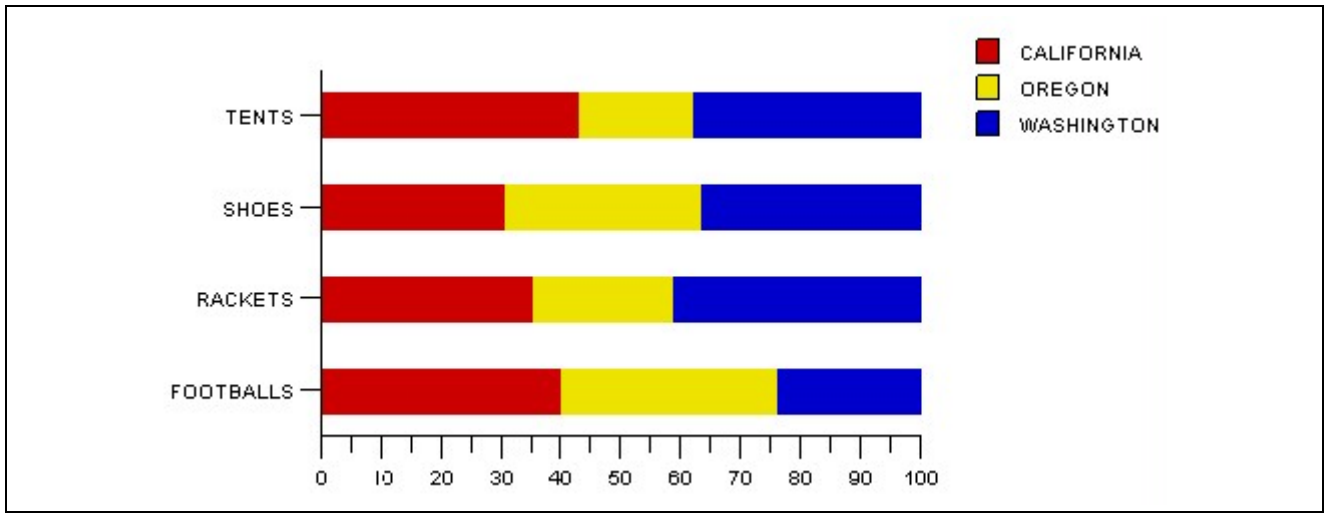
Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_2DHorizStackedBar

2D Horizontal Percent Bar

The following is an example of a two-dimensional horizontal percent bar chart.

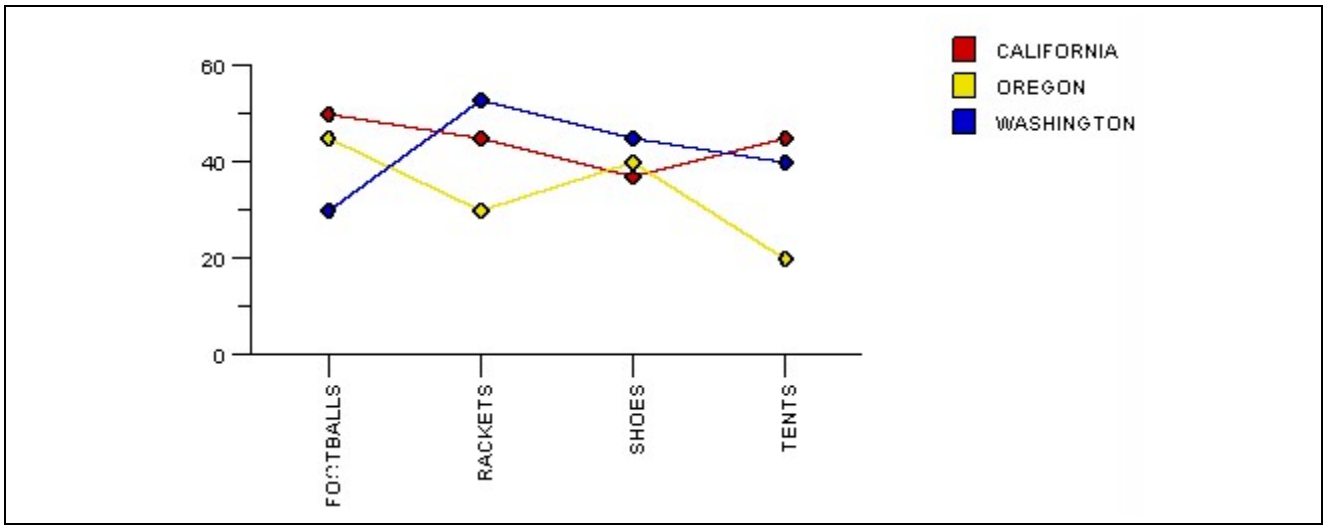


Type = %ChartType_2DHorizPercentBar

2D Line

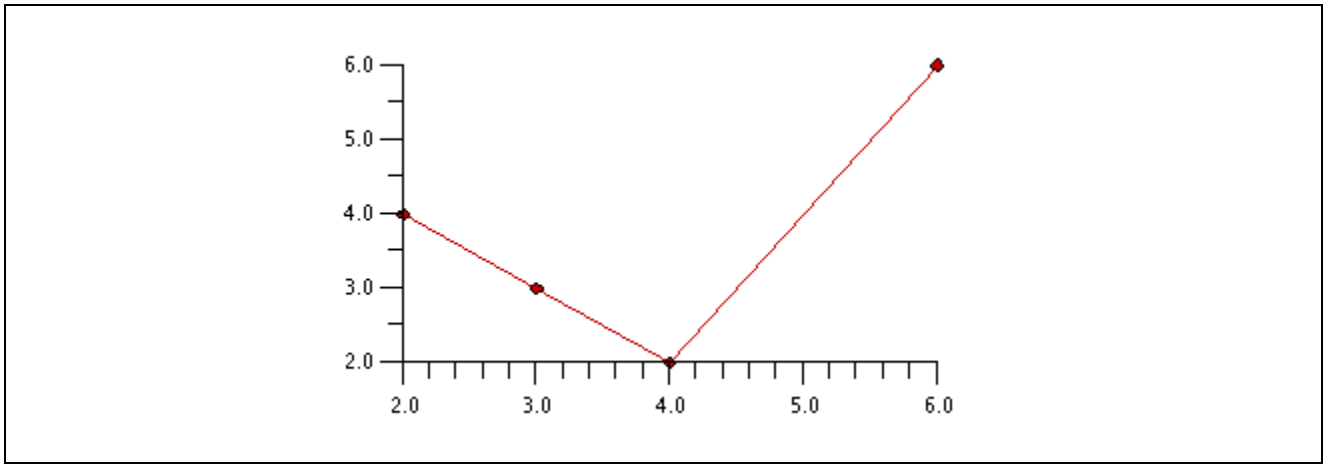
The following is an example of a two-dimensional line chart without numeric axes, that is, with IsTrueXY specified as False.

Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_2DLine

The following is an example of a two-dimensional line chart with a numeric X and Y axis, that is, with IsTrueXY specified as True.

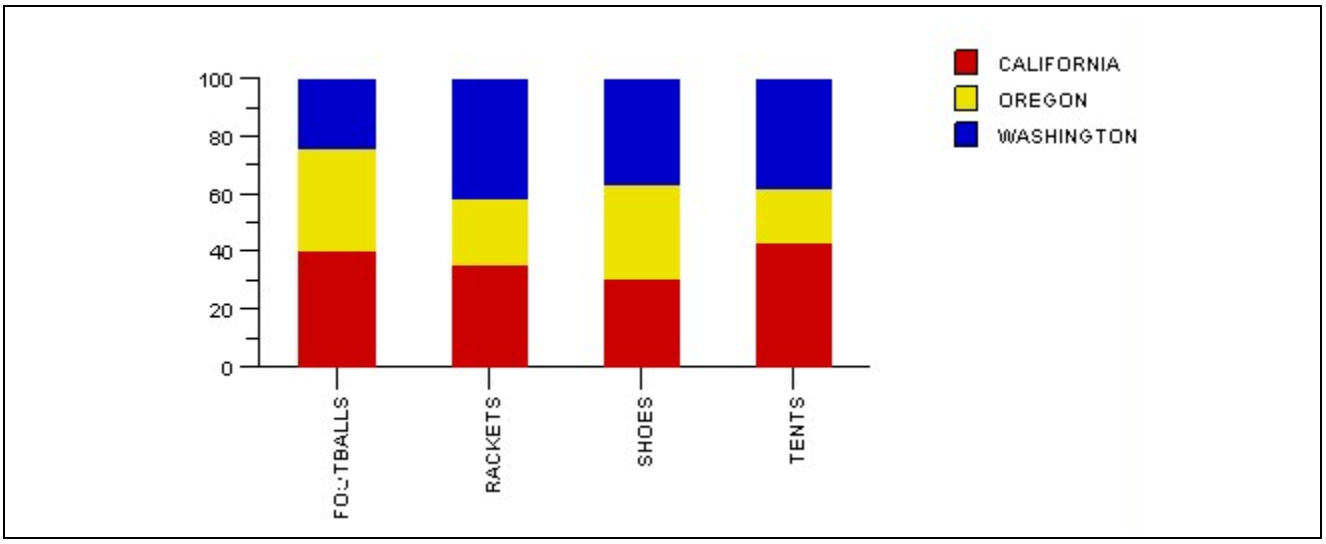


Type = %ChartType_2DLine, IsTrueXY = True

2D Percent Bar

The following is an example of a two-dimensional percent bar chart.

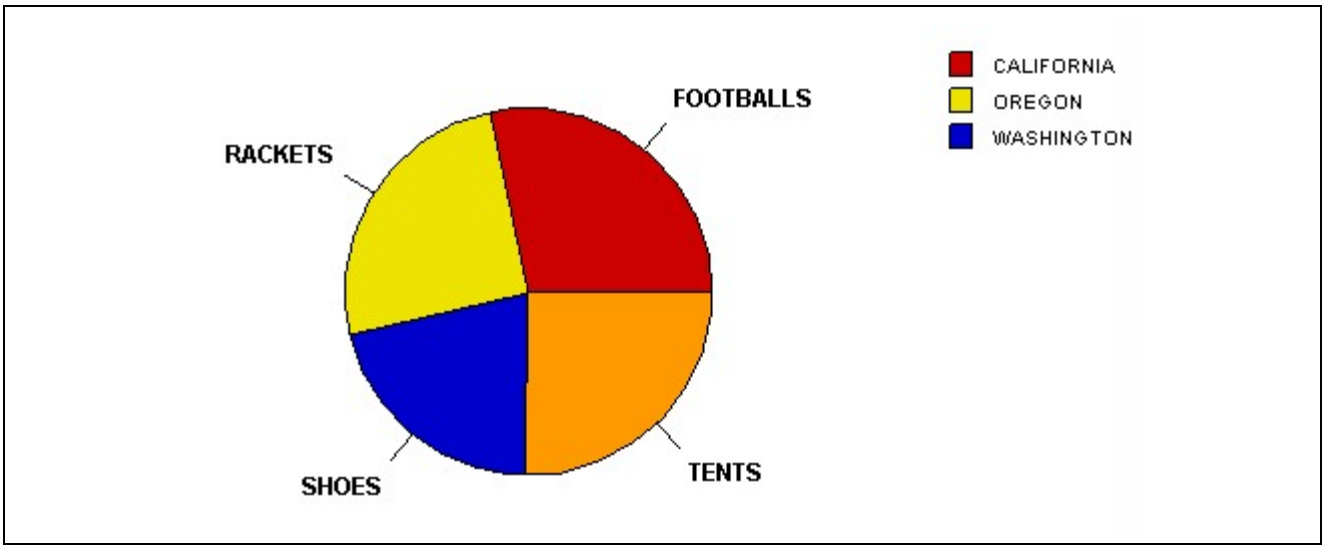
Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_2DPercentBar

2D Pie

The following is an example of a two-dimensional pie chart.

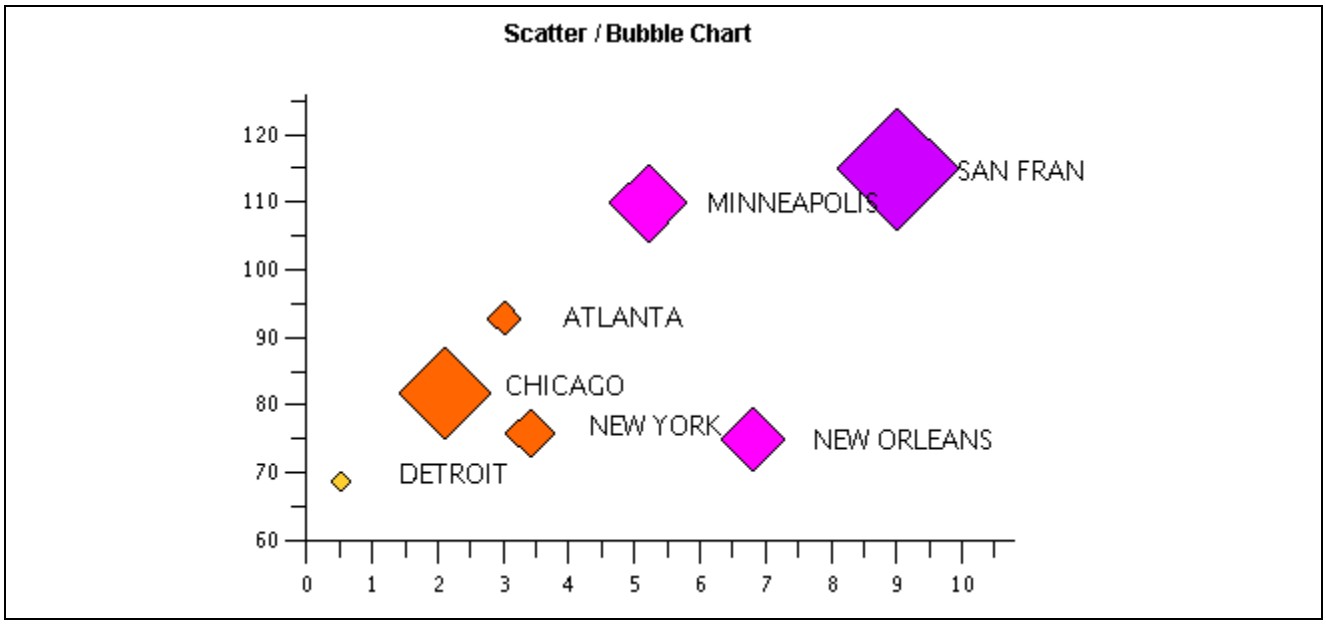


Type = %ChartType_2DPie

2D Scatter

The following is an example of a two-dimensional scatter chart.

Copyright © 1988-2007, Oracle. All rights reserved.

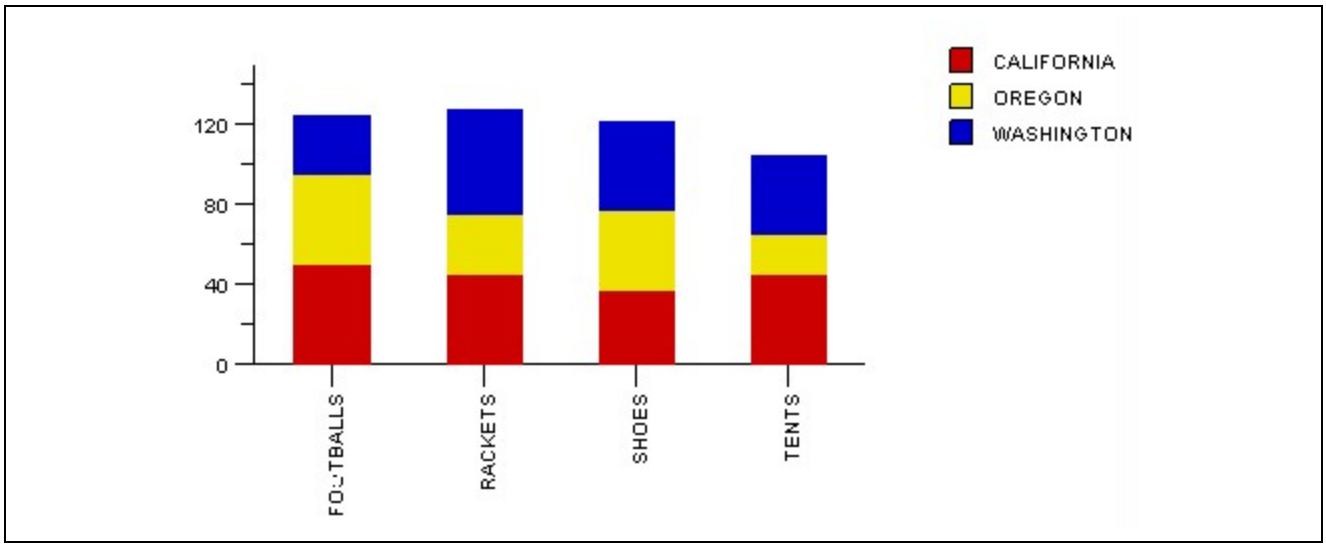


Type = %ChartType_2DScatter

Note. If you have only one point of data, use this type of chart.

2D Stacked Bar

The following is an example of a two-dimensional stacked bar chart.

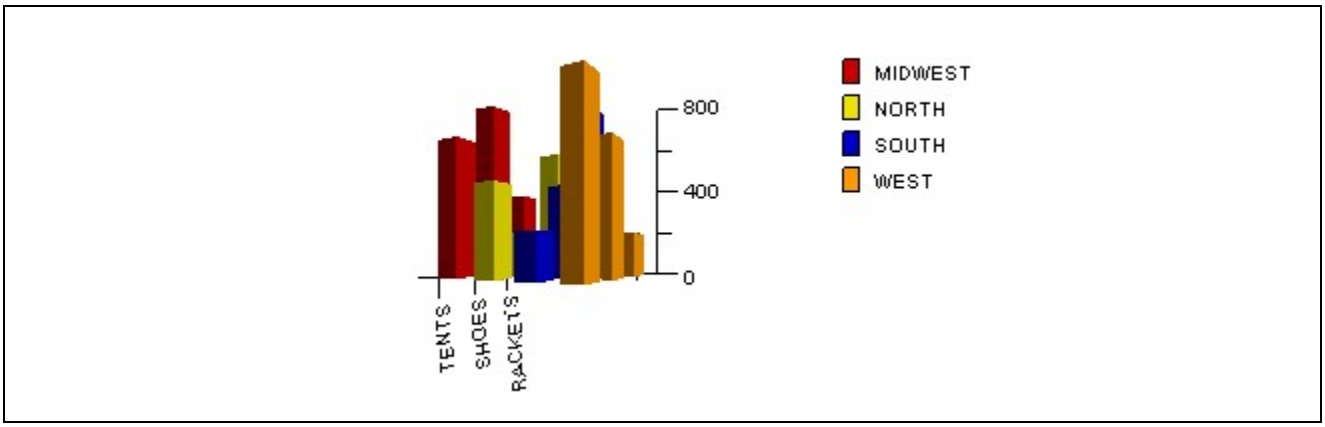


Type = %ChartType_2DStackedBar

3D Bar

The following is an example of a three-dimensional bar chart.

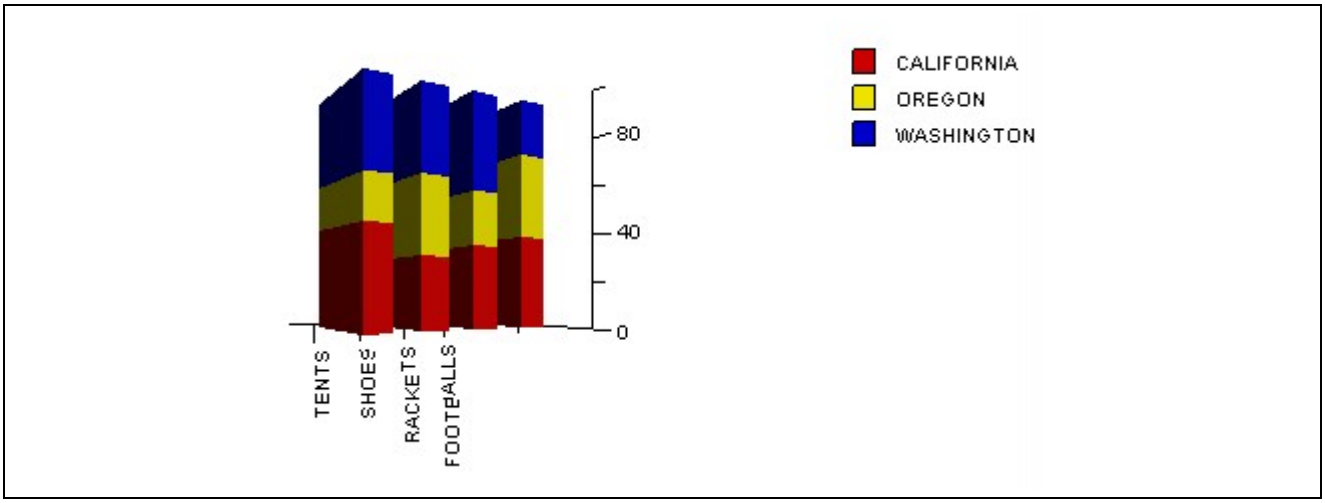
Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_3DBar

3D Percent Bar

The following is an example of a three-dimensional percent bar chart.

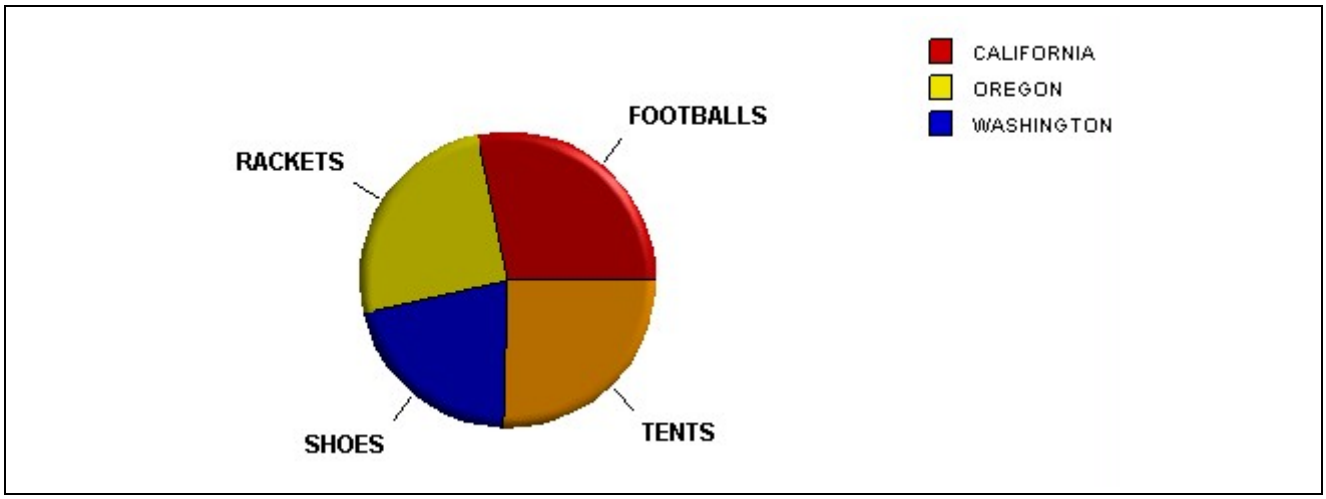


Type = %ChartType_3DPercentBar

3D Pie

The following is an example of a three-dimensional pie chart.

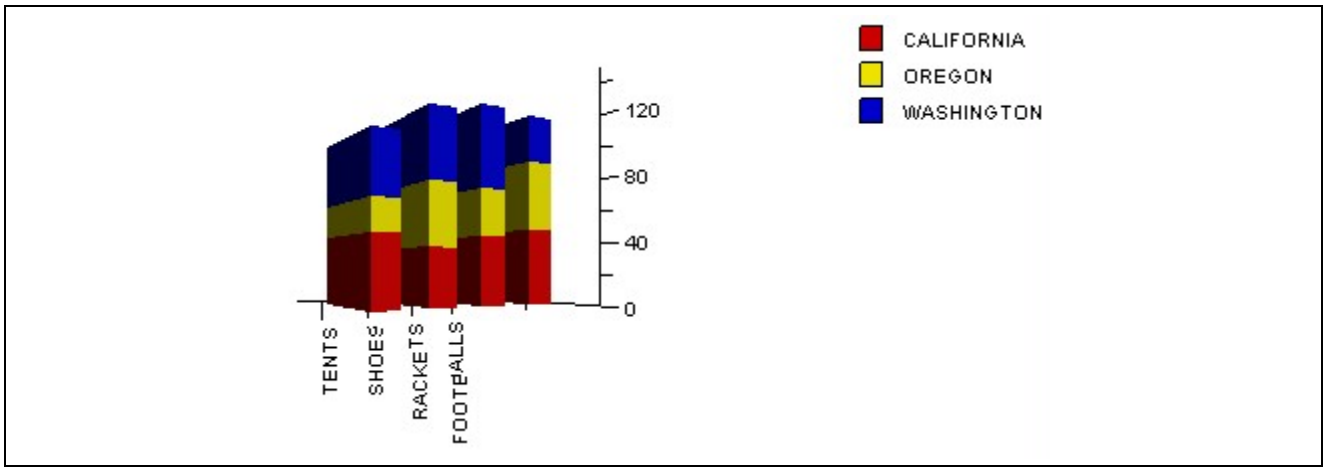
Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_3DPie

3D Stacked Bar

The following is an example of a three-dimensional stacked bar chart.

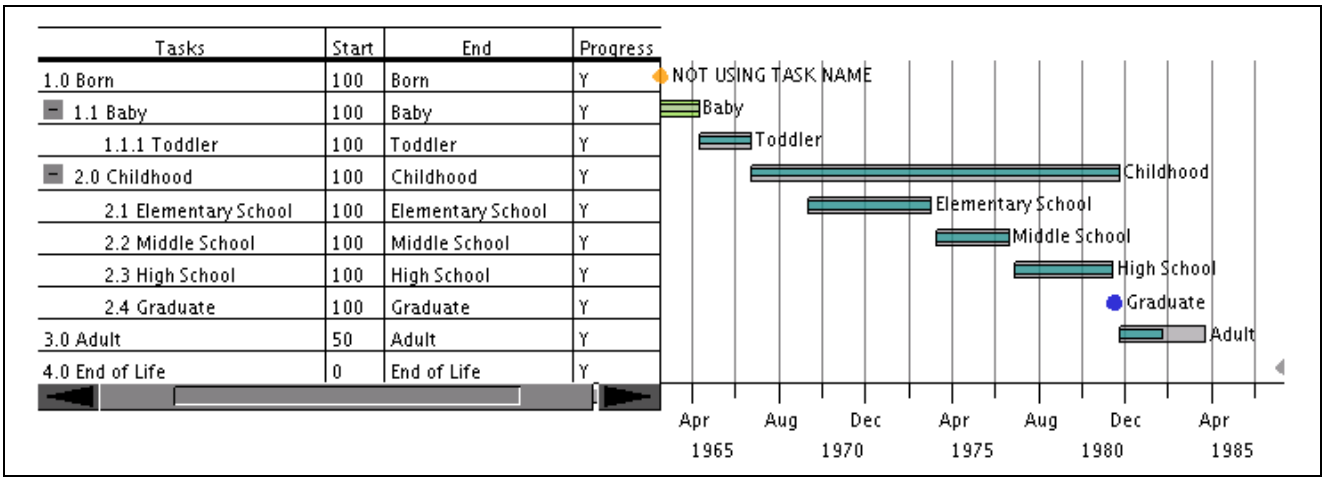


Type = %ChartType_3DStackedBar

Gantt

The following is an example of a Gantt chart.

Copyright © 1988-2007, Oracle. All rights reserved.



Type = %ChartType_Gantt

Style Sheets and Charts

A *style sheet* is a definition, like a record or field definition, that you create in PeopleSoft Application Designer. It contains a collection of formatting styles (called *classes*) each of which can be applied to either the entire chart or to specific elements within a chart.

You can use a style sheet, and the styles in a style sheet, for the different parts of a chart. You can either specify a style sheet for the entire chart (using either the chart Style or StyleSheet property) or for an element (using that element's Style property.) The override rules are:

1. If you specify the Style property for any element (XAxis, XAxisTitle, YAxis, YAxisTitle, Legend, MainTitle, and so on) the style is used.
2. If you don't specify the Style property for any element of a chart object, but you specify either the Style or StyleSheet property for the chart object itself, that style is used for the unspecified elements.
3. If you don't specify styles for any elements of a chart object, and you don't specify style for the chart object itself, a default style for any chart object is used.

Note. PeopleSoft recommends not using your own styles or style sheets unless absolutely necessary.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Style Sheet Definitions"

Considerations Using Style Sheets

The following style sheet properties are only supported for PeopleSoft Chart:

- Font Family.
- Font Foreground Color.
- Font Size.
- Font Style (only Normal and Italic are supported.)
- Font Weight (only Normal and Bold are supported).

Copyright © 1988-2007, Oracle. All rights reserved.

Default Style Sheet Classes

The default style sheet associated with a PeopleSoft Chart is PTSTYLEDEF. The default classes associated with this style sheet include:

Style Sheet Class Name	Associated Chart Element
PSCHARTDEFAULT	Used with the chart section of the Gantt chart, as well as with the following properties: • Style • XAxisStyle • YAxisStyle • LegendStyle
PSCHARTTITLE	used with MainTitleStyle
PSCHARTAXISTITLE	used with XAxisTitleStyle
PSCHARTAXISTITLE	used with YAxisTitleStyle
PSCHARTGANTTGRID	used with the table section of the Gantt chart

Considerations for Gantt Charts

In Gantt charts, the PSCHARTDEFAULT style class uses an 8 point font with a normal weight. This style class affects all text in the chart area, such as task labels, DateTime X-axis labels, and hints. The PSCHARTGANTTGRID style class uses a 10 point font, also with a normal weight, and affects all text in the table area. For the font size between the table and chart areas to be the same, the point size in the PSCHARTGANTTGRID style class should be 1.25 times the point size of the font defined in PSCHARTDEFAULT. Note, however, that the font size for PSCHARTGANTTGRID should not exceed 13; otherwise, column text may bleed over into another column.

Font Considerations for Extended Languages

The environment variable JAVA_FONTS must be set correctly so the fonts that are used in charting can be picked up by the application server's Java Virtual Machine. It must be set to include the paths to any TrueType fonts that are used to support extended languages. This can be set at the system level or in psconfig.sh.

The following is is an example of this variable being set:

```
JAVA_FONTS=$PS_HOME/jre/lib/fonts:/usr/share/fonts/default/TrueType:/usr/share⇒
/fonts/ja/TrueType:/usr/share/fonts/zh_CN/TrueType:/usr/share/fonts/zh_TW/TrueType:⇒
/usr/share/fonts/ko/TrueType;export JAVA_FONTS
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Component Processor Considerations

Sometimes records at level zero on a page are considered *work* records, even when they aren't specified as such in PeopleSoft Application Designer. Work records aren't updated by the database and are skipped when the component is run.

A level zero record gets marked as a work record when:

- All the fields for the record are used as read-only fields.
- All the values for these fields can be read from the input keys.

When a chart field is attached to a level zero record field of a record considered a work record, it's also skipped when the system determines that record's 'work' status.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Referencing Data in the Component Buffer"

Translation Considerations

If you hardcode a value for a label, a title, a data hint, and so on, that value is *not* translated. Be sure to specify field values or message catalog entries that will be translated.

Label Considerations

Chart labels are shown based on the size of the axis, fonts, and charts on the page. If all the labels are not displaying for your chart, you should either make the chart larger or the font smaller.

Pie chart labels are staggered, so that, in most cases, both labels can be read. However, if one or both of the labels are very long, they could be truncated or completely missing from the chart.

For Gantt charts, use double quotes to correctly display the use of single quotes in your label, if applicable.

Bar Chart Considerations

If the bars on a bar chart are too narrow, a user may not be able to click on them. Sometimes this affects a few bars on a chart, sometimes all of the bars cannot be clicked. The workaround is to put fewer bars on your chart, or to make your chart bigger, so that each bar is wider.

Copyright © 1988-2007, Oracle. All rights reserved.

iScripts and Charts

PeopleSoft supports two types of charts:

- Charts generated using the chart control on a page.
- Charts generated at runtime and used in an iScript.

To build a chart at runtime, use the CreateObject function instead of the GetChart function. You can then use the GetChartURL Response class method to use the URL in your application.

The following is an example of building a chart using a standalone rowset:

```
Function IScript_GetChartURL()
local object &MYCHART;
local string &MYURL;

&MYCHART = CreateObject("Chart");
&MYCHART.SetData(xx);

/* xx will be a data row set */

&MYURL = %Response.GetChartURL(&MYCHART);

/* use &MYURL in your application */
...
End-Function;
```

See Also

Chapter 11, “Chart Class,” Creating a Chart Using an iScript, page 407

Tick Mark Considerations

If you want to remove the tick marks from the X or Y axis, you need to specify an empty array for the SetXAxisLabels or SetYAxisLabels methods.

The following example removes the tick marks from the Y axis:

```
&Arr = CreateArray("");
&Chart.SetYAxisLabel(&Arr);
```

If you set the X axis labels to an empty array to remove the ticks, your must be plotting a single series on the chart, because labels are automatically generated for each series if you do not provide them.

See Also

Chapter 11, “Chart Class,” SetXAxisLabels, page 362

Chapter 11, “Chart Class,” SetYAxisLabels, page 363

 Copyright © 1988-2007, Oracle. All rights reserved.

Bubble Chart Considerations

Bubble charts are a type of scatter chart, where points of data are plotted in two dimensional space along a numeric X and Y axis. Bubble charts add the ability of being able to display an extra dimension of data through the size of each point or *glyph*, that is, the size of the glyphs is scaled from the data.

Each point in a bubble chart can be individually colored, clicked on to start PeopleCode program, and can have pop up text associated with it. Programming a scatter chart is similar to creating the other charts: at a minimum, you need to specify the data record (rowset), the X data field, and the Y data field.

The size of the glyphs can be set within the data. Each point in the chart, along with the regular pop up text can also have online annotations associated with it.

When you create a bubble or scatter chart, a TrueXY axis is automatically set, that is, the IsTrueXY property is set to true.

See Also

Chapter 11, "Chart Class," Line Chart Considerations, page 335

Chapter 11, "Chart Class," IsTrueXY, page 354

Line Chart Considerations

The X axis on a line chart is either a category axis or a TrueXY axis:

- With a category axis, each data marker is placed along the axis according to its position in the data series.
- With a TrueXY axis, each data marker is placed along the axis according to its numeric value.

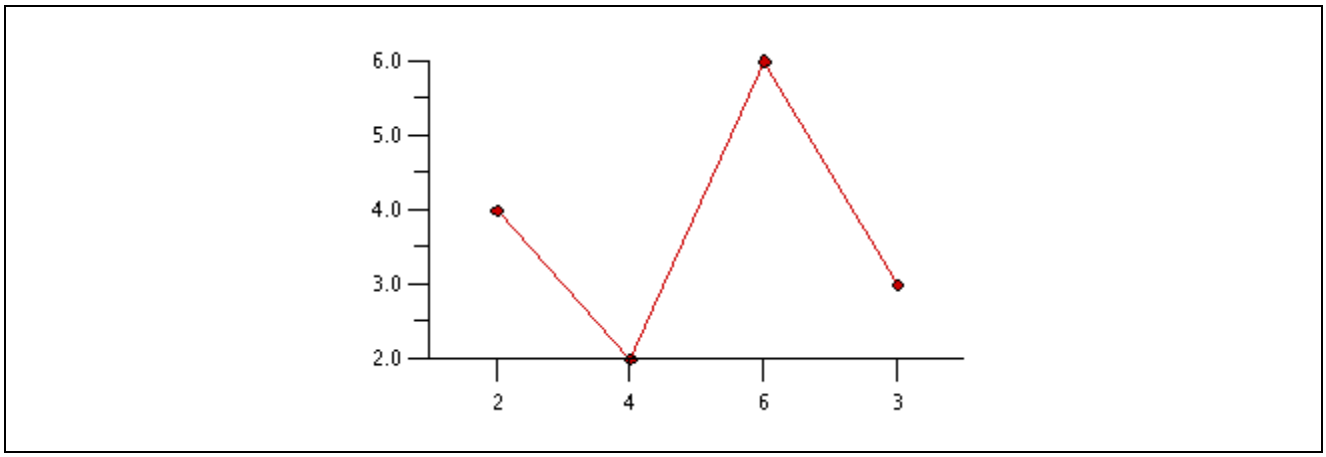
You use the IsTrueXY property to control whether the X axis is a TrueXY axis or not.

The following charts use the exact same data series:

X axis	Y axis
2	4
4	2
6	6
3	3

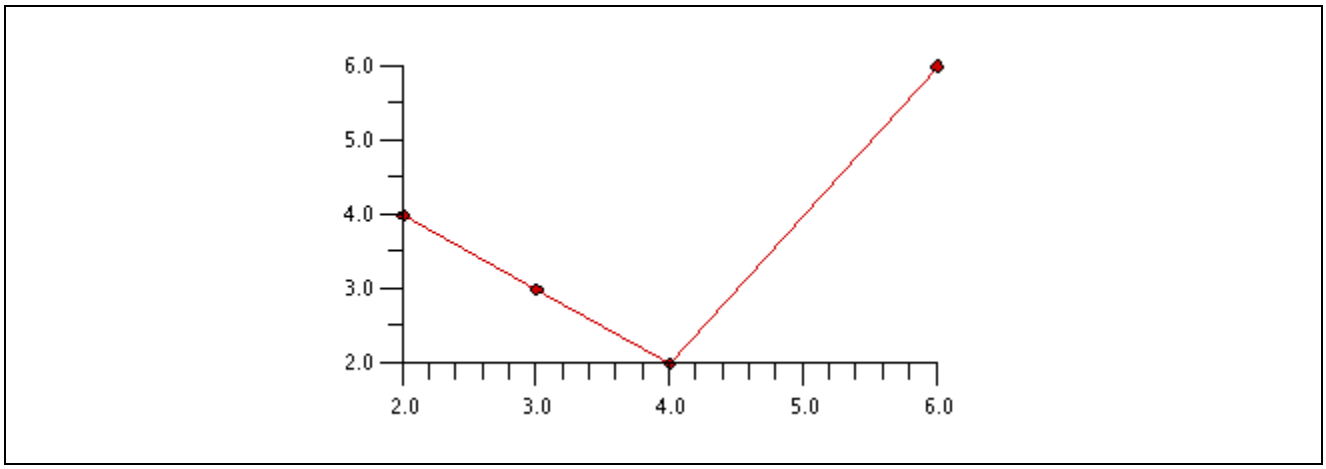
In the first example, IsTrueXY is set to false, which means that the X axis is a category axis and the values along it are place in the order in which they occur in the data set. Note the order of the numbers along the X axis:

Copyright © 1988-2007, Oracle. All rights reserved.



Line chart with category X axis

In the following example, IsTrueXY is set to true, so the X axis is a TrueXY axis. The order of the numbers along the X axis is now numeric.



Line chart with numeric X axis

See Also

Chapter 11, “Chart Class,” IsTrueXY, page 354

Chapter 11, “Chart Class,” Data Series, page 336

Data Series

A data series is a group of related information. You can group the same information into more than one series. All of the following charts use the same data set.

Product	Region	Sales
1	X	5.50
1	Y	4.60

 Copyright © 1988-2007, Oracle. All rights reserved.

Product	Region	Sales
2	X	5.70
2	Y	6.00
3	X	6.00
3	Y	7.20
3	Z	8.25
4	X	6.50
4	Y	7.30
5	X	7.00
5	Y	7.40
6	X	7.50
6	Y	7.50
6	Z	-3.40
7	X	8.00
7	Y	7.00
7	Z	2.30
8	X	8.10

In the following example, there are three data series, based on product:

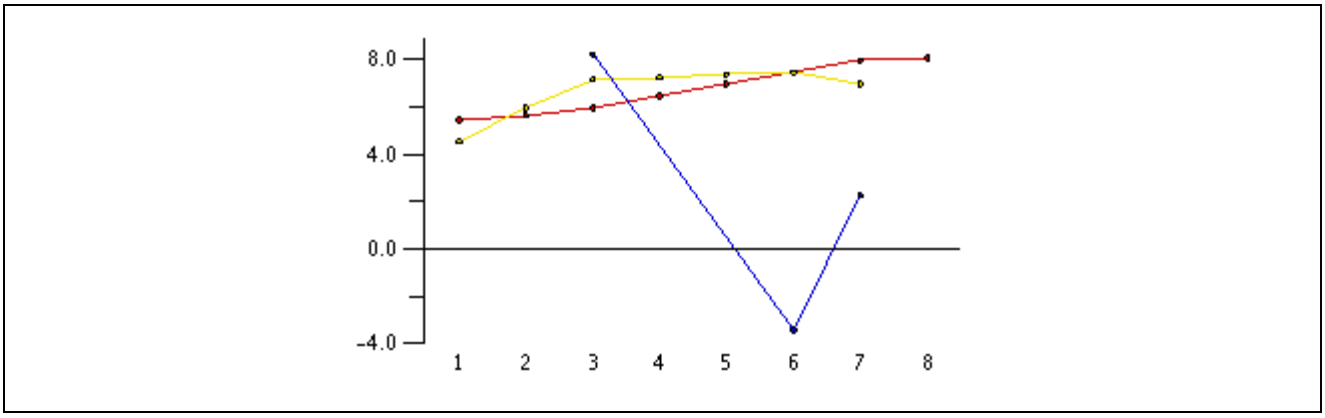


Chart with three data series

Each series is indicated with its own discreet line.

In the following example, each data series is grouped by product. There are several different products, each with its own data series.

Copyright © 1988-2007, Oracle. All rights reserved.

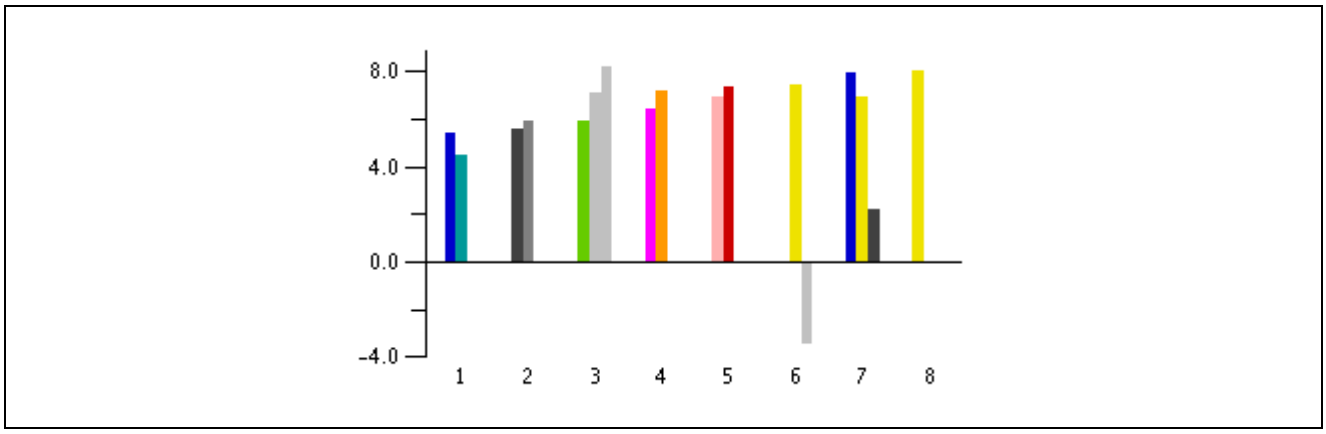


Chart with data series grouped by product

Gantt Charts

A Gantt chart displays tasks along a timeline. Gantt charts are frequently used in project management because they can provide a graphical illustration of a schedule, which helps to plan, coordinate, and track specific tasks in a project.

This section provides an overview of Gantt chart terminology and discusses how to:

- Use Gantt charts in PeopleSoft Pure Internet Architecture.
- Create Gantt charts.
- Specify DateTime axis format.
- Work with start and end dates.
- Use Gantt glyphs.

Understanding Gantt Chart Terms

Gantt charts can be composed of many different elements, such as a title, a label, and so on. The following is a list of terms that apply specifically to Gantt charts.

Activity or Task See Task.

Baseline A point-in-time snapshot of the project plan that allows for the identification of how the project plan has changed since that point in time.

Data area The part of the chart that shows graphically the span of dates that each task (or activity) covers. (The right side of the displayed chart.)

Dependencies A relationship between one task and other tasks that drives when tasks can begin and end. A dependent task cannot start before the task it's dependent upon is either started or completed (depending on business rules defined in the application).

Milestone A specialized row on the Y axis that names a point in time that is meaningful.

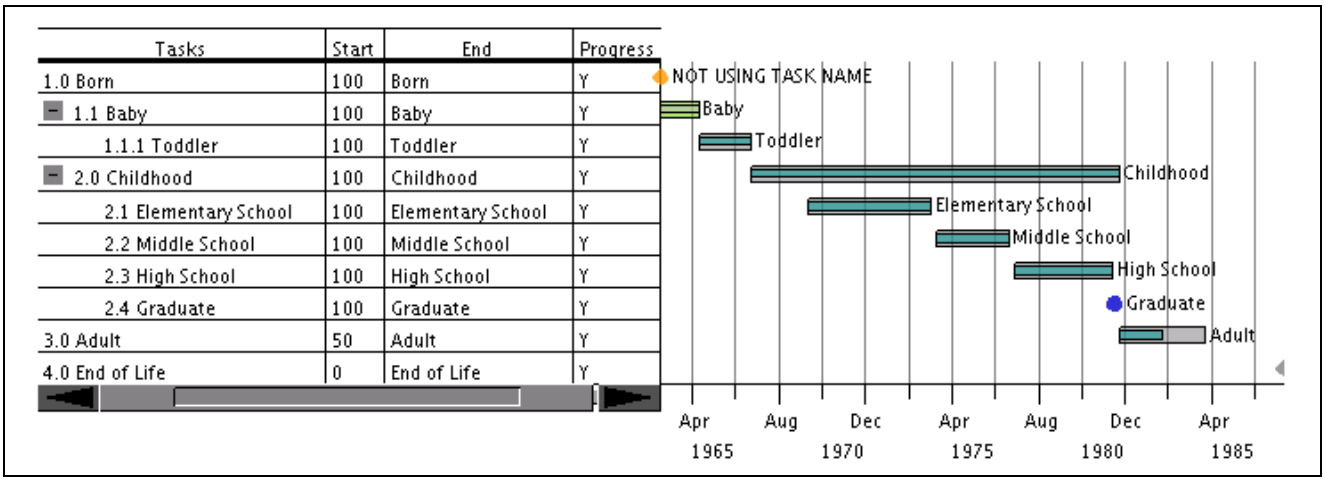
Resource Something that is assigned to work on an activity or task.

Copyright © 1988-2007, Oracle. All rights reserved.

Task or Activity	For a Gantt chart, each row (task or activity), is represented in the Y axis of the chart, and the span of dates for each row is displayed on the X axis.
X axis	The axis that data is measured against. For a Gantt chart, the X axis is the display of dates which items in the Y axis span.
X axis labels	The part of the chart that lists the time scale. The X-axis labels support multiple levels of granularity: second, minute, hour, day, week, month, year.
Y axis	The axis that contains the data. In most charts, this is the vertical axis. In a horizontal bar chart, this is the horizontal axis. For a Gantt chart, the Y axis is the display of tasks or activities, and is displayed as rows.
Y axis labels	The part of the chart that lists the activities and attributes of those activities. The activities are shown hierarchically, and subtasks can be hidden or displayed by expanding or collapsing higher-level tasks.

Using Gantt Charts in PeopleSoft Pure Internet Architecture

The Gantt chart consists of two sections: a table section and chart section.



Example Gantt chart

The left side of the chart, where you see the table, is the table section. The other side, where you see the bars, is the chart section.

Table Section

The table section contains all of the tasks and associated subtasks and displays them in a hierarchy. Tasks that contain subtasks are called *parent tasks* and subtasks are called *child tasks*. You can click on the expand (collapse) image to the left of the task name (Baby, above) to expand or collapse the hierarchy of subtasks. Note that subtasks may also act as parent tasks to other subtasks.

Each task has a name Born, Baby, Childhood, and so on.) It also has a level. Parent tasks have a higher level than child tasks.

Horizontal scrollbars are provided under the table section to support scrolling through the task-related columns. The amount that a table section scrolls depends on the value set for the SetTableXScrollBar method. Each click on the arrow scrollbar scrolls the task either left or right by the amount set with the SetTableXScrollBar method (or 10% of the table area, if a value isn't set.)

See Chapter 11, "Chart Class," SetTableXScrollbar, page 383.

Copyright © 1988-2007, Oracle. All rights reserved.

Chart Section

The chart section displays the tasks, task dependencies, and milestones graphically.

In the example above, each horizontal bar represents a single task. The progress bar, that is, the percentage complete for a particular task, is indicated with a bar of a different color on top of the task bar. The milestone date (in the above screen shot, Graduate) is represented with a circle.

You can also have dependencies between the tasks. These are represented as lines connected the task bars.

Creating Gantt Charts

Every Gantt chart has at least one data set used to define the tasks and the information related to each task, such as start date, end date, milestones, percent finished, and so on.

A second data set can be used to describe dependencies between the tasks.

The following is a list of the required methods for using the Gantt chart:

- SetTaskData

  Use this method to specify where most of the information for the Gantt chart is stored. You can specify either a rowset or a record.

- SetTaskID

  Use this method to specify the task ID, or name, of the task. Every task must have a unique task identifier. The task ID is used to support task linking and dependencies.

- SetPlannedStartDate

  Use this method to specify the planned starting date of the task. Each task must have its own planned starting date.

- SetPlannedEndDate

  Use this method to specify the planned ending date of the task. Each task must have its own planned ending date.

Though not required, PeopleSoft recommends also using the SetTaskName method to display meaningful information in the table section of the Gantt chart.

If you only use the required methods, and do not use the SetTaskAppData method, PeopleSoft recommends that you also use the SetChartArea method and dedicate most, if not all, of the entire area to the chart (and not the table.)

If you specify one actual date, either for start or end, you must specify the other (SetActualStartDate, SetActualEndDate).

In addition, if you want to use dependency data, the following methods are required:

- SetTaskDependencyData

  Use this method to specify where most of the information for the dependency data is stored. You can specify either a rowset or a record.

- SetTaskDependencyParentID

  In a dependency, one task depends on another. Use this method to specify the parent task, that is, the one that the other (child) task depends upon.

- SetTaskDependencyChildID

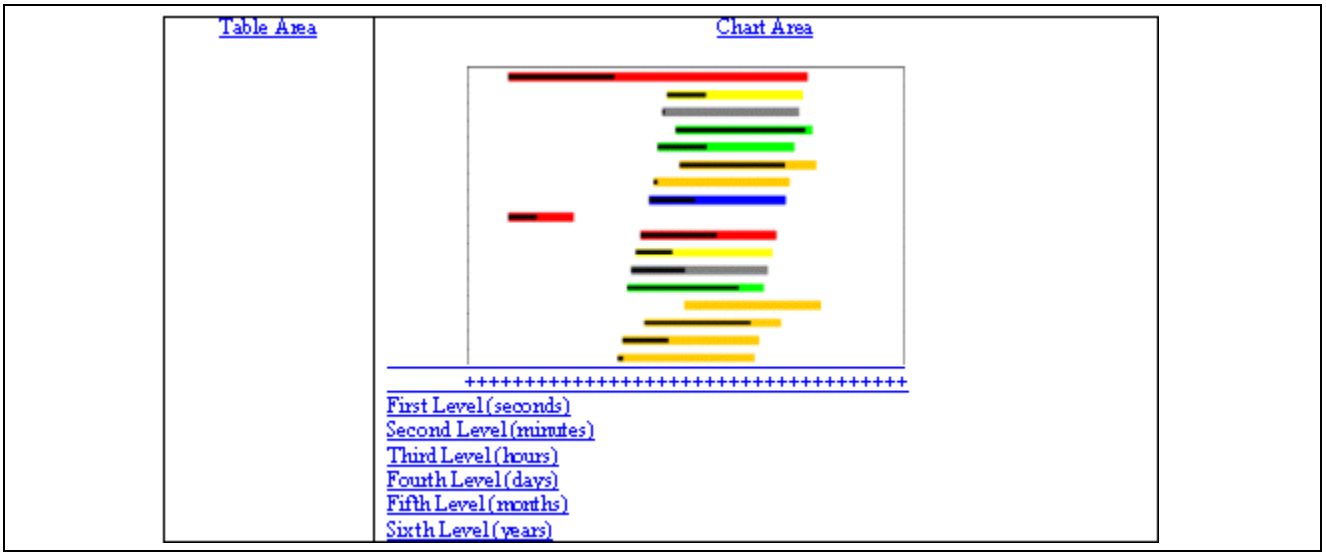
 Copyright © 1988-2007, Oracle. All rights reserved.

In a dependency, one task depends on another. Use this method to specify the child task, that is, the one that depends on another.

Specifying DateTime Axis Formats

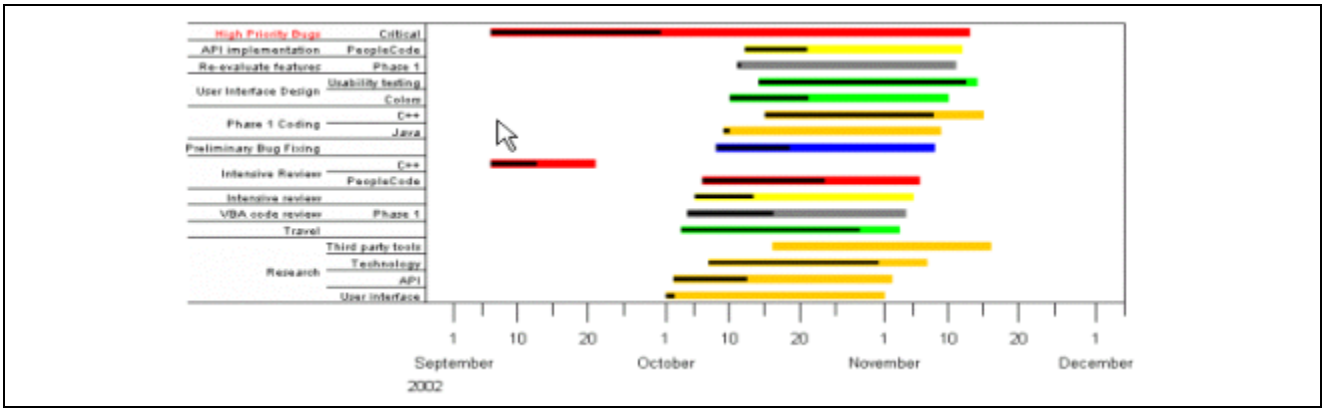
You can control the display of the label format that describes the DateTime axis in up to six different time increments, from seconds to years, or any combination thereof simultaneously for a given time period. Each time increment, if enabled through the DateTimeAxis specific methods, occupies a single level within the X axis hierarchy. Smaller time increments display closer to the axis, while larger time increments display farther from the axis.

The following example shows the levels of the labels for the DateTime axis. The first level, in seconds, is the closest level to the chart edge, next are minutes at the second level, then hours at the third level, and so on. Generally, the smallest unit should be on the lowest level. So, if months and years are displayed, months display at the first level and years display at the second.



Example of levels of datetime axis labels

In the following example, days, months, and years are defined at levels 1, 2, and 3, respectively. This means that the days of the month (1, 10, 20) are displayed closest to the chart area, the months are displayed below them, and the years below that.



DateTime Format Example

By default, the DateTime axis displays both the year and the months, with the major tick mark marking the months.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. The charting algorithm may or may not display a given time increment, even if it is enabled. The resolution of the given time increments may be too great, depending on the length of the time period defined in the chart section.

See Also

Chapter 11, "Chart Class," Gantt Chart DateTimeAxis Specific Methods, page 397

Working with Start and End Dates

You must set the end date to be after the start date, for both the planned as well as actual dates, or else you receive an error.

Both the start and end dates you specify for a task must fall within the axis dates, that is, the dates specified by the AxisEndDateTime and AxisStartDateTime properties. If you specify task dates that fall outside of the axis dates, the task is still displayed in the table section of the Gantt chart, but no bar displays in the chart section of the Gantt chart, even if one of the dates does fall inside the axis dates.

Using Gantt Glyphs

The following lists all the glyphs you can use with the Gantt chart.

You can use either the numeric or constant value.

Constant: %ChartGlyph_Axe

Numeric value: 1

Constant: %ChartGlyph_Bar

Numeric value: 2

Constant: %ChartGlyph_Box

Numeric value: 3

Constant: %ChartGlyph_Circle

Numeric value: 4

Constant: %ChartGlyph_Cross

Numeric value: 5

Constant: %ChartGlyph_Diamond

Numeric value: 6

Copyright © 1988-2007, Oracle. All rights reserved.

Constant: %ChartGlyph_Dodecahedron

Numeric value: 7

Constant: %ChartGlyph_Icosahedron

Numeric value: 8

Constant: %ChartGlyph_Sphere

Numeric value: 9

Constant: %ChartGlyph_Square

Numeric value: 10

Constant: %ChartGlyph_Star

Numeric value: 11

Constant: %ChartGlyph_Triangle

Numeric value: 12

Error Handling

The PeopleCode program is terminated if a field or record is missing at runtime.

If a valid record is specified but no data is found, or for any other error, the chart is replaced by the message " Image creation failed."

If the Java Runtime Environment (JRE) is missing, an error appears and is logged.

Data Type of a Chart Object

Chart objects are declared using the Chart data type. For example,

```
Local Chart &MyChart;

Component Chart &Abs_Hist_Chart;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Scope of a Chart Object

A chart object can be instantiated only from PeopleCode.

You can use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message notification, a Component Interface, and so on.

Chart Class Reference

The following sections contain the methods and properties used with a chart.

The reference is divided up among the different parts of a chart, however, all methods and properties are used with the chart object: there are no subobjects in a chart.

- Chart Data Methods and Properties: Used to set X and Y axes data.
- Chart Appearance Properties: Used to set the type of graph, the line style, and so on.
- Chart Color Methods: Used to set and manipulate colors for either a series or for each point.
- Chart Axis Methods and Properties: Used to further control the appearance of the axes.
- Chart Title Properties: Used to set the text and style of titles for the graph and the axes.
- Chart Legend Methods and Properties: Used for each series legend.
- Chart Overlay Methods and Properties: Used to set and manipulate overlay data.
- Bubble Chart Methods and Properties: Used to set and manipulate bubble charts.
- TrueXY Line Chart Properties: Used to set and manipulate TrueXY line charts.
- Gantt Chart Methods and Properties: Used to set and manipulate Gantt charts.

Chart Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateObject

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetChart

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetChartURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetGanttChart

Copyright © 1988-2007, Oracle. All rights reserved.

Chart Data Methods

These methods are used to set X and Y axes data.

Refresh

Syntax

```
Refresh()
```

Description

Use the Refresh method if the underlying data of the chart has changed, and you want to update the chart.

Note. You don't need to use the Refresh method after setting a property or method of the chart itself, such as setting the starting point or changing colors. When a method or property is used, the refresh method is automatically called.

Parameters

None.

Returns

None.

Example

```
&MyChart.Refresh();
```

See Also

Chapter 11, "Chart Class," DataStartRow, page 353, Chapter 11, "Chart Class," DataWidth, page 353, Chapter 11, "Chart Class," SetDataXAxis, page 351 and Chapter 11, "Chart Class," SetDataYAxis, page 352.

Reset

Syntax

```
Reset()
```

Description

Use the Reset method to clear out all existing chart settings and data.

Parameters

None.

Returns

None.

See Also

Chapter 11, "Chart Class," Refresh, page 345

Copyright © 1988-2007, Oracle. All rights reserved.

SetData

Syntax

SetData({*Record_Name* | *&Rowset*})

Description

Use the SetData method to specify where the data from the chart is coming from.

You can use either a record name or an already instantiated, populated rowset. In general, specify a record when building a chart from page data and a rowset when building a chart from a standalone rowset.

If you make a change to the underlying data of a chart, use the Refresh method to update the chart.

Parameters

Parameter	Description
Record_Name	Specify the name of a record to be used to populate the chart with data.
&Rowset	Specify the name of an already instantiated rowset to populate the chart with data. This is generally a standalone rowset.

Returns

None.

Example

The following is the minimum code you need to create a Peoplesoft chart.

```
Local Chart &MyChart;

&MyChart = GetChart(ABS_HIST.CHART);
&MyChart.SetData(ABS_HIST);
&MyChart.SetDataYAxis(ABS_HIST.Reason);
&MyChart.SetDataXAxis(ABS_HIST.Duration);
```

The following example creates a chart using a standalone rowset.

```
Function IScript_GetChartURL()
local object &MYCHART;
local string &MYURL;
local rowset &MYROWSET;

&MYCHART = CreateObject("Chart");

&MYROWSET = CreateRowset(Record.ABS_HIST);

&EmplID = %Emplid;

&MYROWSET.Fill("Where EMPLID :=1", &EmplID);

&MYCHART.SetData(&MYROWSET);
&MYCHART.SetDataXAxis(ABS_HIST.ABSENCE_TYPE);
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYCHART.SetDataYAxis(ABS_HIST.DURATION_DAYS);

&MYURL = %Response.GetChartURL(&MYCHART);

/* use &MYURL in your application */
...
End-Function;
```

See Also

Chapter 11, "Chart Class," SetDataXAxis, page 351 and Chapter 11, "Chart Class," SetDataYAxis, page 352.

SetDataHints

Syntax

SetDataHints(*Record_Name.Field_Name*)

Description

Use the SetDataHints method to specify the field that contains the text that displays as a data hint.

What displays for a data hint depends on the value specified for the data hints method, as well as the value specified for the series and the data method.

Series Data but No Data Hints

If you do not specify data hints, but you *do* specify both the data series and data methods, the hints appear as follows:

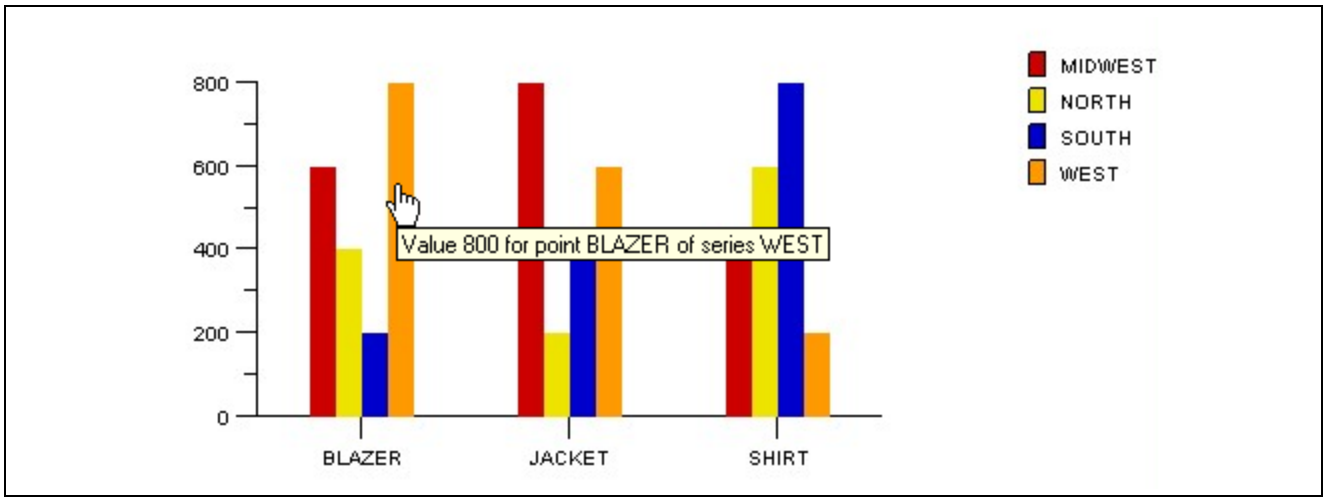
```
Value value for point x position of series series
```

Where:

- *value* is the Y data value
- *x position* is the X axis label
- *series* is the series value

The following is an example of this type of data hints. The pop-up menu appears when the cursor hovers over any data value.

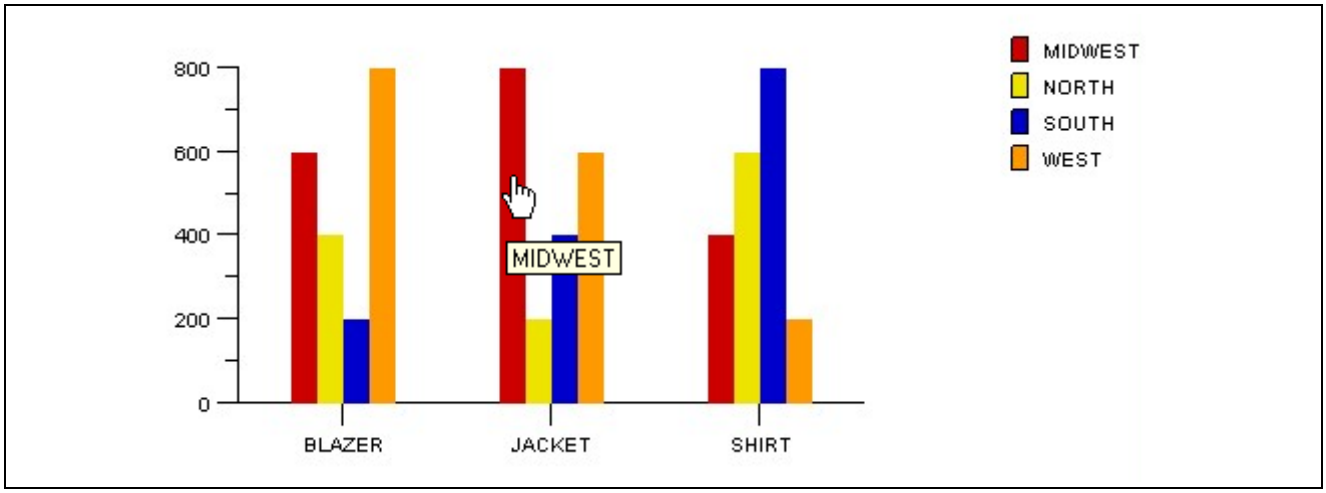
Copyright © 1988-2007, Oracle. All rights reserved.



Data hints with multiple series example

Data Hints, Data, and Series

If you specify data hints, that is the value that appears. In the following example, the data hints are set to the same value as the series is set, that is, the region.

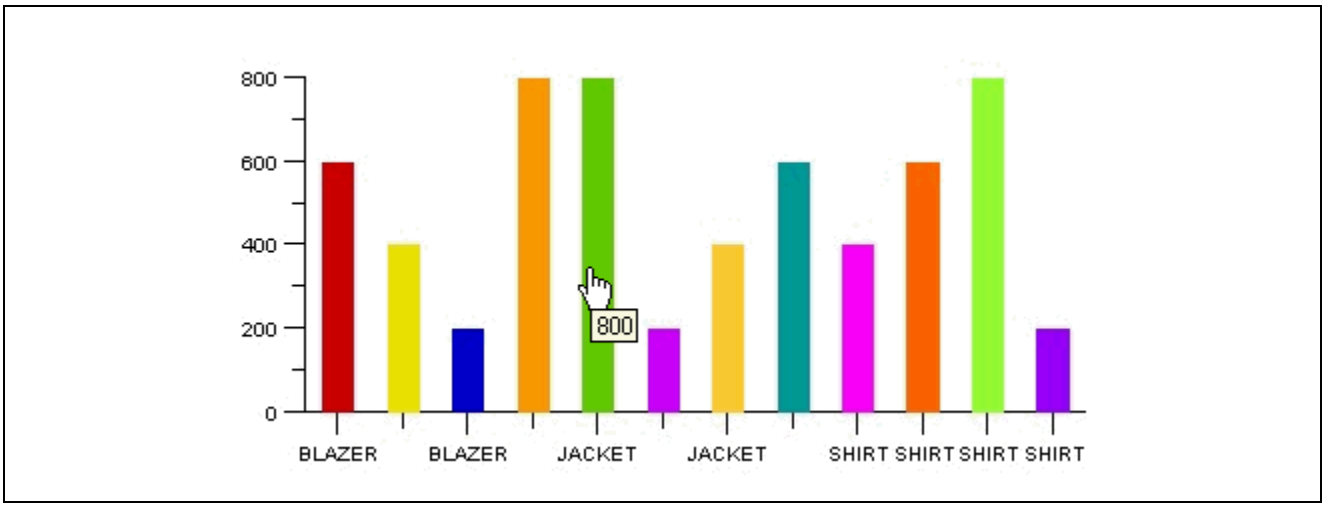


Data hints set to the region value

Data and Series, but no Data Hints

If you do not specify SetDataHints and do not specify SetDataSeries, the value of the Y axis appears in the data hint:

Copyright © 1988-2007, Oracle. All rights reserved.



Data hints from default data

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data hints for the chart.

Returns

None.

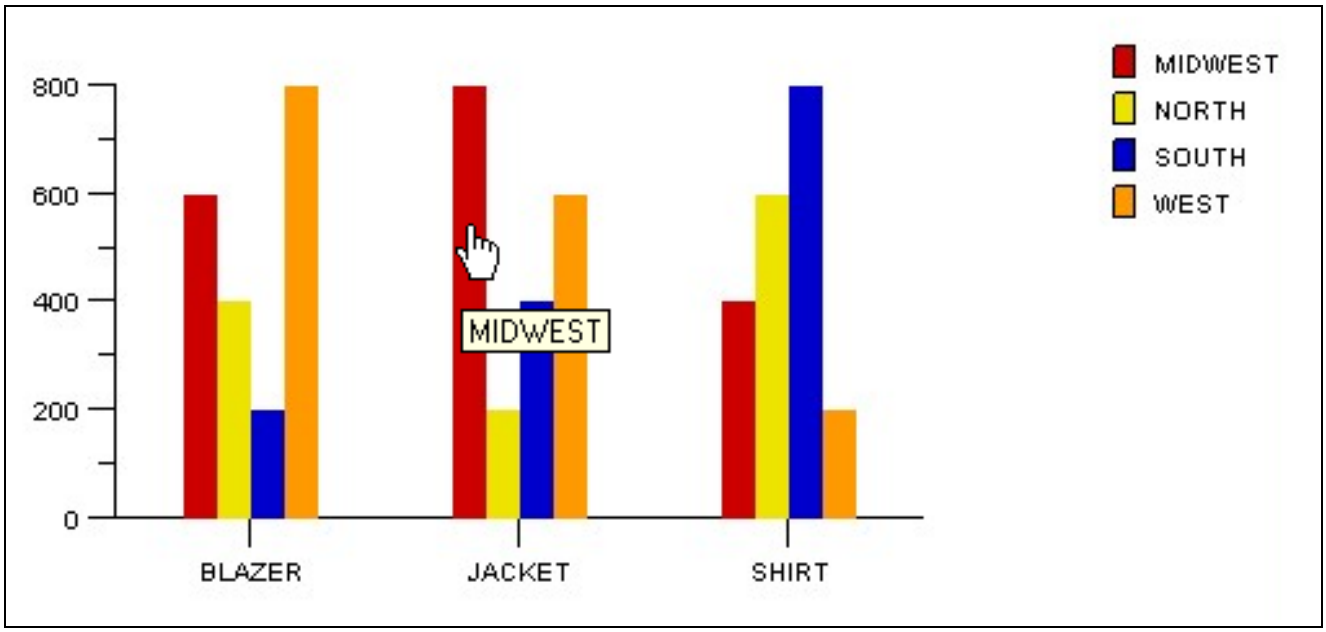
Example

The following PeopleCode example:

```
&oChart.SetDataHints(QE_CHART_RECORD.QE_CHART_REGION);
```

Could produce data hints as follows:

Copyright © 1988-2007, Oracle. All rights reserved.



Data hints set to the region value

See Also

Chapter 11, “Chart Class,” SetDataXAxis, page 351 and Chapter 11, “Chart Class,” SetDataSeries, page 350.

SetDataSeries

Syntax

SetDataSeries(*Record_Name.Field_Name*)

Description

Use the SetDataSeries method to specify the name of the field containing the series values. Every distinct value in this field is considered a unique series.

The series is plotted in the order given by the record.

By default, the legend is populated by the value of this field.

If this value is Null, the system assumes there is only one series to plot.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the series values for the chart.

Returns

None.

Example

```
&MYCHART.SetDataSeries(MYRECORD.MYSERIES);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 11, "Chart Class," SetDataXAxis, page 351 and Chapter 11, "Chart Class," SetDataYAxis, page 352.

SetDataURLs

Syntax
SetDataURLs(*FieldName*)

Description

Use the SetDataURLs method to specify a URL to be launched when a data point is clicked in the chart. The URL for each point must be given as a string, that is, enclosed with quotation marks. You can also specify *recordname.fieldname*. Use this method for charts created from a chart control or a standalone rowset.

One use of this method could be to launch a JavaScript that generates a pop-up menu.

Parameters

Parameter	Description
FieldName	Specify a field name that's used to populate the URL for each data point.

Returns

None.

Example

```
&MyChart.SetDataURLs(MyRecord.URL);
```

SetDataXAxis

Syntax
SetDataXAxis(*Record_Name.Field_Name*)

Description

Use the SetDataXAxis method to specify the groups along the X axis.

If this value is not set or Null, the Y values are plotted along the X axis in groups labeled by their order number.

The order of the data is plotted in the order of the data in the record.

By default, the labels along the X axis are populated by the value of this field.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the X axis for the chart.

Returns

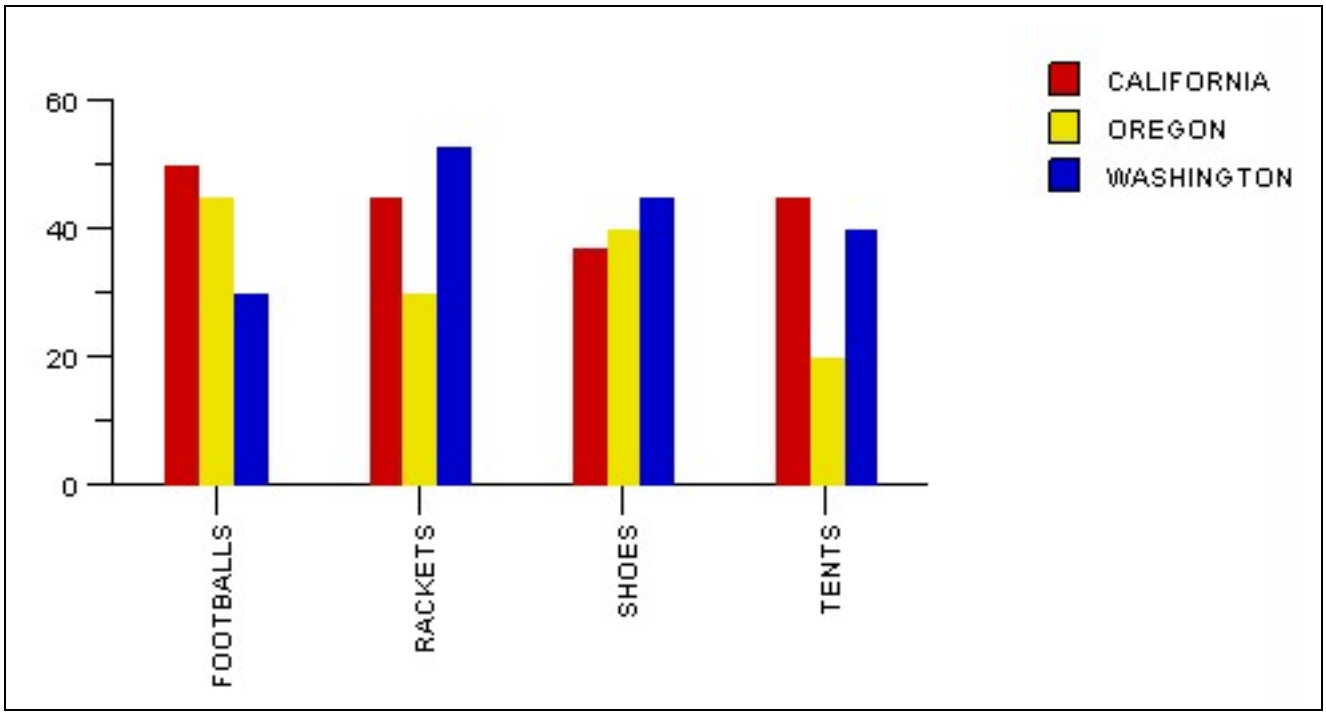
None.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

In the following example, the Y axis is set to a numeric amount, while the X axis is set to the products.

```
&oChart = GetChart(QE_CHART_DUMREC.QE_CHART_FIELD);

&oChart.SetData(Record.QE_CHART_RECORD);
&oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
&oChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);
&oChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);
```



X axis set with product data

See Also

Chapter 11, “Chart Class,” SetDataYAxis, page 352 and Chapter 11, “Chart Class,” SetData, page 346.

SetDataYAxis

Syntax

SetDataYAxis(*Record_Name.Field_Name*)

Description

Use the SetDataYAxis to specify the data to plot in the chart.

The Y axis is always considered the data axis.

The Y axis must always be numeric. If the greatest Y axis value is less than ten, the Y axis labels are given a decimal place. If the greatest Y axis value is greater than or equal to ten, the Y axis labels are displayed as integers.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. If you've specified negative numbers in the Y axis, the X axis is drawn according to the XAxisCross property.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the Y axis for the chart.

Returns

None.

Example

```
&oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
```

See Also

Chapter 11, "Chart Class," SetDataXAxis, page 351 and Chapter 11, "Chart Class," SetData, page 346.

Chart Data Properties

These properties are used to set X and Y axes data.

DataStartRow

Description

This property sets or returns the row number (after any preprocessing) at which to start plotting. This is useful when you have many rows of data in a rowset, and you want to start at a particular row. This can also be useful for creating your own 'scrolling' effect, by specifying both the start row and how many rows to be displayed (by using the DataWidth property.)

If this property is not set, then the value is 1.

This property is read-write.

See Also

Chapter 11, "Chart Class," DataWidth, page 353

DataWidth

Description

This property returns or sets the number of rows to plot for a series. The starting point for the number of rows is set with the DataStartRow property. If the DataWidth property value is not set, then the width is from the DataStartRow to the last element in the rowset. If the property value is less than zero, then it will automatically be set to one.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

Suppose your rowset returned 400 rows. You wouldn't want all 400 to display in your chart. Instead, you'd pick a subset of those rows. The row to start with is set with DataStartRow, while how many rows to display is set with DataWidth.

The following code could be used with a pushbutton connected with a chart. The pushbutton enables the next set of data to be displayed with the chart. Note that you don't need to use the Refresh method with this code example. When a method or property is used, the refresh method is automatically called.

```
Local Chart &MyChart;

&MyChart = GetChart(ChartRec.DisplayField);
&Start = &MyChart.DataStartRow;
/* display the next 20 rows of data */
&MyChart.DataStartRow = &Start + 20;
&MyChart.DataWidth = 20;
```

See Also

Chapter 11, "Chart Class," DataStartRow, page 353

ImageMap

Description

This property returns the HTML client-side image map. The image map is what makes the chart interactive.

This property is read-only.

See Also

Chapter 11, "Chart Class," Creating a Chart Using an iScript, page 407

IsTrueXY

Description

Use this property to enable a TrueXY, or numeric, axis. This causes the X axis data to be loaded as continuous linear numeric data, rather than discrete data, as is used in bar charts for example.

This property is only applicable to scatter, line or histogram charts where the X axis can be numeric.

This property is read-write.

See Also

Chapter 11, "Chart Class," Line Chart Considerations, page 335

Chapter 11, "Chart Class," TrueXY Line Chart Properties, page 377

Chart Appearance Properties

These properties are used to set the type of graph, the line style, and so on.

 Copyright © 1988-2007, Oracle. All rights reserved.

GridLines

Description

Use this property to specify whether to show grid lines with the chart.

If you don't specify a value, no grid lines are shown.

This property controls only the display of grid lines. If you want to specify the style of the grid lines, use the GridLineType property.

The values are:

Numeric Value	Constant Value	Description
0	%ChartGrid_None	Don't show grid lines (default)
1	%ChartGrid_Horizontal	Show horizontal grid lines
2	%ChartGrid_Vertical	Show vertical grid lines
3	%ChartGrid_Both	Show both vertical and horizontal grid lines

This property is read-write.

Considerations for Gantt Charts

Gantt charts only support the %ChartGrid_Vertical and %ChartGrid_None values.

Grid lines are shown within the chart section of the Gantt chart, if enabled, and are tied to the major tick mark. The major tick mark is associated with the smallest time increment displayed on the DateTime axis using the SetFormat functions such as the SetMonthFormat, SetDayFormat, and so on. For example, if you choose to display both years and months on the DateTime axis, the major tick marks denote the months.

See Chapter 11, "Chart Class," Specifying DateTime Axis Formats, page 341.

See Also

Chapter 11, "Chart Class," GridLineType, page 355

GridLineType

Description

Use this property to specify the style of the grid lines when either or both vertical or horizontal grid lines are turned on.

The default value is %ChartLine_Solid. The possible values are:

Value	Description
%ChartLine_Solid	Draw grid lines with solid lines.
%ChartLine_Dash	Draw grid lines with lines composed of dashes.
%ChartLine_Dot	Draw grid lines with lines composed of dots.
%ChartLine_DashedDot	Draw grid lines with lines composed of both dashes and dots.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Height

Description

Use this property to specify the height of the chart. This property takes a numeric value. The unit of measurement is pixels.

Generally this property is used with charts created for iScripts, to specify the height of the image, before generating the image map. You can also use it to overwrite the height set in PeopleSoft Application Designer.

If you try to read this property before setting it, the value returned is zero.

Note. The Height and Width properties must be set to the same value to change the size of a 2D Scatter chart, that is, a chart with the Type property set to %ChartType_2DScatter.

This property is read-write.

See Also

Chapter 11, "Chart Class," Height, page 356

IsDrillable

Description

Use this property to specify whether the end-user can click on the chart and trigger a PeopleCode program in the FieldChange event for that row.

By default, when the chart data comes from a record, a PeopleCode program in the FieldChange event for the Y-Data field of the clicked row is executed.

This property takes a Boolean value: True if the end-user can click on the chart to initiate an action, False otherwise.

Note. Setting this property has no effect if the IsPlainImage property is set as True.

Note. When using a chart in an iScript, this property must be used in conjunction with the SetDataURLs method. The SetDataURLs method must point to a field that is populated with the URLs to be triggered for each data point in the chart.

The default value for this property is False.

This property is read-write.

Considerations for Gantt Charts

The IsDrillable property is used with the task bars as well as the task dependency lines in the chart section of a Gantt chart. Where the user is directed depends on whether a URL is provided or not.

 Copyright © 1988-2007, Oracle. All rights reserved.

User Action	If URL is not provided:	If URL is provided:
Clicks on a task bar.	FieldChange event associated with the Planned End Date field in the Task table.	Redirects to URL provided by field identified using the SetTaskBarURL method.
Clicks on a task dependency line.	FieldChange event associated with the Child ID field in the Task Dependency table.	Redirects to URL provided by field identified using the SetTaskDependencyURL method.
Clicks on an expand/collapse icon.	FieldChange event associated with the task data.	Redirects to URL provided by field identified using the SetTaskBarURL method.

See Also

Chapter 11, "Chart Class," IsPlainImage, page 357

IsPlainImage

Description

Use this property to specify whether the chart is built without any extra HTML, such as SRC or MAP tags.

If this property is specified as True, the end-user won't be able to click on the chart to trigger an event. In addition, the pop-up data hints won't appear when the end-user passes a mouse over the chart. However, you may see a performance improvement if your chart has an extremely complicated image map and you specify this property as True.

This property takes a Boolean value: True if the chart is built without extra HTML, False otherwise.

The default value of this property is False.

This property is read-write.

LineType

Description

Use the LineType property to specify the style of series that are plotted as lines. This is for a regular series only. To specify the style of line for overlay data, use the OLLineType property.

This property takes either a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
1	%ChartLine_Solid	Draw series with a solid line.
2	%ChartLine_Dash	Draw series with a line composed of dashes.
3	%ChartLine_Dot	Draw series with a line composed of dots.
4	%ChartLine_DashedDot	Draw series with a line composed of both dashes and dots.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 11, “Chart Class,” OLLineType, page 374

RotationAngle

Description

Use this property to specify the rotation angle of the chart. This property is valid only with 3D style charts, that is, charts that have Type set as %ChartType_3DBar, %ChartType_3DStackedBar, and so on.

Note. This property has no effect on the 3D pie chart.

If you change the value of this property, you don’t have to use the Refresh method to refresh the chart.

This property takes a numeric value from 1 to 360.

This property does not take negative numbers.

The chart is rotated in a clockwise direction.

This property is read-write.

Style

Description

A *style sheet* is a definition, like a record or field definition, that you create in PeopleSoft Application Designer. It contains a collection of formatting styles, called *classes*, each of which can be applied to either the entire chart or to specific elements within a chart.

Use this property to specify the overall style of the chart by specifying a style sheet class. The value must be a valid class within the style sheet specified for the chart.

Individual style properties (such as SetXAxisStyle, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

This property is read-write.

StyleSheet

Description

Use this property to associate a style sheet with a chart.

The PTSTYLEDEF style sheet is the default style sheet associated with a chart.

Both the Style property for the chart, and individual style properties (such as SetXAxisStyle, MainTitleStyle, and so on) can be used to override this value for individual elements of the chart.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

Type

Description

Use this property to specify the type of chart that you want. You can specify either a numeric or constant value for this property. The valid values are:

Numeric Value	Constant Value	Description
0	%ChartType_2DBar	Two-dimensional bar chart
1	%ChartType_2DStackedBar	Two-dimensional stacked bar chart
2	%ChartType_2DPercentBar	Two-dimensional percent bar chart
3	%ChartType_2DHorizStackedBar	Two-dimensional stacked horizontal bar chart
4	%ChartType_2DLine	Two-dimensional line chart
5	%ChartType_2DHistogram	Two-dimensional histogram chart
6	%ChartType_2DPie	Two-dimensional pie chart
7	%ChartType_3DBar	Three-dimensional bar chart
8	%ChartType_3DStackedBar	Three-dimensional stacked bar chart
9	%ChartType_3DPie	Three-dimensional pie chart
10	%ChartType_2DHorizPercentBar	Two-dimensional horizontal percent chart
11	%ChartType_3DPercentBar	Three-dimensional percent bar char
13	%ChartType_2DHorizBar	Two-dimensional horizontal bar chart
14	%ChartType_Scatter	Scatter chart.
15	%ChartType_Gantt	Gantt chart.

This property is read-write.

See Also

Chapter 11, “Chart Class,” Chart Types, page 322

Width

Description

Use this property to specify the width of the chart. This property takes a numeric value. The unit of measurement is pixels.

Generally this property is used with charts created for iScripts, to specify the width of the image, before generating the image map. You can also use it to overwrite the width set in PeopleSoft Application Designer.

If you try to read this property before setting it, the value returned is zero.

Note. The Height and Width properties must be set to the same value to change the size of a 2D Scatter chart, that is, a chart with the Type property set to %ChartType_2DScatter.

This property is read-write.

See Also

Chapter 11, “Chart Class,” Width, page 359

Copyright © 1988-2007, Oracle. All rights reserved.

Chart Colors

The following lists all the values you can use with the chart color methods. You can use either the numeric or constant value.

Numeric Value	Constant Value	Description
-1	%ChartColor_Series_Default	Default series color. When a data point is set to this, the default series color or the color that has been set by the user for that series is used instead. This value works only with the SetColorArray method.
0	%ChartColor_Black	Black
1	%ChartColor_Blue	Blue
2	%ChartColor_Cyan	Cyan
3	%ChartColor_DarkGray	DarkGray
4	%ChartColor_Gray	Gray
5	%ChartColor_Green	Green
6	%ChartColor_LightGray	LightGray
7	%ChartColor_Magenta	Magenta
8	%ChartColor_Orange	Orange
9	%ChartColor_Pink	Pink
10	%ChartColor_Red	Red
11	%ChartColor_White	White
12	%ChartColor_Yellow	Yellow
13	%ChartColor_Red_Orange	Red_Orange
14	%ChartColor_Yellow_Green	Yellow_Green

Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
15	%ChartColor_Blue_Violet	Blue_Violet
16	%ChartColor_Purple	Purple
17	%ChartColor_Yellow_Orange	Yellow_Orange

Chart Color Methods

These methods are used to set and manipulate colors for either a series or for each point.

SetColorArray

Syntax

SetColorArray(*&Array_of_Color*)

Description

Use the SetColorArray method to set the colors for a series. You can specify only a one-dimensional array for the parameter. This method applies only to base chart data, not to the overlay data.

The type of array can be any of the following:

- String
- Number
- Integer
- Decimal

SetDataColor does not work with line charts. Individual line segments can not be colored as there are less line segments than points. SetColorArray must be used instead to color each complete line (series).

Parameters

Parameter	Description
&Array_of_Color	Specify an already instantiated array that contains the color values that you want for the series. You can specify either a constant or a number.

See Chapter 11, "Chart Class," Chart Colors, page 360.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example sets the four series in the chart:

```
&NumArray = CreateArray(1, 3, 6, 9);

&oChart.SetColorArray(&NumArray);
```

See Also

Chapter 8, "Array Class," page 215

SetDataColor

Syntax

SetDataColor(*Record_Name.Field_Name*)

Description

Use the SetDataColor method to specify a field that contains the color for each data point. Each field for each datapoint must contain either a number or a constant, specifying the color you want to use.

SetDataColor does not work with line charts. Individual line segments can not be colored as there are less line segments than points. SetColorArray must be used instead to color each complete line (series).

Note. This method has no effect when used with an overlay.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the colors for each data point. You can specify either a constant or a number.

See Chapter 11, "Chart Class," Chart Colors, page 360.

Returns

None.

Chart Axes Methods

These methods are used to further control the appearance of the axes.

SetXAxisLabels

Syntax

SetXAxisLabels(*&Array_of_Any*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetXAxisLabels method to specify an array of labels for the X axis. By default, the labels are populated from the record field specified by SetDataXAxis. Labels specified with SetXAxisLabels overwrite the default labels.

Note. Default label text is automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

Parameters

Parameter	Description
&Array_of_Any	Specify an already instantiated array of any that contain the labels that you want to use for the X Axis.

See Also

Chapter 11, "Chart Class," SetDataXAxis, page 351

Chapter 8, "Array Class," page 215

Chapter 11, "Chart Class," Tick Mark Considerations, page 334

SetYAxisLabels

Syntax

```
SetYAxisLabels(&Array_of_Any)
```

Description

Use the SetYAxisLabels method to specify an array of labels for the Y axis. Labels specified with SetYAxisLabels overwrite the default labels.

If the greatest Y axis value is less than ten, the default Y axis labels are given a decimal place. If the greatest Y axis value is greater than or equal to ten, the default Y axis labels are displayed as integers.

Note. Default label text is automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

If you set the labels yourself, you may need to set the YAxisTicks and YAxisMax because you no longer have numbers that correlate to the data points.

Parameters

Parameter	Description
&Array_of_Any	Specify an already instantiated array of any that contain the labels that you want to use for the Y Axis.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 11, “Chart Class,” SetDataXAxis, page 351, Chapter 11, “Chart Class,” YAxisTicks, page 367 and Chapter 11, “Chart Class,” YAxisMax, page 366.

Chapter 8, “Array Class,” page 215

Chapter 11, “Chart Class,” Tick Mark Considerations, page 334

Chart Axes Properties

These properties are used to further control the appearance of the axes.

XAxisCross

Description

Use this property to specify the appearance of the X axis data when there are both negative and positive values.

You can specify either a numeric value or a constant for this property. Values are:

Numeric Value	Constant Value	Description
0	%ChartXAxisCross_Zero	The X axis extrudes from the zero point on the Y axis.
–1	%ChartXAxisCross_Bottom	The X axis extrudes from the Y minimum on the Y axis.
–2	%ChartXAxisCross_ZeroLineBttm	The X axis extrudes from the Y minimum on the Y axis, and a line is drawn from the zero point on the Y axis.
-3	%ChartXAxisCross_ZeroLblsBttm	The X axis extrudes from the Y minimum on the Y axis.

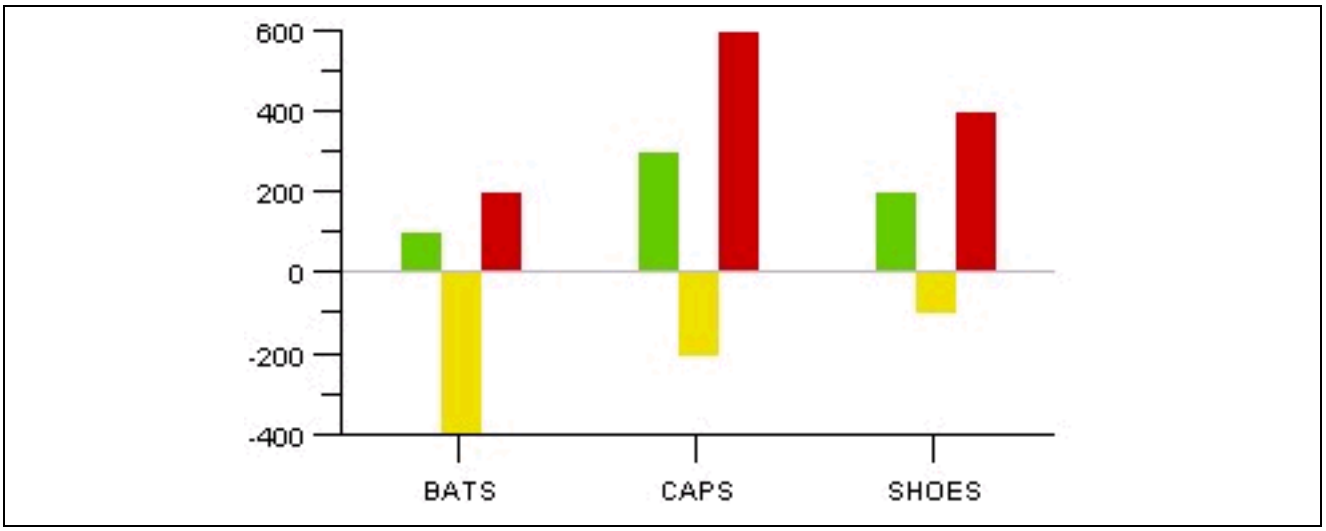
Example

The following code:

```
&MyChart.XAxisCross = %ChartXAxisCross_ZeroLineBttm;
```

produces a chart similar to this:

Copyright © 1988-2007, Oracle. All rights reserved.



Example of %ChartXAxisCross_ZeroLineBttm

XAxisLabelOrient

Description

Use this property to specify whether the X axis label is displayed horizontally or vertically.

The default value is horizontal.

You can specify either a numeric value or a constant for this property. Values are:

Numeric Value	Constant Value	Description
0	%ChartText_Horizontal	The X axis label is displayed horizontally.
90	%ChartText_Vertical	The X axis label is displayed vertically

This property is read-write.

XAxisStyle

Description

Use this property to specify the style of the X axis. The values for this property are the style classes contained in the style sheet associated with the chart.

Note. Pie chart labels take their style from this property.

This property is read-write.

See Also

Chapter 11, “Chart Class,” Style Sheets and Charts, page 331

Copyright © 1988-2007, Oracle. All rights reserved.

XAxisTicks

Description

Use this property to specify the number of minor tick marks along the X axis. This property takes a numeric value. To control the major tick marks use the XAxisScaleResolution method.

This property is read-write.

See Also

Chapter 11, "Chart Class," XAxisScaleResolution, page 378

Chapter 11, "Chart Class," Tick Mark Considerations, page 334

YAxisLabelOrient

Description

Use this property to specify whether the Y axis label is displayed horizontally or vertically.

The default value is horizontal.

You can specify either a numeric value or a constant for this property. Valid values are:

Numeric Value	Constant Value	Description
0	%ChartText_Horizontal	The Y axis label is displayed horizontally.
90	%ChartText_Vertical	The Y axis label is displayed vertically

This property is read-write.

YAxisMax

Description

Use this property to specify the maximum value of the Y (data) axis. This property takes a numeric value.

The maximum value sets the largest value that is displayed on the axis.

Note. If the YAxisMax and YAxisMin properties are not set, or are set to too narrow a range, ticks may not appear on the Y axis.

This property is read-write.

YAxisMin

Description

Use this property to specify the minimum value of the X axis. This property takes a numeric value.

The minimum value sets the point at which the chart plots in positive and negative directions.

The default minimum value is the default value of the Y axis.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. If the YAxisMax and YAxisMin properties are not set, or are set to too narrow a range, ticks may not appear on the Y axis.

This property is read-write.

YAxisStyle

Description

Use this property to specify the style of the Y axis. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Also

Chapter 11, “Chart Class,” Style Sheets and Charts, page 331

YAxisTicks

Description

Use this property to specify the number of minor tick marks along the Y (data) axis. This property takes a numeric value. To control the major tick marks use the YAxisScaleResolution method.

This property is read-write.

See Also

Chapter 11, “Chart Class,” YAxisScaleResolution, page 379

Chapter 11, “Chart Class,” Tick Mark Considerations, page 334

Chart Title Properties

These properties are used to set the text and style of titles for the graph and the axes.

MainTitle

Description

Use this property to specify the text for the main title of the chart.

This property takes a string value.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

MainTitleOrient

Description

Use this property to specify the orientation of the text for the main title.

The default value is horizontal.

You can specify either a number or a constant for this property. The values are:

Numeric Value	Constant Value	Description
0	%ChartText_Horizontal	Title text is displayed horizontally.
90	%ChartText_Vertical	Title text is displayed vertically.

This property is read-write.

MainTitleStyle

Description

Use this property to specify the style of the main title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Also

Chapter 11, “Chart Class,” Style Sheets and Charts, page 331

XAxisTitle

Description

Use this property to specify the text for the X axis title.

This property is read-write.

XAxisTitleOrient

Description

Use this property to specify the orientation of the text for the X axis title.

You can specify either a number or a constant for this property. The values are:

Numeric Value	Constant Value	Description
0	%ChartText_Horizontal	Title text is displayed horizontally.
90	%ChartText_Vertical	Title text is displayed vertically. This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

XAxisTitleStyle

Description

Use this property to specify the style of the X axis title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Also

Chapter 11, "Chart Class," Style Sheets and Charts, page 331

YAxisTitle

Description

Use this property to specify text of the title for the Y axis.

This property is read-write.

YAxisTitleOrient

Description

Use this property to specify the orientation of the text for the Y axis title.

You can specify either a number or a constant for this property. The values are:

Numeric Value	Constant Value	Description
0	%ChartText_Horizontal	Title text is displayed horizontally.
90	%ChartText_Vertical	Title text is displayed vertically. This property is read-write.

YAxisTitleStyle

Description

Use this property to specify the style of the Y axis title. The values for this property are the style classes contained in the style sheet associated with the chart.

This property is read-write.

See Also

Chapter 11, "Chart Class," Style Sheets and Charts, page 331

Chart Legend Methods

These methods are used for each series legend.

Copyright © 1988-2007, Oracle. All rights reserved.

SetLegend

Syntax

SetLegend(*&Array_of_String*)

Description

Use the SetLegend method to specify alternative legends. By default, the legends are populated from the record field specified with SetDataSeries.

This method is used to write both the overlay legend and the series legend. The labels are overwritten in the order of elements in the array, that is, the first element overwrites the first series, the second overwrites the second, and so on, with the last element in the array overwriting the overlay legend. There can be more than one series in an overlay.

If you don't specify an element for an array (that is, a blank or a null) then no legend is listed for that series.

Note. Default label text is automatically translated. If you set your own labels, be sure to use translated text, such as message catalog entries.

Parameters

Parameter	Description
&Array_of_String	Specify an already instantiated array of string, containing the text that you want to use for the legend.

Returns

None.

Example

The following example displays a legend containing only the overlay.

```
&LegendArray = CreateArray("", "", "Revenue");

&MyChart.SetLegend(&LegendArray);
```

See Also

Chapter 11, "Chart Class," SetDataSeries, page 350

Chapter 8, "Array Class," page 215

Chart Legend Properties

These properties are used for each series legend.

 Copyright © 1988-2007, Oracle. All rights reserved.

HasLegend

Description

Use this property to specify if a legend is displayed with the chart. This property takes a Boolean value: True, if the legend is displayed with the chart, False otherwise.

The default value is False.

This property is read-write.

LegendMaxEntries

Description

Use this property to specify the number of legend entries before a new column (or row) is created.

When the legend position is specified as left or right, the legend is drawn vertically so a new column is created when this property is exceeded.

When the legend position is specified as top or bottom, the legend is drawn horizontally and a new row is created when this property is exceeded.

The default value of this property is five.

This property is read-write.

LegendPosition

Description

Use this property to specify where the legend should appear, in relationship to the chart. You can specify either a numeric or constant value for this property.

Note. ChartLegend_Separate generates a legend without a chart. The legend takes over the entire charting area. The legend appears in the center of the chart.

The values are:

Numeric Value	Constant Value	Description
0	%ChartLegend_Left	Display legend to the left of the chart.
1	%ChartLegend_Right	Display legend to the right of the chart.
2	%ChartLegend_Top	Display legend on top of the chart.
3	%ChartLegend_Bottom	Display legend below the chart.
4	%ChartLegend_Separate	Generate a legend without a chart. This property is read-write.

LegendStyle

Description

Use this property to specify the style of the legend. The values for this property are the style classes contained in the style sheet associated with the chart.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

See Also
Chapter 11, "Chart Class," Style Sheets and Charts, page 331

Chart Overlay Methods

These methods are used to set and manipulate overlay data.

SetOLData

Syntax
SetOLData({*Record_Name* | *&Rowset*})

Description

Use the SetOLData method to specify where the data for the overlay is coming from. If you make a change to the underlying data of a chart, use the Refresh method to update the chart.

There are no default values for the overlay data. You must specify the data.

It is possible to specify two different Y axis fields, however, you must make certain that different data plotted against the same axis makes sense. For example, plotting two dollar amounts would probably be okay. However, plotting dollars against the number of sales, and sharing the same axis, though technically it would work, it wouldn't look correct.

Parameters

Parameter	Description
Record_Name	Specify the name of a record to be used to populate the overlay of the chart with data.
&Rowset	Specify the name of an already instantiated rowset to populate the overlay of the chart with data.

See Also
Chapter 11, "Chart Class," SetDataXAxis, page 351, Chapter 11, "Chart Class," SetDataYAxis, page 352 and Chapter 11, "Chart Class," SetData, page 346.

SetOLDataSeries

Syntax
SetOLDataSeries(*Record_Name.Field_Name*)

Description

Use the SetOLDataSeries method to specify the name of the field containing the overlay series values. Every distinct value in this field is considered a unique series.

The series is plotted in the order given by the rows in the record.

If this value is Null, the system assumes there is only one series to plot.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the series values of the overlay for the chart.

See Also
Chapter 11, "Chart Class," SetDataSeries, page 350

SetOLDataXAxis

Syntax
SetOLDataXAxis(*Record_Name.Field_Name*)

Description

Use the SetOLDataXAxis method to specify the groups along the X axis if the axis is non-numeric for the overlay.

If the X axis is numeric, this method is used to give the position of the points along the axis. Only discrete values are supported for the X axis.

If this value is Null, the Y values are plotted along the X axis in groups labeled by their order number.

The order of the data is plotted in the order of rows in the record.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the X axis for the overlay of the chart.

SetOLDataYAxis

Syntax
SetOLDataYAxis(*Record_Name.Field_Name*)

Description

Use the SetOLDataYAxis to specify the data to plot for the overlay of the chart.

This value must always be numeric.

There is no default value for this method.

Parameters

Parameter	Description
Record_Name.Field_Name	Specify the name of the record, and the field on that record, that contains the data for the overlay of the chart.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 11, “Chart Class,” SetDataXAxis, page 351 and Chapter 11, “Chart Class,” SetDataYAxis, page 352.

Chart Overlay Properties

These properties are used to set and manipulate overlay data.

OLLineType

Description

Use the OLLineType property to specify the style of an overlay line.

This property takes either a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
1	%ChartLine_Solid	Draw series with a solid line.
2	%ChartLine_Dash	Draw series with a line composed of dashes.
3	%ChartLine_Dot	Draw series with a line composed of dots.
4	%ChartLine_DashedDot	Draw series with a line composed of both dashes and dots. This property is read-write.

See Also
Chapter 11, “Chart Class,” LineType, page 357

OLType

Description

Use this property to specify the style of the overlay line.

You can specify either a numeric or a constant value for this property. The values are:

Numeric Value	Constant Value	Description
4	%ChartType_2DLine	Overlay is drawn as a 2D line.
5	%ChartType_2DHistogram	Overlay is drawn as a 2D histogram.

This property is read-write.

Scatter Chart Methods

The methods in this section are only applicable when the chart has been specified as a 2D Scatter chart.

 Copyright © 1988-2007, Oracle. All rights reserved.

SetDataAnnotations

Syntax

SetDataAnnotation(*Record_Name.Field_Name*)

Description

Use the SetDataAnnotations method to specify an optional text label that can be associated with each point in the chart.

Parameters

Parameter	Description
Recordname.Fieldname	Specify the field name (and the record it's associated with) that contains the text for the optional label.

See Also

Chapter 11, "Chart Class," SetOLDataAnnotations, page 375

SetDataGlyphScale

Syntax

SetDataGlyphScale(*Record_Name.Field_Name*)

Description

Use the SetDataGlyphScale method to specify a field that contains numerical data defining the size of each glyph. These values are mapped to a range of size values between a predefined minimum and maximum.

Parameters

Parameter	Description
Recordname.Fieldname	Specify the field (and the record associated with it) that contains numerical data used to define the size of each glyph.

See Also

Chapter 11, "Chart Class," SetOLDataGlyphScale, page 376

SetOLDataAnnotations

Syntax

SetOLDataAnnotation(*Record_Name.Field_Name*)

Description

Use the SetOLDataAnnotations method to specify an optional text label that can be associated with each point in the overlay of the chart.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Recordname.Fieldname	Specify the field name (and the record it's associated with) that contains the text for the optional label.

See Also

Chapter 11, "Chart Class," SetDataAnnotations, page 375

SetOLDataGlyphScale

Syntax

SetOLDataGlyphScale(*Record_Name.Field_Name*)

Description

Use the SetOLDataGlyphScale method to specify a field that contains numerical data defining the size of each glyph in the overlay chart. These values are mapped to a range of size values between a predefined minimum and maximum.

Parameters

Parameter	Description
Recordname.Fieldname	Specify the field (and the record associated with it) that contains numerical data used to define the size of each glyph.

See Also

Chapter 11, "Chart Class," SetDataGlyphScale, page 375

Scatter Chart Property

The property in this section is only applicable when the chart has been specified as a 2D Scatter chart.

ShowCrossHair

Description

Use the ShowCrossHair property to specify whether the chart should be split into quadrants.

This property takes a boolean value. If you specify true, two lines are drawn, one from the middle of the X axis and one from the middle of the Y axis.

 Copyright © 1988-2007, Oracle. All rights reserved.

TrueXY Line Chart Properties

The properties in this section are only applicable when the chart has been specified as a TrueXY line chart, that is, if the IsTrueXY property has been set to true.

XAxisCrossPoint

Description

Use the XAxisCrossPoint property to specify the point at which the X axis intersects a numeric Y axis. The value is in the coordinate system of the Y axis scale.

This property is read-write.

See Also

Chapter 11, “Chart Class,” YAxisCrossPoint, page 378

XAxisMax

Description

The maximum value of the X axis.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

XAxisMin

Description

The minimum value of the X axis.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

XAxisPrecision

Description

Use the XAxisPrecision property to control the number of decimal places displayed in the labels of a numeric X axis.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 11, “Chart Class,” YAxisPrecision, page 379

XAxisScaleResolution

Description

Use the XAxisScaleResolution property to control the level of detail displayed on a numeric axis. Use this property to increase or reduce the number of major tick marks and labels on the X axis.

This property is only applicable to scatter, line, bubble, or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

The values are:

Numeric Value	Constant Value	Description
0	%ChartAxisResolution_Fine	Specify fine resolution for the chart, that is, increase the number of major ticks and labels on the X axis. The fine scale resolution uses extra tick marks to give a more detailed division of the axis. This means that there may not be a one-to-one relationship between the axis labels and the tick marks.
1	%ChartAxisResolution_Coarse	Specify coarse resolution for the chart, that is, reduce the number of major ticks and labels on the X axis. The coarse scale value maintains a one-to-one relationship between the axis labels and the tick marks.

This property is read-write.

See Also

Chapter 11, “Chart Class,” YAxisScaleResolution, page 379

YAxisCrossPoint

Description

Use the YAxisCrossPoint property to specify the point at which the Y axis intersects a numeric X axis. The value is in the coordinate system of the X axis scale.

This property is only applicable to scatter, line or histogram charts where the X axis is numeric, that is, the IsTrueXY property has been set to true.

This property is read-write.

See Also

Chapter 11, “Chart Class,” XAxisCrossPoint, page 377

 Copyright © 1988-2007, Oracle. All rights reserved.

YAxisMax

Description

The maximum value of the Y axis.

This property is only applicable to scatter, line, or histogram charts.

This property is read-write.

YAxisMin

Description

The minimum value of the Y axis.

This property is only applicable to scatter, line, or histogram charts.

This property is read-write.

YAxisPrecision

Description

Use the YAxisPrecision property to control the number of decimal places displayed in the labels of a numeric Y axis.

This property is only applicable to scatter, line or histogram charts.

This property is read-write.

See Also

Chapter 11, “Chart Class,” XAxisPrecision, page 377

YAxisScaleResolution

Description

Use the YAxisScaleResolution property to control the level of detail displayed on a numeric axis. Use this property to increase or reduce the number of major ticks and labels on the Y axis.

This property is only applicable to scatter, line, or histogram charts.

The values are:

Numeric Value	Constant Value	Description
0	%ChartAxisResolution_Fine	Specify fine resolution for the chart, that is, increase the number of major ticks and labels on the Y axis. The fine scale resolution uses extra tick marks to give a more detailed division of the axis. This means that there may not be a one-to-one relationship between the axis labels and the tick marks.

Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
1	%ChartAxisResolution_Coarse	Specify coarse resolution for the chart, that is, reduce the number of major ticks and labels on the Y axis. The coarse scale value maintains a one-to-one relationship between the axis labels and the tick marks.

This property is read-write.

See Also

Chapter 11, "Chart Class," XAxisScaleResolution, page 378

Gantt Chart Methods

The methods in this section are only applicable when the chart has been specified as a Gantt chart.

SetActualEndDate

Syntax

SetActualEndDate(*RecordName.FieldName*)

Description

Use the SetActualEndDate method to specify the field in the record that defines the actual end date. This is an optional method, and is used in conjunction with the SetPlannedEndDate method. If an actual end date is specified, then an actual start date must also be specified. When defined, the actual or baseline tasks are visually shown as a separate task bar and are layered beneath the planned task if the two overlap.

Note. The Gantt chart logs a warning to the file PSJChart.log for any task that has a bad date format in this method or the SetActualStartDate method, but continues with its normal processing. For these tasks, the overlayed baseline information isn't displayed in the chart section.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the actual end date. This field must be of DateTime or Date type.

See Also

Chapter 11, "Chart Class," SetActualStartDate, page 380 , Chapter 11, "Chart Class," SetPlannedEndDate, page 382 and Chapter 11, "Chart Class," SetPlannedStartDate, page 382.

SetActualStartDate

Syntax

SetActualStartDate(*RecordName.FieldName*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetActualStartDate method to specify the field in the record that defines the actual start date. This is an optional method, and is used in conjunction with the SetPlannedStartDate method. If an actual start date is specified, then an actual end date must also be specified. When defined, the actual or baseline tasks are visually shown as a separate task bar and are layered beneath the planned task if the two overlap.

Note. The Gantt chart logs a warning to the file PSJChart.log for any task that has a bad date format in this method and the SetActualEndDate method, but continues with its normal processing. For these tasks, the overlayed baseline information is displayed in the chart section.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the actual start date. This field must be of DateTime or Date type.

See Also

Chapter 11, “Chart Class,” SetActualEndDate, page 380 , Chapter 11, “Chart Class,” SetPlannedEndDate, page 382 and Chapter 11, “Chart Class,” SetPlannedStartDate, page 382.

SetActualTaskBarColor

Syntax

SetActualTaskBarColor(*RecordName.FieldName*)

Description

Use the SetActualTaskBarColor method to specify the field in the record that defines the colors for either the actual task bar or the actual milestone glyph.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the task bar color. This field must be of type number, which means you can only use the numeric value for the color. See Chapter 11, “Chart Class,” Chart Colors, page 360.

See Also

Chapter 11, “Chart Class,” SetPlannedTaskBarColor, page 383 and Chapter 11, “Chart Class,” TaskMilestoneGlyph, page 396.

SetChartArea

Syntax

SetChartArea(*Percentage*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetChartArea method to specify what percentage of the Gantt chart is allocated to displaying the chart section. By default, both the task and chart sections are allocated 50% of the available space provided by the Gantt chart.

Parameters

Parameter	Description
Percentage	Specify the percentage of the page space for a Gantt chart that is allocated for displaying the chart area. PeopleSoft recommends setting this value above .20 or 20%. Any point below .20 may cause the task bars and task dependency lines to become compressed and hard to read. If the percentage is set less than or equal to .10 or 10%, this parameter automatically takes a value of 0, enabling the table area to occupy the entire Gantt chart area. This parameter takes a float value.

SetPlannedEndDate

Syntax

SetPlannedEndDate(*RecordName.FieldName*)

Description

Use the SetPlannedEndDate method to specify the field in the record that defines the planned end date. If planned versus actual dates are not used, this method specifies the end date of the task.

This method is required when creating a Gantt chart.

Note. The Gantt chart logs an error to the file PSJChart.log and stops its normal processing if it finds a bad date format in any of the rows in the field associated with this method and the SetPlannedStartDate method.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the planned end date. This field must be of DateTime or Date type.

See Also

Chapter 11, “Chart Class,” SetActualEndDate, page 380 , Chapter 11, “Chart Class,” SetActualStartDate, page 380 and Chapter 11, “Chart Class,” SetPlannedStartDate, page 382.

SetPlannedStartDate

Syntax

SetPlannedStartDate(*RecordName.FieldName*)

Description

Use the SetPlannedStartDate method to specify the field in the record that defines the planned start date. If planned versus actual dates are not used, this method specifies the start date of the task.

 Copyright © 1988-2007, Oracle. All rights reserved.

This method is required when creating a Gantt chart.

Note. The Gantt chart logs an error to the file PSJChart.log and stops its normal processing if it finds a bad date format in any of the rows in the field associated with this method and the SetPlannedEndDate method.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that contains the value used to define the planned start date. This field must be of DateTime or Date type.

See Also

Chapter 11, "Chart Class," SetActualEndDate, page 380 , Chapter 11, "Chart Class," SetPlannedEndDate, page 382 and Chapter 11, "Chart Class," SetPlannedStartDate, page 382.

SetPlannedTaskBarColor

Syntax

SetPlannedTaskBarColor(*RecordName.FieldName*)

Description

Use the SetPlannedTaskBarColor method to specify the field in the record that defines the colors for either the planned task bar or the planned milestone glyph.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the task bar color. This field must be of type number, which mean you can only use the numeric values for specifying color. See Chapter 11, "Chart Class," Chart Colors, page 360.

See Also

Chapter 11, "Chart Class," SetActualTaskBarColor, page 381 and Chapter 11, "Chart Class," TaskMilestoneGlyph, page 396.

SetTableXScrollbar

Syntax

SetTableXScrollbar(*Scroll_ArrowAmount*)

Description

Use the SetTableXScrollbar method to specify the amount that the scroll bar in the table section of the Gantt chart scroll data when the user clicks on the arrows or scrollbar.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Scroll_ArrowAmount	Specify the amount to scroll in the table section of the Gantt chart when the user clicks on the horizontal scrollbar arrow. The scroll amount must be a value between zero and one. The default value is 0.10. The amount specified indicates the percentage of the table section that is scrolled when the user clicks the scroll bar.

SetTaskAppData

Syntax

```
SetTaskAppData(RecordName.FieldName1 [, RecordName.FieldName2
[,RecordName.FieldName3. . .]])
```

Description

Use the SetTaskAppData method to specify the fields in the record that define the application data to be viewed in the table section.

In the table section, column one (or the leftmost column) always displays the work breakdown structure (WBS) numbering (if given) and name of the task (or task ID if the name is not given). This method allows applications to define additional task data columns to be displayed in the rightmost columns of the table section.

Parameters

Parameter	Description
RecordName.FieldNames	Specify the fields and their associated record that defines one or more application specific fields to be displayed as a column in the table section. The order in which the fields are given as parameters defines their column order in the table section. Because the task name (or task ID if the name is not given) is always shown in column one, the first record field value is displayed in column two.

See Also

Chapter 11, “Chart Class,” SetTaskAppDataTitles, page 384 , Chapter 11, “Chart Class,” SetTaskData, page 385 and Chapter 11, “Chart Class,” SetTaskDependencyData, page 386.

SetTaskAppDataTitles

Syntax

```
SetTaskAppDataTitles(&TitleArray)
```

Description

Use the SetTaskAppDataTitles method to specify the column titles to be displayed in the table section, starting with column two. The title for column one (that is, the leftmost column) in the table section is set using the SetTaskTitle method.

The length of the array should match the number of application data fields specified with the SetTaskAppData method. Additional strings beyond the number of application data fields are not displayed. If the number of strings in the array specified by this method is less than the number of application data fields, the blanks are displayed in these title columns.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&TitleArray	Specify an array of strings containing the column titles to be displayed in the table section.

See Also

Chapter 11, "Chart Class," SetTaskAppData, page 384 and Chapter 11, "Chart Class," TaskTitle, page 396.

SetTaskBarURL

Syntax

```
SetTaskBarURL(RecordName.FieldName)
```

Description

Use the SetTaskBarURL method to specify the field in the record which defines the URL the browser redirects to when the user clicks on a task bar.

By default, the FieldChange event on the planned end date field of the Gantt task data record is triggered if the image map based interactivity is turned on (using the IsDrillable property). This can be overriden by specifying a URL that is triggered when the task bar is clicked.

Use the SetTaskDependencyURL method to specify the URL to be used when the user clicks on a dependency line in the chart section.

Parameters

Parameter	Description
Record.Fieldname	Specify the field, and its associated record, that defines the URL the browser is redirected to when the task bar is clicked. This field must be of type character. The URL must be an absolute URL.

See Also

Chapter 11, "Chart Class," SetTaskDependencyURL, page 388

Chapter 11, "Chart Class," IsDrillable, page 356

SetTaskData

Syntax

```
SetTaskData({Record.RecordName | &Rowset})
```

Description

Use the SetTaskData method to specify from where the task data is to be populated, either from a record or an already instantiated rowset.

Note. If you specify a rowset, the data in the rowset must be provided in the correct display order.

This method is required when you're creating a Gantt chart.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RecordName | *&Rowset*	Specify the record or rowset that contains the task data. If you specify a record, you must prefix the record name with the keyword **Record.**

See Also

Chapter 11, “Chart Class,” SetTaskDependencyData, page 386

SetTaskDependencyChildID

Syntax

SetTaskDependencyChildID(*RecordName.FieldName*)

Description

The *dependent* task is the task where the dependency arrow ends. Use the SetTaskDependencyChildID method to specify the field in the record that defines the task ID for the dependent task.

The ID must match up to an ID in the Task data set.

This method is required if you specify dependency data.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the task ID for the dependant task. This field must be of type number.

See Also

Chapter 11, “Chart Class,” SetTaskDependencyData, page 386 , Chapter 11, “Chart Class,” SetTaskDependencyURL, page 388 and Chapter 11, “Chart Class,” SetTaskDependencyParentID, page 388.

SetTaskDependencyData

Syntax

SetTaskDependencyData({*RecordName* | *&Rowset*})

Description

Use the SetTaskDependencyData method to specify either a record, or an already instantiated rowset, that contains the dependency data.

Note. If you specify a rowset, the data in the rowset must be provided in the correct display order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RecordName | *&Rowset*	Specify the record or rowset that contains the task data. If you specify a record, you must prefix the record name with the keyword **Record.**

See Also

Chapter 11, "Chart Class," SetTaskData, page 385 , Chapter 11, "Chart Class," SetTaskDependencyChildID, page 386 , Chapter 11, "Chart Class," SetTaskDependencyURL, page 388 and Chapter 11, "Chart Class," SetTaskDependencyParentID, page 388.

SetTaskDependencyType

Syntax

SetTaskDependencyType(*RecordName.FieldName*)

Description

Use the SetTaskDependencyType method to specify the type of dependency relationship between predecessor and successor tasks.

The values for type are as follows:

Type	Description
Finish-to-Start	The arrow starts at the end of the predecessor task bar and ends at the start of the successor task bar. This is the default value
Finish-to-Finish	The arrow starts at the end of the predecessor task bar and ends at the end of the successor task bar.
Start-to-Start	The arrow starts at the start of the predecessor task bar and ends at the start of the successor task bar.
Start-to-Finish	The arrow starts at the start of the predecessor task bar and ends at the end of the successor task bar.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the type of dependencies between predecessor and successor tasks. This field must be of type integer, and one of the following values:

Value	Description
0 or any value greater than 3	Finish-to-Start
1	Finish-to-Finish
2	Start-to-Start

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
3	Start-to-Finish

See Also
Chapter 11, "Chart Class," SetTaskDependencyChildID, page 386, Chapter 11, "Chart Class," SetTaskDependencyData, page 386 and Chapter 11, "Chart Class," SetTaskDependencyParentID, page 388.

SetTaskDependencyURL

Syntax
SetTaskDependencyURL(*RecordName.FieldName*)

Description

Use the SetTaskDependencyURL method to specify the field in the record that defines the URL the browser redirects to when the user clicks on a task bar.

By default, the FieldChange event on the child ID field of the task dependency record is triggered if the image map based interactivity is turned on (using the IsDrillable property). This can be overriden by specifying a URL that is triggered when the task bar is clicked.

Use the SetTaskBarURL method to specify the URL to be used when the user clicks on a task bar in the chart section.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the URL the browser is redirected to when the dependency arrow is clicked. This field must be of type character. The URL must be an absolute URL.

See Also
Chapter 11, "Chart Class," SetTaskBarURL, page 385 , Chapter 11, "Chart Class," SetTaskDependencyChildID, page 386 , Chapter 11, "Chart Class," SetTaskDependencyData, page 386 and Chapter 11, "Chart Class," SetTaskDependencyParentID, page 388.

SetTaskDependencyParentID

Syntax
SetTaskDependencyParentID(*RecordName.FieldName*)

Description

The *depender* task is the task where the dependency arrow originates. Use the SetTaskDependencyParentID method to specify the field in the record that defines the task ID for the depender task.

The ID must match up to an ID in the Task data set.

This method is required if you specify dependency data.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that defines the task ID for the depender task. This field must be of type number.

See Also

Chapter 11, "Chart Class," SetTaskDependencyChildID, page 386 and Chapter 11, "Chart Class," SetTaskDependencyData, page 386.

SetTaskExpanded

Syntax

```
SetTaskExpanded((RecordName.FieldName)
```

Description

Use the SetTaskExpanded method to specify the field in the record that defines the expanded state of parent tasks.

Whenever an end user changes the expanded state by clicking on the expand (collapse) icon in the table section, the expanded state in the rowset is updated. In addition, a field change event occurs to allow applications to synchronize expanded state changes with other parts of their application.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the text in the task hint popup. This field must be of type character, that is, Y or N.

SetTaskHints

Syntax

```
SetTaskHints(RecordName.FieldName)
```

Description

Use the SetTaskHints method to specify the field in the record which defines the task hint popup text. The description is displayed in a popup bubble when a cursor hovers over the task bar or milestone glyph.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the text in the task hint popup. This field must be of type character.

See Also

Chapter 11, "Chart Class," SetTaskID, page 390 and Chapter 11, "Chart Class," SetTaskName, page 392.

Copyright © 1988-2007, Oracle. All rights reserved.

SetTaskID

Syntax

SetTaskID(*Recordname.FieldName*)

Description

Use the SetTaskID method to specify the field that defines the unique task identifier. The task ID is used to support task linking and dependencies.

This method is required when you're creating a Gantt chart.

Parameters

Parameter	Description
RecordName.FieldName	Specify the record name and the field name of the field that defines the unique task identifier. The field must be of type number.

See Also

Chapter 11, "Chart Class," SetTaskName, page 392

SetTaskLabel

Syntax

SetTaskLabel(*RecordName.FieldName*)

Description

Use the SetTaskLabel method to specify the field in the record that defines the task label. The label is displayed alongside its corresponding task bar in the chart area. If a task label is not provided, the task name is used instead. If a task name is not provided, the task id is used.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that contains the task label. This field must be of type character.

See Also

Chapter 11, "Chart Class," SetTaskID, page 390 and Chapter 11, "Chart Class," SetTaskName, page 392.

SetTaskLevel

Syntax

SetTaskLevel(*RecordName.FieldName*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetTaskLevel method to specify the field in the record that defines the level for the task. The outermost summary tasks are always defined as task level one. Tasks with task levels greater than one are subtasks. Subtasks may also contain other subtasks.

If a field is not provided to define the level, all tasks are defined at level 1. A maximum of 32 levels are supported.

The following table shows an example of the different levels that could be used, as well as what the parent task level is.

Note. The parent task is not actually part of the data.

Task Name	Task Level	Parent Level
Phase I	1	None
Evaluation	2	Phase I
Research	3	Evaluation
Report Findings	3	Evaluation
Create Budget	2	Phase I
Approve Budget	3	Create Budget
Phase II	1	None
Code and Test	2	Phase II
Phase III	1	None
Training	2	Phase III

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that contains the information about the task level. This field must be of type number. Up to 32 levels are supported.

See Also

Chapter 11, “Chart Class,” SetTaskID, page 390

SetTaskMilestone

Syntax

SetTaskMilestone(*RecordName.FieldName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetTaskMilestone method to specify the field in the record that defines whether the task should be treated as a milestone.

The fields specified by the SetPlannedStartDate and SetActualStartDate methods are used for determining the dates for the planned and actual milestones. The other date fields in the rowset are ignored

Parameters

Parameter	Description
RecordName.FieldName	Specify the field and its associated record that defines whether the task should be treated as a milestone. The field must be of type Character, and be one character long. The possible values for this field are Y, used to specify that this task is a milestone, or N, if not.

See Also

Chapter 11, "Chart Class," SetActualStartDate, page 380 , Chapter 11, "Chart Class," SetTaskMilestone, page 391 and Chapter 11, "Chart Class," SetPlannedStartDate, page 382.

SetTaskName

Syntax

SetTaskName(*RecordName.FieldName*)

Description

Use the SetTaskName method to specify the field in the record that defines the task name. If you don't specify a task name, the task ID is used in the table section of the Gantt chart.

Although this method is not required, PeopleSoft recommends using it to display meaningful information in the table section of the Gantt chart.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field name and its associated record that defines the task name. The field must be of type character.

See Also

Chapter 11, "Chart Class," SetTaskID, page 390

SetTaskProgress

Syntax

SetTaskProgress(*RecordName.FieldName*)

Description

Use the SetTaskProgress method to specify the field in the record that defines the length of the progress bar for this task as shown in the chart area of the Gantt chart.

 Copyright © 1988-2007, Oracle. All rights reserved.

The value in the field must be between 0 and 100, inclusive. Zero indicates no progress bar is shown, 100 indicates that the progress bar should cover the entire length of the task bar. Values greater than 100 are automatically converted to 100.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the length of the progress bar for this task. This field must be of type number. The value in the field must be between 0 and 100, inclusive. You must not specify a value greater than 100.

See Also

Chapter 11, "Chart Class," SetTaskProgressBarColor, page 393

SetTaskProgressBarColor

Syntax

SetTaskProgressBarColor(*RecordName.FieldName*)

Description

Use the SetTaskProgressBarColor method to specify the field in the record that defines the color for the progress bar.

In order for the progress bar to be visible, its color must contrast with the task bar color. There are no borders drawn around the progress bar.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field, and its associated record, that defines the task progress bar color. This field must be of type numeric, which means that you can only use the numeric value for the chart colors, not the constant. See Chapter 11, "Chart Class," Chart Colors, page 360.

See Also

Chapter 11, "Chart Class," SetActualTaskBarColor, page 381 and Chapter 11, "Chart Class," SetTaskProgress, page 392.

SetWBSNumbering

Syntax

SetWBSNumbering(*RecordName.FieldName*)

Description

Use the SetWBSNumbering method to specify the field in the record that defines the work breakdown structure (WBS) numbering to be displayed for each task row in the table section of the Gantt chart.

Copyright © 1988-2007, Oracle. All rights reserved.

A WBS is very similar in structure and layout to a document outline. Each item at a specific level of a WBS is numbered consecutively (that is, 10, 20, 30, 40, 50). Each item at the next level is numbered within the number of its parent item (that is, 10.1, 10.2, 10.3, 10.4). For example:

```
1.
   1.1
      1.1.1
      1.1.2
      1.1.3
   1.2
      1.2.1
      1.2.2
2.
. . .
```

If no field name is provided, WBS numbering is not displayed along with the tasks.

Regardless if WBS numbering is provided, child tasks are indented to the right of their parent task.

Parameters

Parameter	Description
RecordName.FieldName	Specify the field name and its associated record that defines the WBS numbering to be displayed. The field must be of type character.

Gantt Chart Properties

This section discusses Gantt chart properties. The properties are described in alphabetical order.

AxisEndDateTime

Description

Use this property to specify the date and time where the chart axis should end. The value must be of type Date or DateTime.

This property is read-write.

AxisStartDateTime

Description

Use this property to specify the date and time where the chart axis should begin. The value must be of type Date or DateTime.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

DataEndDateTime

Description

Use this property to specify where the chart should stop displaying data. The value must be of type DateTime.

This property is read-write.

DataStartDateTime

Description

Use this property to specify where the chart should start displaying data. The value must be of type DateTime.

This property is read-write.

PixelsPerRow

Description

This property returns an integer value of the row height (in pixels) for each row in the table section. Use this property to properly size the height of the table section of your Gantt chart so that all rows in the table are displayed. You can do so by multiplying the value for the PixelsPerRow property by the number of visible rows in the rowset.

The system calculates the value of the PixelsPerRow property at runtime after the Gantt image is rendered. To do so, this property must be read after the PeopleCode Activate event. This can be done by reading in the PixelPerRow value during a FieldChange event on a refresh button on the page.

This property is read-only.

ShowTaskLabels

Description

Use this property to specify whether or not task labels are displayed alongside the corresponding task bar in the chart area. This property takes a boolean value; true to display the labels, false to not display the labels. The default is to display the labels.

This property is read-write.

TaskDependencyLineType

Description

Use this property to specify an integer value that specifies the line type of the line connecting dependent tasks. This value affects all the dependency lines for the entire Gantt chart, not just for a specific set of tasks.

This property is read-write.

Numeric Value	Constant Value	Description
0	%ChartLine_Solid	Solid line
1	%ChartLine_Dashed	Dashed line
2	%ChartLine_Dotted	Dotted line

Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
3	%ChartLine_Mixed	Mixture of dashes and dots

TaskMilestoneGlyph

Description

Use this property of type integer to specify which field in the record defines the glyph to represent a task milestone (for both planned and actual milestones). This glyph is used for the entire chart, not just for a specific task.

By default a diamond shaped glyph will be used.

Milestones associated with planned dates use colors defined by the SetPlannedTaskBarColor method. Milestones associated with actual dates use colors defined by the SetActualTaskBarColor method.

This property is read-write.

See Also

Chapter 11, “Chart Class,” Using Gantt Glyphs, page 342

TaskTitle

Description

Use this property to specify the column title to be displayed in column one (that is, the leftmost column) in the table section of the Gantt chart. Use the SetTaskAppDataTitles method to specify the column titles for the application specific data fields (that is, column two and beyond).

This property is read-write.

See Also

Chapter 11, “Chart Class,” SetTaskAppDataTitles, page 384

XAxisPosition

Description

Use this property to specify the position of the X axis relative to the Y axis.

Note. This property is not supported in Gantt charts.

You can use either a constant or a numeric value for this property. Valid values are:

Numeric Value	Constant Value	Description
1	%ChartAxis_ZeroPosition	Positions the axis in line with the 0 on the opposite axis. Default position for the axis.
2	%ChartAxis_LowPosition	Positions the axis in line with the lowest value on the opposite axis.

 Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
3	%ChartAxis_HighPosition	Positions the axis in line with the highest value on the opposite axis.

This property is read-write.

YAxisPosition

Description

Use this property to specify the position of the Y axis relative to the X axis.

Note. This property is not supported in Gantt charts.

You can use either a constant or a numeric value for this property. Valid values are:

Numeric Value	Constant Value	Description
1	%ChartAxis_ZeroPosition	Positions the axis in line with the 0 on the opposite axis. Default position for the axis.
2	%ChartAxis_LowPosition	Positions the axis in line with the lowest value on the opposite axis.
3	%ChartAxis_HighPosition	Positions the axis in line with the highest value on the opposite axis.

This property is read-write.

Gantt Chart DateTimeAxis Specific Methods

This section discusses the DateTimeAxis specific methods that can be used with Gantt charts. The methods are listed in hierarchical order.

See Also

Chapter 11, “Chart Class,” Specifying DateTime Axis Formats, page 341

SetSecondFormat

Syntax

SetSecondFormat(*Format*)

Description

Use the SetSecondFormat method to specify how seconds display on a DateTime axis.

Locale-specific time separators are used to format the time.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Format	Specify the format for how the seconds should be displayed:

Value	Description
%Chart_SecondFormat	Displays the seconds using a one or two digit number.
%Chart_SecondFormat_2Digit	Displays seconds using a two digit number.

See Also
Chapter 11, "Chart Class," SetHourFormat, page 398 and Chapter 11, "Chart Class," SetMinuteFormat, page 398.

SetMinuteFormat

Syntax
SetMinuteFormat(*Format*)

Description

Use the SetMinuteFormat method to specify how minutes are displayed on the DateTime axis.

Locale-specific time separators are used to format the time.

Parameters

Parameter	Description
Format	Specify how the minutes should be displayed:

Value	Description
%Chart_MinuteFormat	Displays the minutes using a one or two digit number.
%Chart_MinuteFormat_2Digit	Displays the minutes using a two digit number.

See Also
Chapter 11, "Chart Class," SetHourFormat, page 398 and Chapter 11, "Chart Class," SetSecondFormat, page 397.

SetHourFormat

Syntax
SetHourFormat(*Format*)

Description

Use the SetHourFormat method to specify how the hour displays on a DateTime axis.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Format	Specify the format the hours should be displayed in. The values are:

Value	Description
%Chart_HourFormat_24	Display the hour using one or two digits in a 24 hour clock, that is 0–24.
%Chart_HourFormat_24_ 2Digit	Display the hour using two digits for a 24 hour clock, that is 00–24.
%Chart_HourFormat_12	Display the hour using one or two digits using a 12 hour clock. A.M. or P.M. is displayed next to the value.
%Chart_HourFormat_12_ 2Digit	Display the hour using one or two digits using a 12 hour clock. A.M. or P.M. is displayed next to the value.

See Also

Chapter 11, "Chart Class," SetDayFormat, page 399 , Chapter 11, "Chart Class," SetMinuteFormat, page 398 and Chapter 11, "Chart Class," SetSecondFormat, page 397.

SetDayFormat

Syntax

SetDayFormat(*Format*)

Description

Use the SetDayFormat method to specify the format of the day on a DateTime axis.

Parameters

Parameter	Description
Format	Specify the format the day should display in. The values are:

Value	Description
%Chart_DayFormat	Display the day in a one or two digit format.
%Chart_DayFormat_2Digit	Display the day in a two digit format.
%Chart_DayFormat_Name	Display the day using the full name, such as Monday, Tuesday, and so on.
%Chart_DayFormat_DOY	Display the day of the year in a numeric format, that is 1 — 366.
%Chart_DayFormat_DOY_ 2Digit	Display the day of the year in a two digit numeric format, that is 01 — 366.

See Also

Chapter 11, "Chart Class," SetHourFormat, page 398 , Chapter 11, "Chart Class," SetMinuteFormat, page 398 , Chapter 11, "Chart Class," SetMonthFormat, page 400 and Chapter 11, "Chart Class," SetSecondFormat, page 397.

Copyright © 1988-2007, Oracle. All rights reserved.

SetMonthFormat

Syntax

SetMonthFormat(*Format*)

Description

Use the SetMonthFormat method to specify the format of months on a DateTime axis.

Parameters

Parameter	Description
Format	Specify the format of how the month should be displayed. Values are:

Value	Description
%Chart_MonthFormat	Display the month using a numeric one or two digit number.
%Chart_MonthFormat_2Digit	Display the month using a numeric two digit number.
%Chart_MonthFormat_ FullName	Display the month using the full name.

See Also

Chapter 11, “Chart Class,” SetDayFormat, page 399 and Chapter 11, “Chart Class,” SetYearFormat, page 400.

SetYearFormat

Syntax

SetYearFormat(*Format*)

Description

Use the SetYearFormat method to specify the format of years on the DateTime axis.

Parameters

Parameter	Description
Format	Specify the format of how the year should be displayed. Values are:

Value	Description
%Chart_YearFormat	Display the year using a numeric one or two digit number.
%Chart_YearFormat_2Digit	Display the year using a numeric two digit number.
%Chart_YearFormat_4Digit	Display the year using a numeric four digit number.

See Also

Chapter 11, “Chart Class,” SetDayFormat, page 399 and Chapter 11, “Chart Class,” SetMonthFormat, page 400.

 Copyright © 1988-2007, Oracle. All rights reserved.

Chart Examples

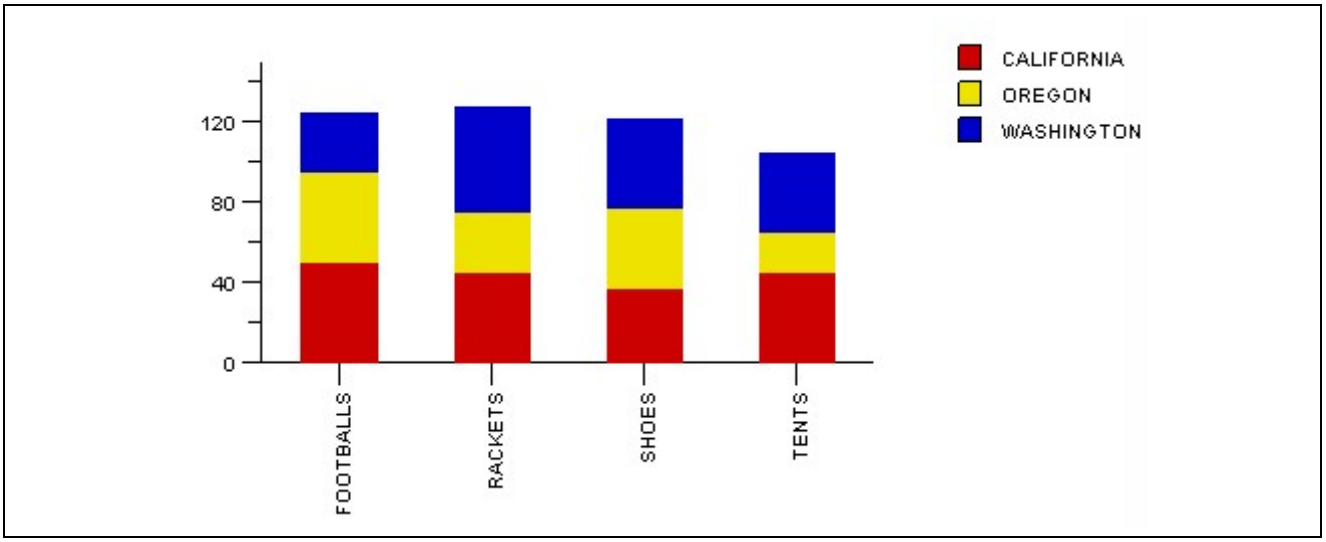
The following are some example charts, with the PeopleCode used to create them.

Creating a Chart

In the following example, the chart is created from a record with three fields, relating to sales, month, and product. The series in the chart are based on the product. The following is the complete code sample: the steps explain each line.

```
Global Chart &oChart;

&oChart = GetChart(QE_CHART_DUMREC.QE_CHART_FIELD);

&oChart.SetData(Record.QE_CHART_RECORD);
&oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
&oChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);

&oChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);

&oChart.Type = %ChartType_2DStackedBar;

&oChart.HasLegend = True;
&oChart.LegendPosition = %ChartLegend_Right;

&oChart.XAxisLabelOrient = %ChartText_Vertical;
```

The following is the chart produced by the previous code:



Test Chart - stacked bar

To create a chart:

1. Open PeopleSoft Application Designer.
2. Open the page where the chart is to be inserted.

Copyright © 1988-2007, Oracle. All rights reserved.

3. Insert the chart control by either:
   - Clicking on the chart icon in the toolbar, or
   - Selecting Insert, Chart
4. Draw the chart control on the page.
5. Associate the chart control with a record field

   Every chart control must be associated with a record field. This is just the field for the chart control. It is *not* the field used for drilling down for data in the chart.

   Bring up the chart control properties by either
   - Double-clicking on the chart control, or
   - Right-clicking on the chart control and selecting Page Field Properties.
6. Select the record name and field for the chart.

   On the Record tab of the chart control properties, select the record and field for the chart.

   To make the control a page anchor, select the Enable as Page Anchor on the General tab.
7. Write your PeopleCode.

   In some event on the page, such as Activate, put the PeopleCode you need for populating the chart.
8. Get the chart.

   The first thing you must do is get the chart. You must use the record and field name that you associated with the chart control on the page.

```
&oChart = GetChart(QE_CHART_DUMREC.QE_CHART_FIELD);
```

9. Set the data records.

   Set the data record. The SetData function associates the record data with the chart. Then use the SetDataYAxis and SetDataXAxis functions to associate specific fields in the record with the Y axis data and the X axis data.

```
&oChart.SetData(Record.QE_CHART_RECORD);
&oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
&oChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);
```

   This is all the code needed to create a chart. You don't need to set the chart type: the default is a 2D bar chart. You don't need to set the series unless you want to group your data. Everything else is also optional.
10. (Optional) Set the data series.

    In this example, we want to set the region as the series. This means that the data will be grouped according to the region.

```
&oChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);
```

11. (Optional) Set the chart type.

    Because we want a stacked bar for the chart, we must set the Type property of the chart. This means that each product (footballs, rackets, and so on) will have a single bar, and the data series (California, Oregon, and so on) will be what is 'stacked'.

```
&oChart.Type = %ChartType_2DStackedBar;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

12. (Optional) Set legend and label attributes.

    For this example, we want a legend, and want it to appear in the right side. In addition, because the text of the series labels is so large, they must be vertical to display all of them.

```
&oChart.HasLegend = True;
&oChart.LegendPosition = %ChartLegend_Right;

&oChart.XAxisLabelOrient = %ChartText_Vertical;
```

Creating a Gantt Chart

The following is the complete code sample: the steps explain each line.

```
&oGantt = GetGanttChart(QE_CHART_DUMREC.QE_CHART_FIELD);

/*/ /////////
// Required
// ////////*/

/*/ Assign record which holds the task related data */
&oGantt.SetTaskData(Record.QE_GANTT_TASK);
&oGantt.SetTaskID(QE_GANTT_TASK.QE_ID);
&oGantt.SetPlannedStartDate(QE_GANTT_TASK.PLANNED_START_DATE);
&oGantt.SetPlannedEndDate(QE_GANTT_TASK.PLANNED_END_DATE);


/*// /////////////////
// General options //
// /////////////////*/

&oGantt.SetTableXScrollbar(0.20);
&oGantt.SetChartArea(QE_GANTT_PROJ.QE_CHART_PERCENT);
&oGantt.width = QE_GANTT_PROJ.QE_WIDTH;
&oGantt.height = QE_GANTT_PROJ.QE_HEIGHT;
&oGantt.DataStartRow = QE_GANTT_PROJ.QE_START_ROW;
&oGantt.DataWidth = QE_GANTT_PROJ.QE_DATA_SIZE;
&oGantt.IsDrillable = QE_GANTT_PROJ.QE_DRILLABLE;
&oGantt.ShowTaskLabels = QE_GANTT_PROJ.QE_SHOWTASKLABELS;

&oGantt.GridLineType = QE_GANTT_PROJ.QE_GRIDLINE_TYPE;
If QE_GANTT_PROJ.QE_GRIDLINE_ENABLE = "Y" Then
   &oGantt.GridLines = %ChartGrid_Vertical;
Else
   &oGantt.GridLines = %ChartGrid_None;
End-If;



/*// ////////////////////////////////////
// Optional fields specific to task data //
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
// ////////////////////////////////////// */

&oGantt.SetTaskName(QE_GANTT_TASK.QE_NAME);
&oGantt.SetTaskWBSNumbering(QE_GANTT_TASK.QE_WBS_ID);
&oGantt.SetTaskLevel(QE_GANTT_TASK.QE_LEVEL);
&Gantt.SetTaskLabel(QE_GANTT_TASK.QE_LABEL);
&Gantt.SetTaskProgress(QE_GANTT_TASK.QE_PROGRESS);
&Gantt.SetTaskHints(QE_GANTT_TASK.QE_HINTS);
&Gantt.SetTaskBarURL(QE_GANTT_TASK.QE_URL);
&oGantt.SetTaskMilestone(QE_GANTT_TASK. QE_MILESTONE);
&oGantt.TaskMilestoneGlyph = QE_GANTT_PROJ.QE_MILESTONE_GLYPH;
&oGantt.SetTaskExpanded(QE_GANTT_TASK.QE_EXPANDED);

&Gantt.SetPlannedTaskBarColor(QE_GANTT_TASK.QE_PLANNED_TASK_COLOR);
&Gantt.SetActualTaskBarColor(QE_GANTT_TASK.QE_ACTUAL_TASK_COLOR);
&Gantt.SetTaskProgressColor(QE_GANTT_TASK.QE_PROGRESS_COLOR);


/*// ////////////////////////
// Specific to Table Area //
// ////////////////////////*/

&oGantt.SetTaskAppData(QE_GANTT_TASK.QE_PLANNED_START,
QE_GANTT_TASK.QE_PLANNED_END,
QE_GANTT_TASK.QE_TASK_PROGRESS,
QE_GANTT_TASK.QE_TASK_NAME,
QE_GANTT_TASK.QE_EXPANDED);

 &oGantt.TaskTitle = "Tasks";
 &appDataArray = CreateArray("Start", "End", "Progress", "Name", "Expanded");
 &oGantt.SetTaskAppDataTitles(&appDataArray);



/*     ////////////////////////////////////////////
     // Optional fields for task dependency data //
     //////////////////////////////////////////// */

/*// Assign record which holds the task dependency related data */
&oGantt.SetTaskDepenencyData(Record.QE_GANTT_TASKD);
&oGantt.SetTaskDependencyParentID(QE_GANTT_TASKD.QE_PARENT_TASK_ID);
&oGantt.SetTaskDependencyChildID(QE_GANTT_TASKD.QE_TASK_ID);
&oGantt.TaskDependencyLineType = QE_GANTT_PROJ.QE_LINETYPE_CHAR;
&oGantt.SetTaskDependencyURL(QE_GANTT_TASKD.URL);


/*// ////////////////////
// Enable baselines //
// //////////////////// */
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
&oGantt.SetActualStartDate(QE_GANTT_TASK.QE_ACTUAL_START);
&oGantt.SetActualEndDate(QE_GANTT_TASK.QE_ACTUAL_END);



/*////////////////////////////
// DateTime Axis specific //
////////////////////////////*/
&oGantt.AxisStartDateTime = DateTimeValue("07/15/2002 8:00:00");
&oGantt.AxisEndDateTime = DateTimeValue("09/15/2002 8:00:00");

&oGantt.SetMonthFormat(%Chart_MonthFormat_FullName);
&Gantt.SetDayFormat(%Chart_DayFormat_2Digit);
```

To create a Gantt chart:

1. Get a Gantt chart.

   Get the chart using the GetGanntChart function. The record and field name specified are associated with a chart definition in PeopleSoft Application Designer.

   ```
   Local Gantt &oGantt;

   &oGantt = GetGanttChart(QE_CHART_DUMREC.QE_CHART_FIELD);
   ```

2. Specify values for the required methods.

   There are several methods that are required for the Gantt chart, such as setting the task data, the start and end dates, and so on.

   ```
   &oGantt.SetTaskData(Record.QE_GANTT_TASK);
   &oGantt.SetTaskID(QE_GANTT_TASK.ID);
   &oGantt.SetPlannedStartDate(QE_GANTT_TASK.START_DATE);
   &oGantt.SetPlannedEndDate(QE_GANTT_TASK.END_DATE);
   ```

3. Specify values for the additional methods controlling the display of the data.

   After you've set the required methods, you can set additional methods to control the display of the chart. The following code specifies the amount that the scroll bars scroll when clicked, the percentage of the entire chart that is taken up by the chart area, the width and height of the chart, on what row the chart should begin displaying data, the width of the data, if the user can click on the chart area of the Gantt chart and either execute a function or a URL, and whether labels should display with the tasks in the task section.

   ```
   &oGantt.SetTableXScrollbar(0.20);
   &oGantt.SetChartArea(QE_GANTT_PROJ.QE_CHART_PERCENT);
   &oGantt.width = QE_GANTT_PROJ.QE_WIDTH;
   &oGantt.height = QE_GANTT_PROJ.QE_HEIGHT;
   &oGantt.DataStartRow = QE_GANTT_PROJ.QE_START_ROW;
   &oGantt.DataWidth = QE_GANTT_PROJ.QE_DATA_SIZE;
   &oGantt.IsDrillable = QE_GANTT_PROJ.QE_DRILLABLE;
   &oGantt.ShowTaskLabels = QE_GANTT_PROJ.QE_SHOWTASKLABELS;
   ```

Copyright © 1988-2007, Oracle. All rights reserved.

4. Specify how the grid lines are displayed.

   Depending on the type of data that you are using, you can specify the type of grid lines that should display in the chart section of the Gantt chart.

```
&oGantt.GridLineType = QE_GANTT_PROJ.QE_GRIDLINE_TYPE;
If QE_GANTT_PROJ.QE_GRIDLINE_ENABLE = "Y" Then
   &oGantt.GridLines = %ChartGrid_Vertical;
Else
   &oGantt.GridLines = %ChartGrid_None;
End-If;
```

5. Additionally specify how the data is to be displayed.

   The following methods control how the different parts of the task data displays, such as the label, what hints display when the mouse is passed over the task data, and so on.

```
&oGantt.SetTaskName(QE_GANTT_TASK.QE_NAME);
&oGantt.SetTaskWBSNumbering(QE_GANTT_TASK.QE_WBS_ID);
&oGantt.SetTaskLevel(QE_GANTT_TASK.QE_LEVEL);
&Gantt.SetTaskLabel(QE_GANTT_TASK.QE_LABEL);
&Gantt.SetTaskProgress(QE_GANTT_TASK.QE_PROGRESS);
&Gantt.SetTaskHints(QE_GANTT_TASK.QE_HINTS);
&Gantt.SetTaskBarURL(QE_GANTT_TASK.QE_URL);
&oGantt.SetTaskMilestone(QE_GANTT_TASK. QE_MILESTONE);
&oGantt.TaskMilestoneGlyph = QE_GANTT_PROJ.QE_MILESTONE_GLYPH;
&oGantt.SetTaskExpanded(QE_GANTT_TASK.QE_EXPANDED);

&Gantt.SetPlannedTaskBarColor(QE_GANTT_TASK.QE_PLANNED_TASK_COLOR);
&Gantt.SetActualTaskBarColor(QE_GANTT_TASK.QE_ACTUAL_TASK_COLOR);
&Gantt.SetTaskProgressColor(QE_GANTT_TASK.QE_PROGRESS_COLOR);
```

6. Specify the additional fields to be displayed in table section.

   You can specify application data to also be displayed with the task data.

```
&oGantt.SetTaskAppData(QE_GANTT_TASK.QE_PLANNED_START,
QE_GANTT_TASK.QE_PLANNED_END,
QE_GANTT_TASK.QE_TASK_PROGRESS,
QE_GANTT_TASK.QE_TASK_NAME,
QE_GANTT_TASK.QE_EXPANDED);
```

7. Specify titles for the task bars.

   You can specify different titles for the different task bars.

```
&oGantt.TaskTitle = "Tasks";
&appDataArray = CreateArray("Start", "End", "Progress", "Name", "Expanded");
&oGantt.SetTaskAppDataTitles(&appDataArray);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

8. Specify the fields for task dependency data.

   You do not need to specify dependency data. However, if you do, you must also specify the parent ID and the child ID (where the dependency data starts and ends.) You can also specify optional things, like the type of line used between the dependencies, as well as a URL that the browser will open to if the user clicks a dependency line.

```
&oGantt.SetTaskDepenencyData(Record.QE_GANTT_TASKD);
&oGantt.SetTaskDependencyParentID(QE_GANTT_TASKD.QE_PARENT_TASK_ID);
&oGantt.SetTaskDependencyChildID(QE_GANTT_TASKD.QE_TASK_ID);
&oGantt.TaskDependencyLineType = QE_GANTT_PROJ.QE_LINETYPE_CHAR;
&oGantt.SetTaskDependencyURL(QE_GANTT_TASKD.URL);
```

9. Specify the actual dates for the project.

   If you know the actual start and end date, in addition to the planned date, you can add those to your chart as well.

```
&oGantt.SetActualStartDate(QE_GANTT_TASK.QE_ACTUAL_START);
&oGantt.SetActualEndDate(QE_GANTT_TASK.QE_ACTUAL_END);
```

10. Specify the DateTime axis values.

    You can specify values for where the DateTime axis starts and ends.

```
&oGantt.AxisStartDateTime = DateTimeValue("07/15/2002 8:00:00");
&oGantt.AxisEndDateTime = DateTimeValue("09/15/2002 8:00:00");
```

11. Specify the format of the DateTime axis values.

    You can specify the format of the DateTime elements, such as how the days of the week display, whether as a number or using the full name, and so on.

```
&oGantt.SetMonthFormat(%Chart_MonthFormat_FullName);
&Gantt.SetDayFormat(%Chart_DayFormat_2Digit);
```

Creating a Chart Using an iScript

You can also create a chart using an iScript. The following example creates a chart, populates it, then sends the chart URL to a response object. The following is the complete code sample: the steps explain each line.

```
Function IScript_GetChartURL()

   Local Chart &oChart;
   Local Rowset &oRowset;
   Local string &sMap;
   Local string &sURL;

   &oChart = CreateObject("Chart");

   &oRowset = CreateRowset(Record.QE_CHART_RECORD);
   &oRowset.Fill("where QE_CHART_REGION= :1", "MIDWEST");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
    &oChart.SetData(&oRowset);

    &oChart.Width = 400;
    &oChart.Height = 300;

    &oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
    &oChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);
    &oChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);

    &oChart.HasLegend = True;
    &oChart.LegendPosition = %ChartLegend_Right;

    &sURL = %Response.GetChartURL(&oChart);
    &sMap = &oChart.ImageMap;

    %Response.Write("<HTML><IMG SRC=");
    %Response.Write(&sURL);
    %Response.Write("  USEMAP=#THEMAP></IMG><MAP NAME=THEMAP>");
    %Response.Write(&sMap);
    %Response.Write("</MAP></HTML>");

End-Function;
```

To create a chart using an iScript:

1. Create the chart object.

   For this example, there is no chart control on a page to be referenced. So to create the chart object, you must use the CreateObject function. The string passed in to the function must be the word 'chart'.

   ```
   &oChart = CreateObject("Chart");
   ```

2. Create a rowset for the chart data and set the chart data.

   The CreateRowset function creates a rowset data structure. Using the Fill method populates the empty rowset with data. The SetData method associates the rowset data with the chart.

   ```
   &oRowset = CreateRowset(Record.QE_CHART_RECORD);
    &oRowset.Fill("where QE_CHART_REGION= :1", "MIDWEST");
    &oChart.SetData(&oRowset);
   ```

3. Set the height and width for the chart.

   Because the chart isn't associated with a chart control on a page, you have to specify the size of the chart image to be generated using the Height and Width properties. The unit of measurement for both these properties is pixels.

   ```
   &oChart.Width = 400;
    &oChart.Height = 300;
   ```

4. Set the data axes, series, and legend for the chart.

   As with all charts, you must set the data axes. If necessary for your data, also set the data series and the legend.

   ```
   &oChart.SetDataYAxis(QE_CHART_RECORD.QE_CHART_SALES);
   ```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&oChart.SetDataXAxis(QE_CHART_RECORD.QE_CHART_PRODUCT);
&oChart.SetDataSeries(QE_CHART_RECORD.QE_CHART_REGION);

&oChart.HasLegend = True;
&oChart.LegendPosition = %ChartLegend_Right;
```

5. Generate the URL for the chart.

   Use the GetChartURL method of the Response class to generate the URL referencing the chart data.

   ```
   &sURL = %Response.GetChartURL(&oChart);
   ```

6. Generate the image map.

   Use the ImageMap property to generate the image map for the chart. The Response object uses this to draw the map.

   ```
   &sMap = &oChart.ImageMap;
   ```

7. Create the response chart.

   Use the Write Response class method to generate the chart. Note that first the data is set, using the URL generated with GetChartURL, then the image map is used.

   ```
   %Response.Write("<HTML><IMG SRC=");
    %Response.Write(&sURL);
    %Response.Write("  USEMAP=#THEMAP></IMG><MAP NAME=THEMAP>");
    %Response.Write(&sMap);
    %Response.Write("</MAP></HTML>");
   ```

Note. If one or more labels does not render in the chart increase the width of the chart, use shorter label text, or reduce the data set being charted.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 12

Component Interface Classes

This chapter provides an overview of Component Interface class and discusses the following topics:

- Life-cycle of a Component Interface.
- Declaring a Component Interface object.
- Scope of a Component Interface object.
- Implementing a Component Interface object.
- Component Interface methods and timeouts.
- Traversing a Component Interface and using data collections.
- Working with effective-dated data.
- Reusing existing code.
- Component Interface Reference.

Understanding Component Interface Class

Component Interfaces are the focal points for externalizing access to existing PeopleSoft components. They provide realtime synchronous access to the PeopleSoft business rules and data associated with a component outside the PeopleSoft online system. Component Interfaces can be viewed as "black boxes" that encapsulate PeopleSoft data and business processes, and hide the details of the structure and implementation of the underlying page and data.

Component Interfaces are one of the many APIs that PeopleSoft provides for enabling integration with other systems.

A Component Interface maps to one, and only one, PeopleSoft component. The *Component Interface object*, instantiated from a session object, is created at runtime to access the data specified by the Component Interface.

When you instantiate a Component Interface object:

- All the PeopleCode programs associated with the record fields, pages, component, and so on, and
- The runtime component processor still perform all of the work that they do in the online environment.

The exceptions are any GUI manipulation found in a PeopleCode program, and search dialog specific processing.

Component Interfaces are programmable through a C interface, an OLE/COM or C/C++ interface, and through PeopleCode. Application Engine programs, message notification programs, or any other PeopleCode programs are able to use Component Interfaces.

Copyright © 1988-2007, Oracle. All rights reserved.

Like a component, you create the structure of a Component Interface in PeopleSoft Application Designer, then at runtime, you populate the structure with data. This document is concerned with the runtime portion of a Component Interface.

When you populate a Component Interface with data, the first thing you fill out are its keys, as you would in a component. These can be keys for getting an existing instance of the data or for creating a new instance of the data.

In addition to keys, a Component Interface is composed of *properties* and *methods*.

- Component Interface *properties* provide access to the data in a component buffer.
- Component Interface *methods* are functions that can be called to perform operations on a Component Interface.

There are two types of both methods and properties: standard and user-defined. Standard properties and methods are provided automatically when you create a Component Interface. They perform operations common to all Component Interfaces, such as indicating what mode to operate the Component Interface, saving, or creating a Component Interface. User-defined properties are the specific record fields that an application developer has chosen to expose to an external system with the Component Interface. User-defined methods are PeopleCode programs that an application developer can write to perform operations on a Component Interface. Each is specific to that Component Interface.

You can instantiate a Component Interface object only from a session object. Through the session object you can control access to the Component Interface, check for errors, control the runtime environment, and so on.

See Also

Chapter 12, "Component Interface Classes," Reusing Existing Code, page 422

Chapter 36, "Session Class," page 1511

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Component Interfaces

Life Cycle of a Component Interface

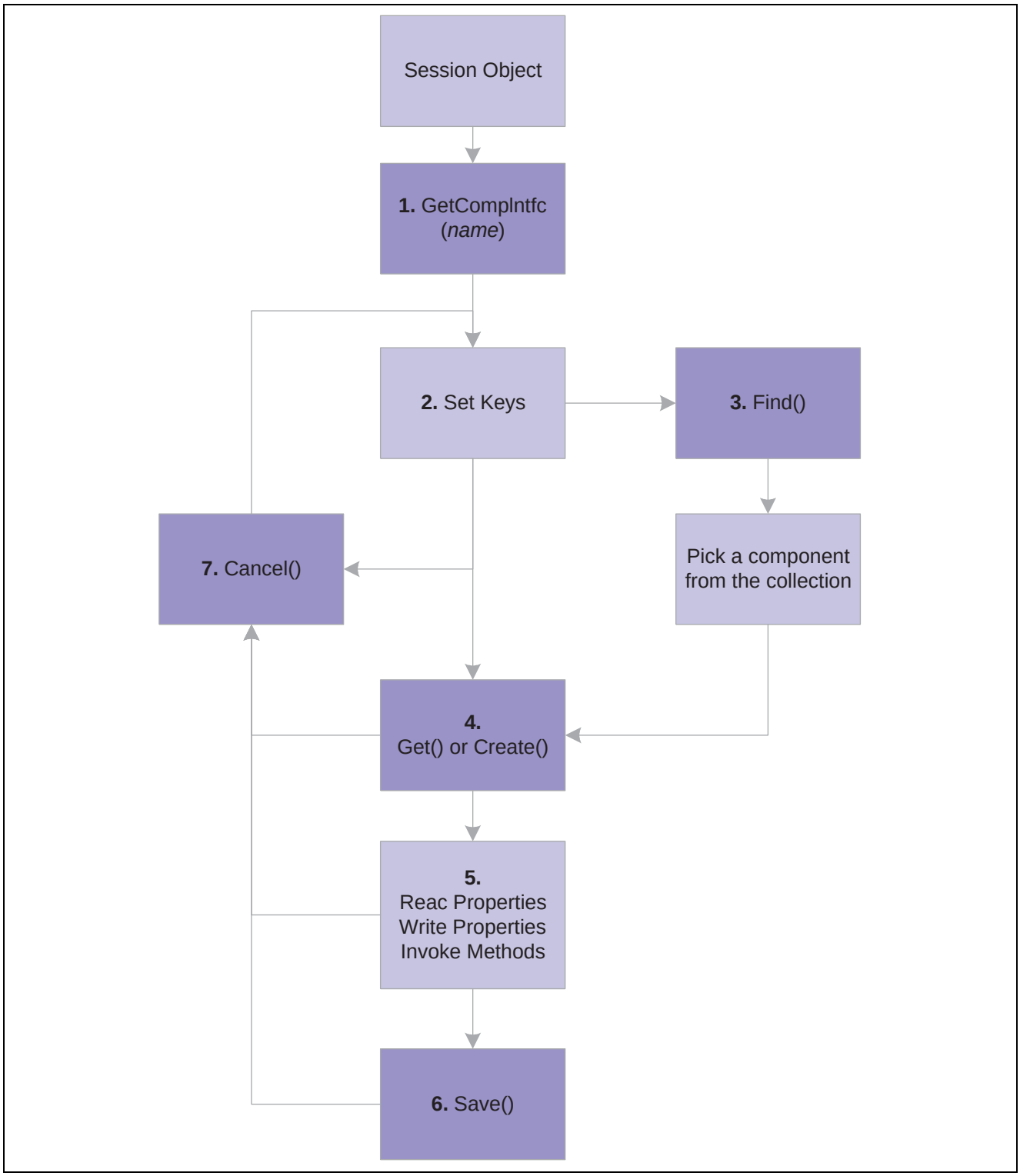
At runtime, there are certain things you want to do with a Component Interface, such as getting an instance of it, populating it with data, and so on. The following is an overview of this process. These steps are expanded in other sections.

1. Execute the GetCompIntfc method on the PeopleSoft session object to get a Component Interface.
2. Set the key values for the Component Interface object. If the keys you specify don't uniquely describe a component (partial keys), proceed to step 3. If the keys uniquely describe a component, skip to step 4.
3. Do *one* of the following:
   - Execute the Find standard method on the Component Interface. This returns a collection of Component Interfaces with their key values filled out. The user can then select the unique Component Interface they want.
   - Execute either the Get or Create standard method to instantiate the Component Interface (populate it with data.)
4. Get property values, set property values, or execute user-defined methods. Setting a property value will fire the standard PeopleSoft business rules for the field associated with the property (any PeopleCode program associated with FieldChange, RowInsert, and so on.)

 Copyright © 1988-2007, Oracle. All rights reserved.

5. Execute the Save standard method to fire the standard PeopleSoft save business rules (any PeopleCode programs associated with SavePreChange, WorkFlow, and so on.) and commit any changes to the database.
6. At any point, the standard method Cancel can be executed to reset the Component Interface to its state in step 1.

Copyright © 1988-2007, Oracle. All rights reserved.



Component interface life cycle

Setting Component Interface Keys

The keys for a Component Interface are based on the key fields of the underlying component. There can be different types of keys for a Component Interface.

Copyright © 1988-2007, Oracle. All rights reserved.

- CREATEKEYS: A list of the primary key fields of the search record specified to be used in Add mode for the component. This list is automatically generated.
- GETKEYS: A list of the primary (required) key fields on the search record. This list is automatically generated.
- FINDKEYS: A list of primary and alternate key fields on the primary search record for the component.

If the Component Interface has CREATEKEYS, these are the keys you must set before you execute the Create() method to create a new instance of the data. If the Component Interface doesn't have CREATEKEYS, use the GETKEYS to specify a new instance of the data.

Use either the GETKEYS or FINDKEYS to specify an existing instance of the data.

To set key values, use the field names listed under GETKEYS, CREATEKEYS or FINDKEYS like properties on the Component Interface object. The following example sets the CREATEKEYS values to create a new instance of the data.

```
&MYCI = GetCompIntfc(CompIntfc.EXPRESS_ISSUE);
&MYCI.BUSINESS_UNIT = "H01B";
&MYCI.INTERNAL_FLG = "Y";
&MYCI.ORDER_NO = "NEXT&rsquo;;
&MYCI.Create();
```

Standard and User-Defined Component Interface Methods

Every Component Interface comes with a standard set of methods:

- Cancel()
- Create()
- Find()
- Get()
- Save()

Use these methods during runtime to affect the data of the Component Interface.

The application developer can, at design time, disable any of these methods for the Component Interface.

In addition, an application developer can write their own methods. These methods are written as Functions using Component Interface PeopleCode. For example, suppose you wanted to be able to copy an instance of Component Interface data. You might write your own Clone method.

Note. User-defined method names must *not* be named *GetPropertyName*. The C header for Component Interfaces creates functions with that name so you can access each property. If you create your own *GetPropertyName* functions, you receive errors at runtime. User-defined methods can take only simple types of arguments (such as number, character, and so on) because user-defined methods can be called from C/C++ or COM as well as from PeopleCode. PeopleCode can use more complex types (like rowset, array, record, and so on), but these types of arguments are unknown to C/C++ or COM.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 12, "Component Interface Classes," Component Interface Class Methods, page 426

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Component Interfaces

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Component Interfaces, "Developing the Component Interface," Creating User-Defined Methods

Accessing Component Interface Standard Properties

Every Component Interface comes with a standard set of properties. These properties can be divided as follows:

Properties that affect how the Component Interface is executed

The following properties affect how a component interface is executed:

- EditHistoryItems
- GetHistoryItems
- InteractiveMode

These properties must be set before the Component Interface is populated with data. That is, you must set these properties before you use the Get or Create methods.

EditHistoryItems and GetHistoryItems work together to determine how data is accessed:

- If EditHistoryItems is False (the default) and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in update/display All mode. This means all history rows are returned, however, you can edit rows only with a date set in the future.
- If EditHistoryItems is True and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in correction mode. This means all history rows are returned, and you can edit them.
- If GetHistoryItems is False, you access the data in the Component Interface in a similar manner as if you were accessing a component in update mode. The EditHistoryItems has no effect when GetHistoryItems is False.

  InteractiveMode causes the Component Interface to emulate an online component. For example, if you set a value for a field in a Component Interface and you have set InteractiveMode to True, then any FieldChange PeopleCode programs associated with that field fire as soon as you set that value.

Properties that return information about the structure of the Component Interface

The following properties return information about the structure of the Component Interface:

- CreateKeyInfoCollection
- FindKeyInfoCollection
- GetKeyInfoCollection
- PropertyInfoCollection

See Also

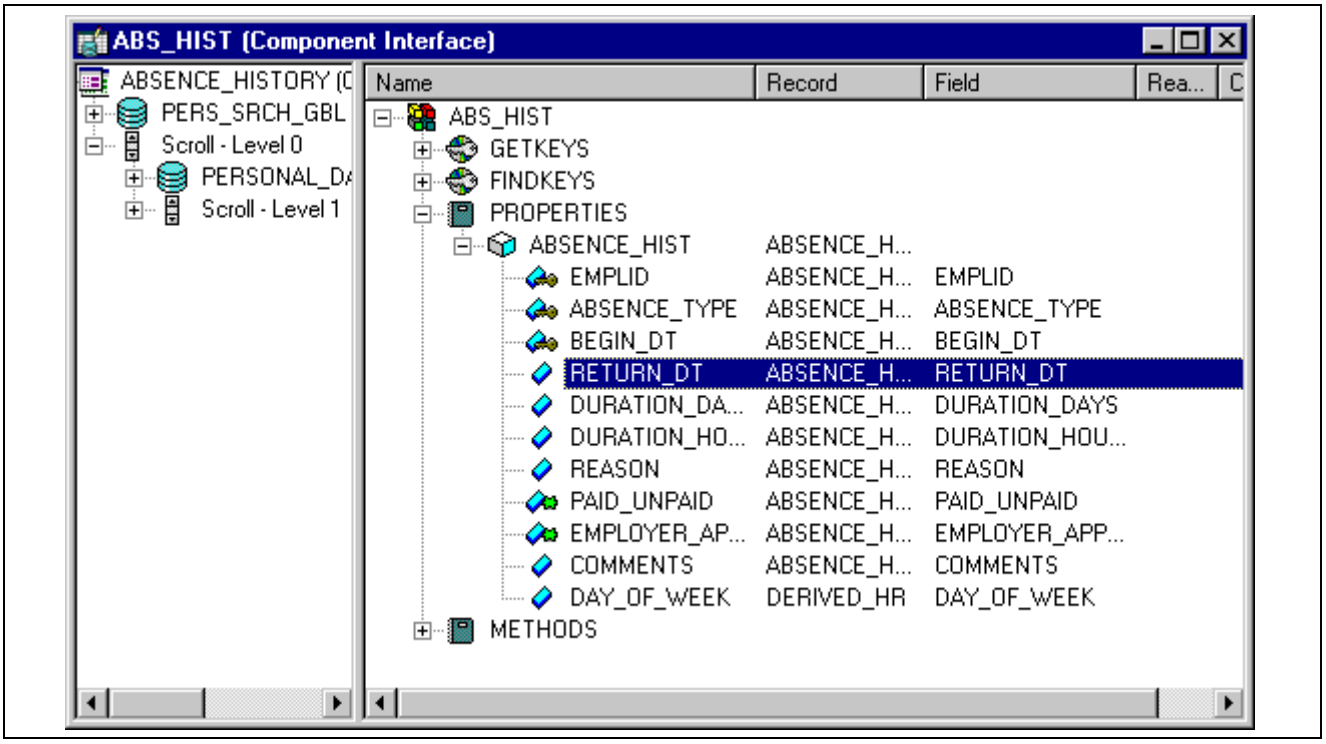
Chapter 12, "Component Interface Classes," Component Interface Class Properties, page 440

 Copyright © 1988-2007, Oracle. All rights reserved.

Accessing User-Defined Component Interface Properties

Every user-defined property in a Component Interface *definition* can be used like a property on the *object* instantiated from that Component Interface at runtime.

For example, the following Component Interface definition has RETURN_DT defined as one of its properties.



RETURN_DT Component Interface property highlighted

At runtime, you can use PeopleCode to assign a value to that property (field), or to access the value of that field.

```
&MYCI.RETURN_DT = "05/05/2000";

/*  OR  */

&DATE = &MYCI.RETURN_DT;
```

Data Type of a Component Interface Object

Component Interfaces are declared as type ApiObject. For example,

```
Local ApiObject &MYCI;
```

Note. Component Interface objects can be declared only as Local.

Copyright © 1988-2007, Oracle. All rights reserved.

Scope of a Component Interface Object

A Component Interface can be instantiated from PeopleCode, from a Visual Basic program, from Java, COM and C/C++.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

If you're instantiating a Component Interface from an online page, after you finish working with the component, you may want to refresh your page. The Refresh method, on a rowset object, reloads the rowset (scroll) using the current page keys. This causes the page to be redrawn. GetLevel0().Refresh() refreshes the entire page. If you want a particular scroll only to be redrawn, you can refresh just that part.

Considerations Using Component Interfaces and Related Languages

If you update a field using a Component Interface through an external program (such as Java, VB, and so on) the Related Language Table gets updated, even if they are using a base language of ENG.

Considerations Using Component Variables With Component Interfaces

A component variable is scoped locally to its component or its Component Interface. This means that you cannot use a component variable to share data between a component and a Component Interface. In order to share information between components, use global variables.

If your page (component) calls a Component Interface (using the existing session), that in turn initializes a component variable, that component variable is shared with the calling component. When the Component Interface is closed, the component variable is no longer in scope.

When the calling component is closed, any component scoped variables created by it or by the Component Interface go out of scope.

See Also

Chapter 33, "Rowset Class," Refresh, page 1438

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Component Interfaces

Implementing a Component Interface

After you create your Component Interface definition, you can use PeopleCode to access it. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the Component Interface definition from PeopleSoft Application Designer Project View into an open PeopleCode edit pane. PeopleSoft Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify.

You can also access your Component Interface using COM. You can automatically generate a Visual Basic or a C template, similar to the PeopleCode template, to get you started.

The following are the usual actions that you perform with a Component Interface:

- Create a new instance of data
- Get an existing instance of data

 Copyright © 1988-2007, Oracle. All rights reserved.

- Retrieve a list of instances of data

The following procedures cover each of these actions in more detail.

Another standard action is inserting or deleting a row of data (an item). This involves traversing a Component Interface (going from level zero to level one, from level one to level two, and so on) and accessing data collections.

You may want to work with effective-dated information. There are several properties you can set to allow you to do this.

In addition to these standard actions, you can also look at the structure of a Component Interface.

See Also

Chapter 12, "Component Interface Classes," Create a New Instance of Data Example, page 458

Chapter 12, "Component Interface Classes," Getting an Existing Instance of Data Example, page 461

Chapter 12, "Component Interface Classes," Retrieving a List of Instance of Data Example, page 463

Chapter 12, "Component Interface Classes," Traversing a Component Interface and Using Data Collections, page 419

Chapter 12, "Component Interface Classes," Working With Effective-Dated Data, page 421

Chapter 12, "Component Interface Classes," Accessing the Structure of a Component Interface, page 451

Component Interface Methods and Timeouts

The file pstools.properties controls when a Component Interface method times out. If you're having problems with methods timing out, you may want to change the values in this file. This file is located in the default directory for the application you're running. If this file isn't in this directory, copy it into the directory before you make your changes to it.

Traversing a Component Interface and Using Data Collections

The data in a Component Interface can be contained in a hierarchy: like the page it's built on, there may be data at level zero, level one, level two, and so on.

Each level of data in a Component Interface is known as a collection, such as:

```
level zero

  -- level one (Collection)

    -- Level 2 (Collection)
```

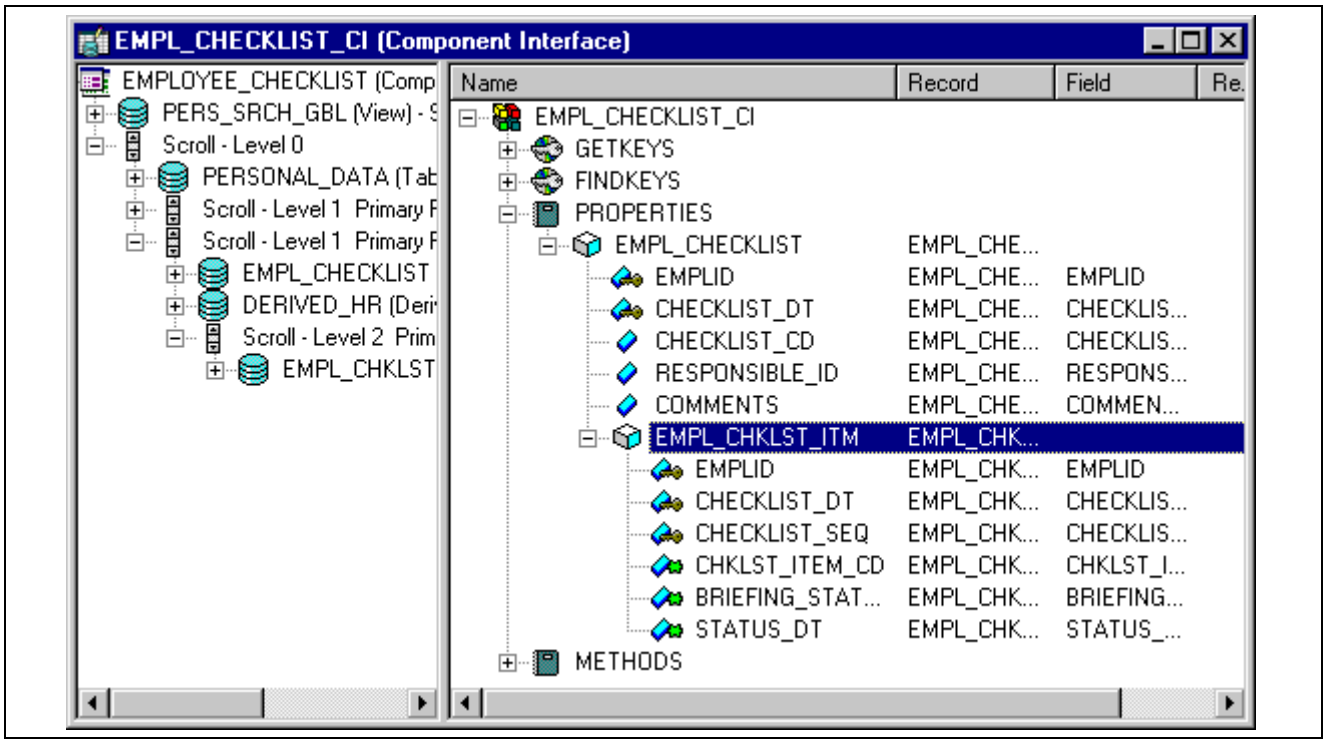
A *collection* is a set of similar things, like a group of already existing Component Interfaces. Most collections have the same standard properties and methods, with some additional ones specific to that collection. For example, all collections have the property *Count*, which indicates how many items are in that collection, but only a data collection has the method *ItemByKeys*.

Copyright © 1988-2007, Oracle. All rights reserved.

A *data collection* is the collection of data, available at runtime or during test mode, associated with a particular scroll (or record.) The data collection object returns information about every *row of data* (item) that is returned for that record at runtime.

To access the level two collection, in general, you could use the following:

```
&Level_1 = &CI.Level_1;
&Level_1_Item = &Level_1.Item(ItemNumber);

&Level_2 = &Level_1_Item.Level_2;
```



Sample Component Interface with collection highlighted

This example shows a Component Interface with a two-level hierarchy, that is, *two* data collections: EMPL_CHECKLIST and EMPL_CHKLST_ITM. To get a data collection for the first level, use the following code:

```
&Level1 = &MYCI.EMPL_CHECKLIST;
```

To access the secondary scroll (EMPL_CHKLST_ITM) you have to get the appropriate row (item) of the upper level scroll first:

```
&Level1 = &MYCI.EMPL_CHECKLIST;
&Item = &Level1.Item(1);
&Level2 = &Item.EMPL_CHKLST_ITM;
```

Note. Scrolls represent a hierarchical order. You must get the first level row that contains the secondary scroll *before* you can access the secondary scroll.

Every data collection has a set of methods and standard properties.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 12, “Component Interface Classes,” Inserting or Deleting a Row of Data Example, page 469

Chapter 12, “Component Interface Classes,” Data Collection Methods, page 443

Chapter 12, “Component Interface Classes,” Data Collection Properties, page 449

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer’s Guide, “Accessing the Data Buffer”

Working With Effective-Dated Data

When you work with effective-dated data, use some combination of the following methods:

- CurrentItem
- GetEffectiveItem

Both these methods return an item, however, there are some differences:

- The CurrentItem method takes no parameters, so it bases the item it returns on the *current system date*.
- The GetEffectiveItem method bases the item it returns on a *user-provided date*. They all perform the same logic, they just use a different date.

Note. To show all history for an effective-dated collection, you must set GetHistoryItems to True before you populate the Component Interface.

See Also

Chapter 12, “Component Interface Classes,” CurrentItem, page 444

Chapter 12, “Component Interface Classes,” GetEffectiveItem, page 446

Chapter 12, “Component Interface Classes,” GetHistoryItems, page 442

Chapter 12, “Component Interface Classes,” Inserting Effective-Dated Data Example, page 465

Chapter 12, “Component Interface Classes,” Inserting Effective-Dated Data Example Using Visual Basic, page 468

How a Developer Uses GetEffectiveItem

If a user is making an update to an effective-dated record, they don’t always want to insert a row at the end.

Suppose the database contained the following data:

EMPLID	EFFDT	SEQNO
8000	3/1/99	0
8000	7/1/99	0

Copyright © 1988-2007, Oracle. All rights reserved.

EMPLID	EFFDT	SEQNO
8000	9/1/99	0
8000	12/1/99	0

Now suppose the user wants to enter info with EFFDT of 11/1/99. If they were looking at a PeopleSoft component, they would visually scan to see where that date falls and then press ALT+7 and ENTER at the row that they want to insert after.

GetEffectiveItem enables you to pass in the correct effective date, instead of having to loop through every item in the collection doing a comparison, looking for the correct item.

Why Can't it Just Go at the End?

The InsertItem method simulates pressing ALT+7 and ENTER online to insert a row in a scroll. Part of the logic that occurs in the Component Processor is that if the scroll is effective-dated, the ALT+7 and ENTER carries the values forward from the previous row. This functionality is still there if you use InsertItem at the end of the collection, but the values may be incorrect.

Reusing Existing Code

One of the advantages of using a Component Interface is that it enables you to reuse existing PeopleCode and business logic. However, a Component Interface isn't exactly equivalent to a component, which means there are a few key areas in which you should expect differences in behavior between a Component Interface and the component it's based on. This section discusses the differences in:

- Search dialog processing
- Event and function behavior

Differences in Search Dialog Processing

When you run a Component Interface, the *SearchInit*, *SearchSave*, and *RowSelect* events don't fire. This means that any PeopleCode associated with these events won't run. The first event to run is *RowInit*.

Differences in PeopleCode Event and Function Behavior

PeopleCode events and functions that relate exclusively to GUI and online processing can't be used by Component Interfaces. These include:

- Menu PeopleCode and pop-up menus. The ItemSelected and PrePopup PeopleCode events are not supported. In addition, the DisableMenuItem, EnableMenuItem, and HideMenuItem functions aren't supported.
- Transfers between components, including modal transfers. The DoModal, EndModal, IsModal, Transfer, TransferPage, DoModalComponent, TransferNode, TransferPortal, and IsModalComponent functions cannot be used.
- Cursor position. SetControlValue cannot be used.
- WinMessage cannot be used.

 Copyright © 1988-2007, Oracle. All rights reserved.

- Save in the middle of a transaction. DoSave can't be used.

For the unsupported functions, you should put a condition around them, testing whether there's an existing Component Interface.

```
If %ComponentName Then
   /* process is being called from a Component Interface */
   /* do CI specific processing */
Else
   /* do regular processing */
   .  .  .
End-if;
```

Using %Menu Conditions

If you associate a Component Interface definition with a menu in PeopleSoft Application Designer, a PeopleCode program that conditionally checks for that menu runs when the Component Interface is executed. For example, suppose you associate a Component Interface with the MP_HR_MENU. The following PeopleCode program, if called from that Component Interface, executes:

```
If %Menu = MP_HR_MENU Then
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Component Interfaces, "Developing the Component Interface," Associating a Component Interface with a Menu

Session Class Methods

Component Interfaces don't have any built-in functions. They are instantiated from the Session class.

This section discusses the Session class methods used to instantiate Component Interfaces. The methods are discussed in alphabetical order.

FindCompIntfcs

Syntax

FindCompIntfcs([*ComponentName*])

Description

The FindCompIntfcs method, used with the session object, returns a reference to a Component Interface collection, filled with zero or more Component Interfaces. The *ComponentName* parameter takes a string value. You can use a partial value to limit the set of Component Interfaces returned. You can also specify a null value, that is, two quotation marks with no space between them, ("") to return the entire list of Component Interfaces available.

Using the FindCompIntfcs method is equivalent to using File, Open, and selecting Component Interface in PeopleSoft Application Designer.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

In the following example, a partial key was used to open all components starting with "APP".

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCICOLL = &MYSESSION.FindCompIntfcs("APP");
&MYCI = &MYCICOLL.First();
```

GetCompIntfc

Syntax

GetCompIntfc([**COMPINTFC.**]*ComponentName*)

Description

The GetCompIntfc method, used with the session object, returns a reference to a Component Interface. *ComponentName* when used by itself takes a string value. If you use COMPINTFC.*componentname* it isn't a string value: it's a constant that automatically is renamed in your code if the Component Interface definition is ever renamed. You must specify an existing Component Interface, otherwise you receive a runtime error.

Example

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.PERSONAL_DATA_BC);
```

The following example uses the '@' operator to dynamically call a Component Interface from PeopleCode.

```
&MYCI = &MYSESSION.GetCompIntfc(@("CompIntfc." | &CIname));
```

Component Interface Collection Methods

A Component Interface collection is a list of the available Component Interfaces. An equivalent list is generated by starting PeopleSoft Application Designer, selecting File, Open, Component Interface.

You get a Component Interface collection by using the FindCompIntfcs method with a session object.

First

Syntax

First()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The First method returns the first Component Interface in the Component Interface collection object executing the method. This returns the structure of the Component Interface with only the key fields filled in. The rest of the data is not present. To populate the Component Interface with data, you must set the key values and use the Get method.

Example

```
&MYCI = &CICOLLECTION.First();
```

Item

Syntax

Item(*number*)

Description

The Item returns the Component Interface that exists at the number position in the Component Interface collection executing the method. This returns the structure of the Component Interface with only the key fields filled in. The rest of the data is not present. To populate the Component Interface with data, you must set the key values and use the Get method. *number* takes a number value.

Example

```
For &I = 1 to &COLLECTION.Count;
&MYCI = &COLLECTION.Item(&I);
/* do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next Component Interface in the Component Interface collection object executing the method. You can use this method only after you have used either the First or Item methods: otherwise the system doesn't know where to start. This returns the structure of the Component Interface with only the key fields filled in. To populate the Component Interface with data, you must set the key values and use the Get method.

Example

```
&MYCI = &COLLECTION.Next();
```

Component Interface Collection Property

This section discusses the Count property.

Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of Component Interfaces in the Component Interface collection, as a number.

This property is read-only.

Example

```
&COUNT = &MYCI_COLLECTION.Count;
```

Component Interface Class Methods

This section discusses the Component Interface class methods. The methods are discussed in alphabetical order.

Cancel

Syntax

Cancel()

Description

The Cancel method cancels the instance of the Component Interface object executing the method, rolling back any changes that were made. This sets the Component Interface state to the same state it was in immediately after it was created by GetCompIntfc. This closes the Component Interface.

Parameters

None.

Returns

A Boolean value: True if component was successfully cancelled, False otherwise.

CopyRowset

Syntax

CopyRowset(*&Rowset* [, *InitialRow*] [, *record_list*]);

Where *record_list* is a list of record names in the form:

```
[RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]].  .  .
```

Description

The CopyRowset method copies the specified rowset object to the Component Interface executing the method, copying like-named record fields and data collections (child rowsets) at corresponding levels. If pairs of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

 Copyright © 1988-2007, Oracle. All rights reserved.

CopyRowset uses the Page Field order when copying properties. This helps ensure that dependent fields are set in the required order.

Note. This method works in PeopleCode only. This method uses the names of the records in the collection, *not* the name you may give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not* copied.

Use this method when you are using a Component Interface to verify the data in your application message.

Note. The structure of the rowset you're copying data from must exactly match the existing rowset structure, with the same records at level zero, 1, 2, and so on. CopyRowset is intended to be used with a notification process, that is, with message data. As all notifications work only in two-tier mode, CopyRowset works only in two-tier mode.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify a list of source and target record names. All *source_recnames* are records being copied *from*, in the rowset object being copied from. All *target_recnames* are records being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

The following example would be in your notification PeopleCode. The Exit(1) causes all changes to be rolled back, and the message is marked with the status ERROR so you can correct it.

```
Local message &MSG;
Local ApiObject &SESSION;
Local ApiObject &PO;
Local rowset &ROWSET;

&MSG = %IntBroker.GetMessage();
&ROWSET = &MSG.GetRowset();

&SESSION = %Session;

If &SESSION <> NULL Then
/* Session connected correctly */
/* Set key values to create component */
&PO = &SESSION.GetCompIntfc(COMPINTFC.PO);
&PO.BU = &MSG(&I).PO_HDR.BU.Value;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&PO.PO_ID = &MSG(&I).PO_HDR.PO_ID.Value;
&PO.Create();

&PO.CopyRowset(&ROWSET);
If NOT (&PO.Save()) Then
   Exit(1);
End-if;
Else
/* do error processing */
End-If;
```

The following example loops through all the transactions of a message rowset.

```
Local message &MSG;
Local ApiObject &SESSION;
Local ApiObject &CI;
Local rowset &ROWSET;

&MSG = %IntBroker.GetMessage();
&ROWSET = &MSG.GetRowset();


&SESSION = %Session;
If &SESSION <> Null Then)
   &CI = &SESSION.GetCompIntfc(CompIntfc.VOL_ORG);
   &I = 0;
   While (&I < &ROWSET.RowCount)
      &I = &I + 1;
      &CI.VOLUNTEER_ORG = &ROWSET.GetRow(&I).VOLNTER_ORG_2.  VOLUNTEER_ORG.Value;
      If &CI.Create() Then
         If &CI.CopyRowset(&ROWSET, &I, Record.VOLNTER_ORG_TBL, Record.VOLNTER_ORG_⇒
TBL) Then

         /* App specific processing */

            If Not &CI.Save() Then
               Winmessage("Save Failed");
               /* Other app specific processing */
            End-If;
         End-If;
      End-If;
   &CI.Cancel();
   End-While;
End-If;
```

CopyRowsetDelta

Syntax

CopyRowsetDelta(*&Rowset* [, *InitialRow*] [, *record_list*]);

Where *record_list* is a list of record names in the form:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
[RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]].  .  .
```

Description

The CopyRowsetDelta method copies the changed rows in the specified rowset object to the Component Interface executing the method, copying like-named record fields and data collections (child rowsets) at corresponding levels. If pairs of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

Note. This method works in PeopleCode only. This method uses the names of the records in the collection, *not* the name you give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not*copied.

You will generally use this method with a Component Interface to verify data from a message.

Note. CopyRowsetDelta is intended for a notification process, that is, with message data. As all notifications work only in two-tier mode, CopyRowsetDelta works only in two-tier mode.

If the rowset you're copying from is a message rowset, the CopyRowsetDelta method uses the AUDIT_ACTN field in the PSCAMA table in the message to know whether the row is to be inserted, updated, or deleted inside the Component Interface.

If the rowset you're copying from *is not* a message rowset, that rowset must have the same structure as a message, that is, it must have a PSCAMA record with an AUDIT_ACTN field on every level of the rowset.

Warning! CopyRowsetDelta uses a record's keys to locate the target row to change for all audit actions other than Add. CopyRowsetDelta actions (other than Add) therefore work only on rowsets that have keys *that uniquely identify all rows in the rowset*. Rowsets that do *not*distinguish between rows using a key field will be updated in an unpredictable fashion.

Considerations Using CopyRowsetDelta with Effective-Dated Rowsets

If a message data row inserted using a PSCAMA Audit action of "A" belongs to an effective dated scroll containing child scrolls, the insertion of the parent row causes child rows of the previous effective-dated row to be copied over, and their effective date is updated with that of the inserted parent.

If the message also contains a child row being inserted with a PSCAMA Audit action of "A", the component interface being populated will end up having two child rows: the one inserted as part of the Effective-dated processing and the one inserted using the PSCAMA Audit action "A" in the message.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify a list of source and target record names. All *source_recnames* are records being copied *from*, in the rowset object being copied from. All *target_recnames* are records being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

The following PeopleCode would exist in your notification process. The Exit(1) causes all changes to be rolled back, and the message is marked with the status ERROR so you can correct it.

```
Local Message &MSG;
Local ApiObject &SESSION;
Local ApiObject &PO;
Local Rowset &ROWSET;

&MSG = %IntBroker.GetMessage();
&ROWSET = &MSG.GetRowset();

&SESSION = %Session;

If &SESSION <> Null Then
/* Session connected correctly */
/* Set key values to get component */
&PO = &SESSION.GetCompIntfc(COMPINTFC.PO);
&PO.BU = &MSG(&I).PO_HDR.BU.Value;
&PO.PO_ID = &MSG(&I).PO_HDR.PO_ID.Value;
&PO.Get();


&PO.CopyRowsetDelta(&ROWSET);
If NOT (&PO.Save()) Then
   Exit(1);
End-if;
Else
   /* do error processing */
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Understanding Supported Message Structures," PSCAMA

CopySetupRowset

Syntax

```
CopySetupRowset(&Rowset [, InitialRow] [, record_list]);
```

Where *record_list* is a list of record names in the form:

```
RECORD.source_recname, RECORD.target_recname
```

Description

The CopySetupRowset method is used to copy a setup table application message to a Component Interface. A setup table has the same record at level zero and level one, while a Component Interface has only a single copy of a record. This method copies the contents of the message (at level zero) to the first level collection (level one) of the Component Interface.

The CopySetupRowset method copies *like-named* record fields. If a pair of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

Note. This method works in PeopleCode only. This method uses the names of the records in the collection, *not* the name you may give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not* copied.

Note. CopySetupRowset is for a notification process, that is, with message data. As all notifications work only in two-tier mode, CopySetupRowset works only in two-tier mode.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify source and target record names. The *source_recname* is the record being copied *from*, in the rowset object being copied from. The *target_recname* is the record being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

The following example would be in your Notification PeopleCode.

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local ApiObject &SESSION;
Local ApiObject &PSMESSAGES;

Local ApiObject &CI;

Local Message &MSG;
Local Rowset &RS;

&SESSION = %Session;
&PSMESSAGES = &SESSION.psmessages;

&MSG = %IntBroker.GetMessage();
&RS = &MSG.GetRowset();

&CI = &SESSION.getcomponent(Component.VOL);

/** Set Business Component Properties **/
&CI.gethistoryitems = True;
/*&CI.InteractiveMode = True;*/
/*&CI.stoponfirsterror = True;*/

For &TRANSACTION = 1 To &RS.RowCount

   &CI.VOLUNTEER_ORG = &RS(&TRANSACTION).VOLNTER_ORG_TBL.VOLUNTEER_ORG.Value;

   /* note: You can achieve better performance if you add code here to check  if⇒
 the keys are the same L0 keys as the last time  through the loop and, if so,skip⇒
 the Get/Create section. */

   If Not &CI.create() Then
      &PSMESSAGES.DeleteAll();
      If Not &CI.get() Then
         /** Check Error Messages **/
         For &I = 1 To &PSMESSAGES.count
            &TYPE = &PSMESSAGES.item(&I).type;
            &TEXT = &PSMESSAGES.item(&I).text;
         End-For;
         Exit(1);

      End-If;
   End-If;

   If Not &CI.CopySetupRowset(&RS, &TRANSACTION) Then
      /** Check Error Messages **/
      For &I = 1 To &PSMESSAGES.count
         &TYPE = &PSMESSAGES.item(&I).type;
         &TEXT = &PSMESSAGES.item(&I).text;
      End-For;
      Exit(1);
   End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
      If Not &CI.Save() Then
         /** Check Error Messages **/
         For &I = 1 To &PSMESSAGES.count
            &TYPE = &PSMESSAGES.item(&I).type;
            &TEXT = &PSMESSAGES.item(&I).text;
         End-For;
         Exit(1);
      End-If;

      &CI.Cancel();
   End-For;
```

See Also

Chapter 12, "Component Interface Classes," CopySetupRowsetDelta, page 433 and Chapter 12, "Component Interface Classes," CopyRowset, page 426.

CopySetupRowsetDelta

Syntax

CopySetupRowsetDelta(*&Rowset* [, *InitialRow*] [, *record_list*]);

Where *record_list* is a list of record names in the form:

RECORD.*source_recname*, RECORD.*target_recname*

Description

The CopySetupRowsetDelta method is used to copy a setup table with *changed* rows in an application message to a Component Interface. A setup table has the same record at level zero and level one, while a Component Interface has only a single copy of a record. This method copies the contents of the message (at level zero) to the first level collection (level one) of the Component Interface.

Note. This method works in PeopleCode only. CopySetupRowsetDelta copies all the like-named fields from the changed *row* into the message. It is *not* copying only the changed fields.

The CopySetupRowsetDelta method copies *like-named* record fields, in those rows where the IsChanged property is True. If a pair of source and destination record names are given, these are used to pair up the records and data collections *before* checking for like-named record fields.

Note. This method uses the names of the records in the collection, *not* the name you may give the collection when you create the Component Interface.

If there are blanks in *source* rowset or record, they are copied over to the Component Interface only if the field's IsChanged property is set to True. Otherwise blanks are *not* copied.

Note. CopySetupRowsetDelta is for a notification process, that is, with message data. As all notifications work only in two-tier mode, CopySetupRowsetDelta works only in two-tier mode.

Copyright © 1988-2007, Oracle. All rights reserved.

Warning! CopySetupRowsetDelta uses a record's keys to locate the target row to change for all audit actions other than Add. CopySetupRowsetDelta actions (other than Add) therefore work only on rowsets that have keys that *uniquely identify all rows in the rowset*. Rowsets that do *not*distinguish between rows using a key field will be updated in an unpredictable fashion.

Parameters

Parameter	Description
&Rowset	Specify an existing, instantiated rowset object that contains data.
InitialRow	Specify the initial transaction row to begin with. This parameter provides a quick way to loop through a rowset that has multiple level zero rows. The default value for this parameter is 1.
record_list	Specify source and target record names. The *source_recname* is the record being copied *from*, in the rowset object being copied from. The *target_recname* is the record being copied *to*, in the Component Interface being copied to.

Returns

None.

Example

```
Local ApiObject &SESSION;
Local ApiObject &PSMESSAGES;

Local ApiObject &CI;

Local Message &MSG;
Local Rowset &RS;

&SESSION = %Session;
&PSMESSAGES = &SESSION.psmessages;

&MSG = %IntBroker.GetMessage();
&RS = &MSG.GetRowset();

&CI = &SESSION.getcomponent(Component.VOL);

/** Set Business Component Properties **/
&CI.gethistoryitems = True;
/*&CI.InteractiveMode = True;*/
/*&CI.stoponfirsterror = True;*/

For &TRANSACTION = 1 To &RS.RowCount

   &CI.VOLUNTEER_ORG = &RS(&TRANSACTION).VOLNTER_ORG_TBL.VOLUNTEER_ORG.Value;

   /* note: You can achieve much better performance if you add code here to check ⇒
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
if the keys are the same L0 keys as the last time through the loop and, if so,⇒
skip the Get/Create section. */

  If Not &CI.create() Then
     &PSMESSAGES.DeleteAll();
     If Not &CI.get() Then
        /** Check Error Messages **/
        For &I = 1 To &PSMESSAGES.count
           &TYPE = &PSMESSAGES.item(&I).type;
           &TEXT = &PSMESSAGES.item(&I).text;
        End-For;
        Exit(1);

     End-If;
  End-If;

  If Not &CI.CopySetupRowsetDelta(&RS, &TRANSACTION) Then
     /** Check Error Messages **/
     For &I = 1 To &PSMESSAGES.count
        &TYPE = &PSMESSAGES.item(&I).type;
        &TEXT = &PSMESSAGES.item(&I).text;
     End-For;
     Exit(1);
  End-If;

  If Not &CI.Save() Then
     /** Check Error Messages **/
     For &I = 1 To &PSMESSAGES.count
        &TYPE = &PSMESSAGES.item(&I).type;
        &TEXT = &PSMESSAGES.item(&I).text;
     End-For;
     Exit(1);
  End-If;

  &CI.Cancel();
```

See Also

Chapter 12, “Component Interface Classes,” CopySetupRowset, page 431 and Chapter 12, “Component Interface Classes,” CopyRowsetDelta, page 428.

Create

Syntax

```
Create()
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Create method associates the Component Interface object executing the method with a new, open Component Interface that matches the key values that were set prior to using the Create method. If there are CREATEKEYS values associated with the Component Interface, these are the values you must set. If there are no CREATEKEYS values, you must set the required GETKEYS values instead. All keys required for creating a new Component Interface must be set before using the Create method, otherwise you receive a runtime error. If you do not use unique key values, (that is, you try to set the keys to values that already exist in the database) you receive a runtime error.

Setting the key values prior to using the Create method is analogous to filling in the key values in the Add dialog for a component when you access it in add mode.

Parameters

None.

Returns

A Boolean value: True if component was successfully created, False otherwise.

Example

```
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.ACTION_REASON);
&MYCI.ACTION = "Additional Job";
&MYCI.ACTION_REASON = "0001";
&MYCI.Create();
```

Find

Syntax

```
Find()
```

Description

Note. Find can be used only at level zero within the Component Interface.

You do not have to set values for all the key values. You can use the same wildcards in your Find that you can use in the search dialog, that is, % for one or more characters in a search pattern, and _ (underscore) for exactly one character. In addition, you can use partial values. For example, the following code finds all the data items where the employee ID starts with an "8":

```
&MYCI.Emplid = "8";
&MYDC = &MYCI.Find();
```

This is analogous to using a partial key from the search dialog, opening a component.

After you have a data collection, you can use one of the data collection methods to open the Component Interface.

Parameters

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A collection of Component Interfaces.

See Also

Chapter 12, "Component Interface Classes," Data Collection Methods, page 443

Get

Syntax

Get()

Description

The Get method associates the Component Interface object executing the method with an open Component Interface that matches the key values that were set prior to using the Get method. The key values you must set are the required GETKEYS values for the Component Interface.

Note. To retrieve all the history data for a Component Interface, you must specify the GetHistoryItems property as True *before* you use the Get method. If you want any PeopleCode programs associated with the fields to fire immediately after a value is changed, you must set the InteractiveMode property as True *before*you use the Get method.

After any execution of Get, you should check if there are any errors pending on the session object. In some special circumstances (involving failure of previously cached operations failing after the Get has executed) Get returns True even though the component wasn't retrieved.

Parameters

None.

Returns

A Boolean value: True if component was successfully retrieved, False otherwise.

Example

```
&MYCI.EMPLID = "8001";
&MYCI.Get();
If %Session.ErrorPending Then
   /* Get Unsuccessful, do error processing */
Else
   /* do regular processing */
End-if;
```

GetPropertyByName

Syntax

GetPropertyByName(*string*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The GetPropertyByName method returns the value of the property specified by *string*. For a collection, it returns a reference to the collection. Generally this function is used only in applications that cannot get the names of the component interface properties until runtime.

Parameters

Parameter	Description
string	The name of the property.

Returns

String. The value of the property.

ApiObject. For a collection. The value of the property.

Example

```
Local ApiObject &oSession, &oCI;
Local array of string &Keys_Arr, &Temp_Arr, &Prop_Arr;
Local string &strCIName, &PropertyValue;
Local number &I, &J, &K;

Function getPropertyValue(&oDataColl As ApiObject, &PropertyName As string)⇒
 Returns string
rem ***** Return property value
   Return &oDataColl.GetPropertyByName(&PropertyName);
End-Function;

Function getCollection1(&collectionName As string)
   Local ApiObject &oL1_DataColl, &oL1_DataItem;
rem ***** Return collection
   &oL1_DataColl = &oCI.GetPropertyByName(&collectionName);
   For &J = 1 To &oL1_DataColl.Count
      &oL1_DataItem = &oL1_DataColl.Item(&J);
      For &K = 1 To &Prop_Arr.Len
         &Temp_Arr = Split(&Prop_Arr [&K], "|");
         If &Temp_Arr [1] = "1" Then
            If &Temp_Arr [3] = "Property" Then
               &PropertyValue = getPropertyValue(&oL1_DataItem,
                   &Temp_Arr [2]);
            Else
               rem ***** Code to Get Collection 2 goes here *****;
            End-If;
         End-If;
      End-For;
   End-For;
End-Function;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 12, “Component Interface Classes,” SetPropertyByName, page 439

Save

Syntax
Save()

Description

Saves any changes that have been made to the data of the Component Interface executing the method. You must save any new Component Interfaces you create before they are added to the database.

The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on.) is executed after you execute this method. If you didn’t specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time.

If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, you may want to delete it from the PSMessages collection.

Note. If you’re running a Component Interface from an Application Engine program, the data won’t actually be committed to the database until the Application Engine program performs a COMMIT.

Parameters

None.

Returns

A Boolean value: True if component was successfully saved, False otherwise.

Example

```
&MYCI.Emplid = "8001";
&MYCI.Get();
&MYCI.CHECKLIST_CD = "00001";
&MYCI.Save;
```

See Also
Chapter 12, “Component Interface Classes,” InteractiveMode, page 443

Chapter 36, “Session Class,” ErrorPending, page 1519

SetPropertyByName

Syntax
SetPropertyByName(*string, value***)**

Description

The SetPropertyByName method sets the value of the property specified by *string*. Generally this function is used only in applications that cannot set the names of the component interface properties until runtime.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
string	The name of the property.
value	The value to which the property is to be set.

Returns

None.

Example

```
Local ApiObject &oSession, &oCI;
Local array of string &Keys_Arr, &Temp_Arr, &Prop_Arr;
Local string &strCIName, &PropertyValue;
Local number &I, &J, &K;

&strCIName = "SDK_BUS_EXP";
&Keys_Arr = CreateArrayRept("", 0);
&Keys_Arr.Push("SDK_EMPLID|" | "8001");

&oSession = %Session;
&oCI = &oSession.GetCompIntfc(@("CompIntfc." | &strCIName));

For &I = 1 To &Keys_Arr.Len
   &Temp_Arr = Split(&Keys_Arr [1], "|");
   &oCI.SetPropertyByName(&Temp_Arr [1], &Temp_Arr [2]);
End-For;
```

See Also

Chapter 12, “Component Interface Classes,” GetPropertyByName, page 437

Component Interface Class Properties

In this section, we discuss the Component Interface class properties. The properties are discussed in alphabetical order.

ComponentName

Description

This property returns the name of the Component Interface, as defined in PeopleSoft Application Designer, as a string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

CreateKeyInfoCollection

Description

This property returns a CompIntfPropInfoCollection collection that contains a CompIntfPropInfoCollection object for every key in CREATEKEYS.

This property is read-only.

Example

```
&CREATEKEYS = &CI.CreateKeyInfoCollection;
```

See Also

Chapter 12, “Component Interface Classes,” Accessing the Structure of a Component Interface, page 451

EditHistoryItems

Description

This property works with the GetHistoryItems property to specify what data is accessed, and whether you can edit that data:

- If EditHistoryItems is False (the default) and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update/Display All mode. This means all history rows are returned, however, you can edit only rows with a date set in the future.
- If EditHistoryItems is True and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Correction mode. This means all history rows are returned, and you can edit them.
- If GetHistoryItems is False, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update mode. The EditHistoryItems has no effect when GetHistoryItems is False.

You must set this property to True *before* you execute the Get method.

This property is read-write.

See Also

Chapter 12, “Component Interface Classes,” GetHistoryItems, page 442

FindKeyInfoCollection

Description

This property returns a CompIntfPropInfoCollection collection that contains a CompIntfPropInfoCollection object for every key in FINDKEYS.

This property is read-only.

See Also

Chapter 12, “Component Interface Classes,” Accessing the Structure of a Component Interface, page 451

Copyright © 1988-2007, Oracle. All rights reserved.

GetDummyRows

Description

When a new scroll is inserted on a page, that scroll is displayed even though it has no underlying data. Any scroll that is empty has one dummy row displayed with only the defaults set.

This property is True if the dummy row is to be displayed, False if you do not want to display the dummy row. The default value for this property is True.

This property is read-write.

Example

```
&MyCI.GetDummyRows = False;
```

GetHistoryItems

Description

This property works with the EditHistoryItems property to specify what data is accessed, and whether you can edit that data:

- If EditHistoryItems is False (the default) and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update/Display All mode. This means all history rows are returned, however, you can edit only rows with a date set in the future.
- If EditHistoryItems is True and GetHistoryItems is True, you access the data in the Component Interface in a similar manner as if you were accessing a component in Correction mode. This means all history rows are returned, and you can edit them.
- If GetHistoryItems is False, you access the data in the Component Interface in a similar manner as if you were accessing a component in Update mode. The EditHistoryItems has no effect when GetHistoryItems is False.

You must set this property to True *before* you execute the Get method.

This property is read-write.

Example

The following example checks the current status of the mode, then sets the GetHistoryItems and EditHistoryItems properties to True if the mode is Correction mode.

```
If %Mode = "C" Then
   &CI.EditHistoryItems = True;
   &CI.GetHistoryItems = True;
End-if;
```

See Also

Chapter 12, “Component Interface Classes,” EditHistoryItems, page 441

GetKeyInfoCollection

Description

This property returns a CompIntfPropInfoCollection collection that contains a CompIntfPropInfoCollection object for every key in GETKEYS.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

See Also
Chapter 12, “Component Interface Classes,” Accessing the Structure of a Component Interface, page 451

InteractiveMode

Description

When this property is set as True, the Component Interface runs the same as a component: that is, any PeopleCode programs associated with FieldChange, RowInsert, and so on, run immediately after you make a change. If this property is set to False, these programs won’t run until you execute the Save method.

This property is read-write.

See Also
Chapter 36, “Session Class,” ErrorPending, page 1519

PropertyInfoCollection

Description

This property returns a CompIntfPropInfoCollection object for every property that isn’t a collection (that is, a scroll.) If the property is a collection (scroll), use the PropertyInfoCollection property to get another collection.

This property is read-only.

See Also
Chapter 12, “Component Interface Classes,” Accessing the Structure of a Component Interface, page 451

StopOnFirstError

Description

When this property is set as True, the CopyRowset (or CopyRowsetDelta) method currently executing halts its processing at the first error generated by the Component Interface.

When this property is set as True and InteractiveMode is set as False, processing of queued operations at save time is halted at the first error.

The default value is False.

This property is read-write.

Data Collection Methods

A *data collection* is the collection of data, available at runtime or during test mode, associated with a particular scroll (or record.) The data collection object returns information about every *row of data* (item) that is returned for that record at runtime.

To access a data collection, use the name of the record (scroll) as a property on a Component Interface.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 12, "Component Interface Classes," Traversing a Component Interface and Using Data Collections, page 419

CurrentItem

Syntax

```
CurrentItem()
```

Description

If the component associated with the Component Interface is effective-dated, CurrentItem returns a reference to the current effective-dated item (row of data). To get a specific item based on a date, use GetEffectiveItem.

If there is no current item, a Null is returned.

Example

```
&MYCD = &MYCI.EMPL_CHKLST_ITM;
&ITEM = &MYDC.CurrentItem();
```

DeleteItem

Syntax

```
DeleteItem(number)
```

Description

The DeleteItem method deletes the item (row of data) at the position specified by *number*. This parameter takes a number value.

When the DeleteItem method is executed, if there are any RowDelete PeopleCode programs associated with any of the fields, they fire as well, as if the user pressed ALT+8 and ENTER or clicked the DeleteRow icon. However, the programs are executed as if turbo mode was selected. (In turbo mode default processing is performed *only* for the row being deleted.)

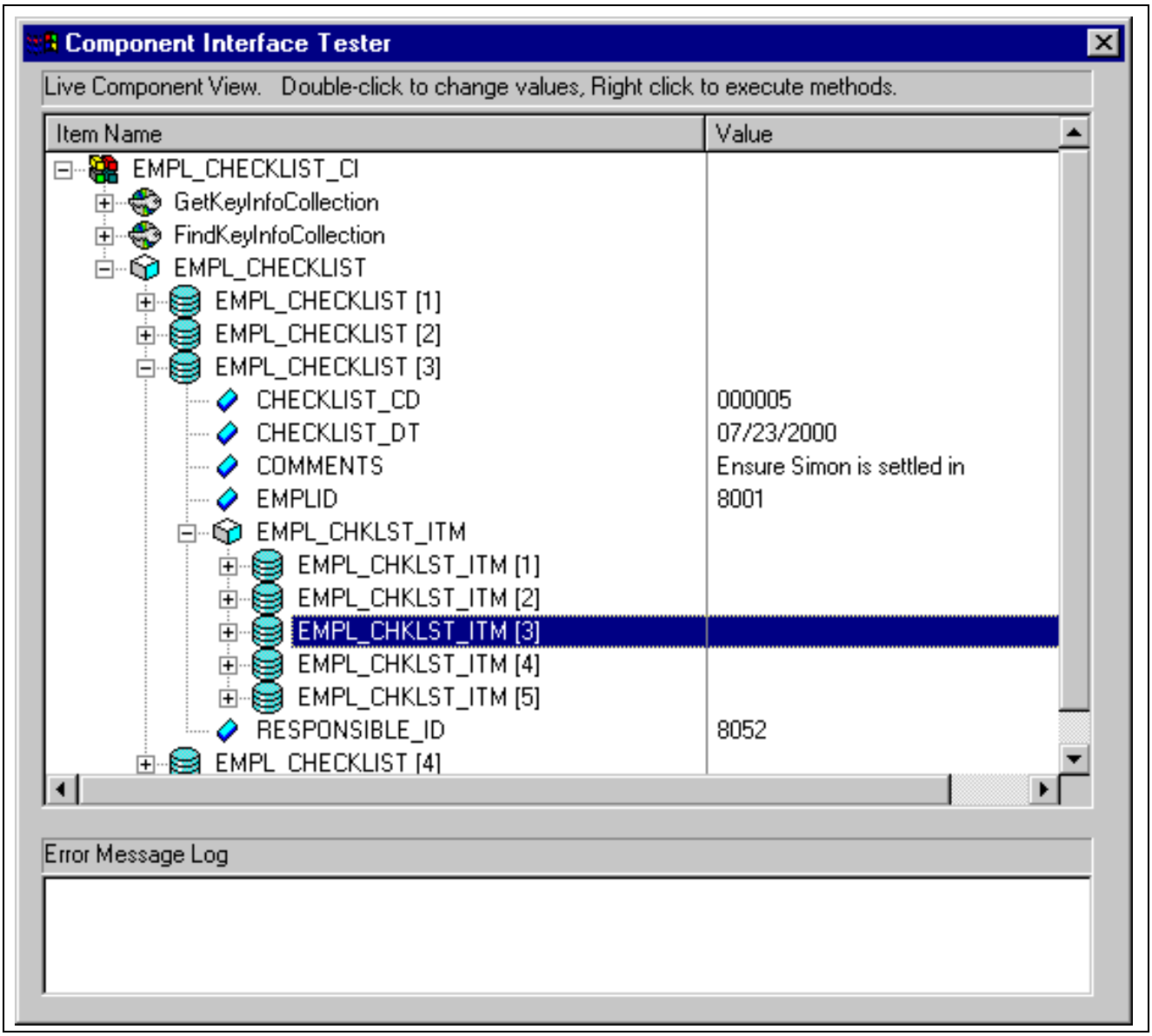
If you set the InteractiveMode property to True, any RowDelete PeopleCode runs immediately after you execute DeleteItem. If this property is set to False, any RowDelete PeopleCode runs after you execute the Save method.

The deleted item is not actually deleted from the database until after you use the Save method.

Example

For example, suppose your Component Interface had two scrolls: EMPL_CHECKLIST and EMPL_CHKLST_ITM. The data collection EMPL_CHECKLIST has four items (rows of data.) The data collection EMPL_CHKLST_ITM (under the third item) has five items (rows of data.) If you run this component in the Component Interface Tester, it would look as follows:

 Copyright © 1988-2007, Oracle. All rights reserved.



EMPL_CHK_BC in Component Interface Tester

To delete the third row in the third item, use the following:

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHK_BC);
&MYLVL1 = &MYCI.EMPL_CHECKLIST;
&ITEM2 = &MYLVL1.Item(3);
&MYLVL2 = &ITEM2.EMPL_CHKLST_ITM;
&MYLVL2.DeleteItem(3);
&MYCI.Save();
```

Copyright © 1988-2007, Oracle. All rights reserved.

GetEffectiveItem

Syntax

GetEffectiveItem(*DateString, SeqNo*)

Description

If the component associated with the Component Interface is effective-dated, GetEffectiveItem returns a reference to the closest effective-dated item (row of data) that is less than or equal to the date specified by the *DateString*. To get an item based on the current effective date, use CurrentItem.

Note. *DateString* takes only a string value. You must convert a date variable into a string before you can use it for *DateString*. You can use the String function to do this.

Parameters

Parameter	Description
DateString	Specify the year, month, and day of the effective date that you want to look for. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or*MM/DD/YY.
SeqNo	Specify the sequence number of the effective date that you want to look for. This parameter takes a number value.

Returns

A reference to an effective-dated item.

Example

```
&MYCD = &MYCI.EMPL_CHKLST_ITM;
&DSTRING = String(&MYDATE);
&ITEM = &MYDC.GetEffectiveItem(&DSTRING, 1);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," String

GetEffectiveItemNum

Syntax

GetEffectiveItemNum(*DateString, SeqNo*)

Description

If the component associated with the Component Interface is effective-dated, GetEffectiveItemNum returns a reference to the number of the closest effective-dated item (row of data) that is less than or equal to the date specified by the *DateString*. To get an item number based on the current effective date, use CurrentItemNum.

Note. *DateString* takes only a string value. You must convert a date variable into a string before you can use it for *DateString*. You can use the String function to do this.

Copyright © 1988-2007, Oracle. All rights reserved.

Considerations for Returning Rows

GetEffectiveItemNum returns a valid row number only when EFFDT is less than or equal to *DateString*. If the value you use for *DateString* pre-dates all the rows in the data collection, this method returns a zero and logs a message in the PSMessages collection.

For example, if 12/01/1990 was the earliest date in the collection, the following would return zero:

```
&NUM = &MYDC.GetEffectiveItemNum("01/01/1900", 1);
```

Parameters

Parameter	Description
DateString	Specify the year, month, and day of the effective date that you want to look for. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or*MM/DD/YYYY.
SeqNo	Specify the sequence number of the effective date that you want to look for. This parameter takes a number value.

Returns

A number. This method returns 0 if no row matching the criteria is found.

Example

```
&MYCD = &MYCI.EMPL_CHKLST_ITM;
&DSTRING = String(&MYDATE);
&ITEMNUM = &MYDC.GetEffectiveItemNum(&DSTRING, 1);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," String

InsertItem

Syntax

InsertItem(*number*)

Description

The InsertItem method inserts the item (row of data) at the position specified by *number*. This parameter takes a number value. You can insert items below only the zero level scroll. If you need to add a new data item, use the Create method instead.

InsertItem adds the new row *after* the current row. If the row has an effective date (EFFDT) or an effective sequence (EFFSEQ), these values are copied into the new row.

If you specify -1 for *number*, InsertItem inserts a new item (row) after the *last* row.

The InsertItem method returns a reference to the newly created item (row of data).

When the InsertItem method is executed, if there are any RowInsert PeopleCode programs associated with any of the fields, they fire also, as if the user pressed ALT+7 and ENTER or clicked the InsertRow icon. However, the programs are executed as if turbo mode was selected. (In turbo mode default processing is performed *only* for the row being inserted.)

Copyright © 1988-2007, Oracle. All rights reserved.

If you set the InteractiveMode property to True, any RowInsert PeopleCode runs immediately after you execute InsertItem. If this property is set to False, any RowInsert PeopleCode runs after you execute the Save method.

The inserted item is not added to the database until after you use the Save method.

Example

In the following example a new item (row of data) is added at the *end* of the current collection.

```
&MYDC = &MYCI.EMPL_CHECKLIST;
&COUNT = &MYDC.Count;
&ITEM = &MYDC.InsertItem(&COUNT);
&ITEM.CHECKLIST_CD = "00001";
&ITEM.RESPONSIBLE_ID = "6609";
&RSLT = &MYCI.Save();
```

Item

Syntax

Item(*number*)

Description

The Item returns the item (row of data) that exists at the *number* position in the data collection executing the method. The parameter takes a number value.

ItemByKeys

Syntax

ItemByKeys(*key_values*)

Description

The ItemByKeys method returns the item specified by the parameters. The number and type of keys are unique to each specific collection. Each key must be separated by a comma.

The collection reference must be the name of the Component Interface, followed by the record name. This method won't work on a collection reference (that is, &CI.RECNAME.ItemByKeys, not &MYCOLLECTION.ItemByKeys).

After you've returned an item, use the ItemNum property to determine the number of the item.

The keys must be in the *exact* order as in the Component Interface. A second level data collection also contains the keys of the parent data collection.

An easy way to determine the keys and their order in PeopleCode is to open the Component Interface in PeopleSoft Application Designer, and use the Test Component. To determine the keys in Visual Basic, use the Object Browser.

See Chapter 12, "Component Interface Classes," ItemNum, page 450.

To see the signature for ItemByKeys:

1. Open the Component Interface in PeopleSoft Application Designer.
2. Start the Component Interface Tester.

 Copyright © 1988-2007, Oracle. All rights reserved.

Select the open Component Interface, then right-click, and select Test Component Interface from the pop-up menu.

3. Instantiate an object.

   Add data to the Get or Create keys and click Get Existing or Create New, respectively.

4. Expand the instantiated component until you find the collection in which you're interested.
5. Right-click on the collection and select ItemByKeys from the resulting pop-up menu.
6. The dialog that follows shows you the specific parameters, types, and order in which you should call ItemByKeys.

Returns

An item (row) of data from a data collection.

Example

```
Local ApiObject &MYSESSION;
Local ApiObject &CI;
Local ApiObject &CI_COLLECTION;
Local ApiObject &CI_ITEM;

&MYSESSION = %Session;
&CI = &MYSESSION.GetCompIntfc(COMPINTFC.CM_EVAL);
&CI.EMPLID = "8001";
If &CI.Get() <> 1 Then
   Exit;
End-If;

&CI_COLLECTION = &CI.CM_EVALUATIONS;
&COUNT = &CI_COLLECTION.Count;

&CI_ITEM = &CI.CM_EVALUATIONS.itembykeys("01");
&CI_ITEM.DESCR50 = "TEST";
If &CI.Save() <> 1 Then
   Exit;
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Component Interfaces, "Developing the Component Interface," Testing the Component Interface

Data Collection Properties

This section explains the following Data Collection properties:

- Count
- CurrentItemNumber

Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of data items (rows of data) in the data collection.

This property is read-only.

Example

```
&CI = &MYSESSION.GetCompIntfc(COMPINTFC.CM_EVAL_BC);
&CI.EMPLID = "8001";
&CI.Get()

&CI_COLLECTION = &CI.CM_EVALUATIONS;
&COUNT = &CI_COLLECTION.Count;
```

CurrentItemNumber

Description

If the component associated with the Component Interface is effective-dated, this property returns the item number for the current effective-dated item (row of data).

This property is read-only.

Data Item Class Property

This section discusses the ItemNum property.

ItemNum

Description

This property returns the number of the data item (row) in the collection. For example, many of the data collection methods takes a number as a parameter. Use this property to determine the item number (row number) of an item in a collection, then use that number in another method.

This property is read-only.

Example

```
Evaluate USER_ACTION

.  .  .

When = "D"
   &CI_ITEM = &CI_LVL1_NAMES.ItemByKeys(&NAME_TYPE, &NAME_PART);
   If &CI_ITEM <> Null then
      &I = &CI_LVL1_NAMES.ItemNum;
      &CI_LVL1_NAMES.DeleteItem(&I);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
      End-if;
   .  .  .
End-Evaluate;
```

Accessing the Structure of a Component Interface

The structure of a Component Interface can be accessed using a CompIntfPropInfoCollection object. You access a CompIntfPropInfoCollection object from a CompIntfPropInfoCollection collection. There is more than one way to instantiate a CompIntfPropInfoCollection collection.

Note. You don't have to populate a Component Interface with data before you access the structure. You can access the structure of a Component Interface immediately after you use the GetCompIntfc method with a session object. Accessing the structure of a Component Interface before you populate it with data may increase your performance.

See Chapter 12, "Component Interface Classes," GetCompIntfc, page 424.

CompIntfPropInfoCollection Collection

There are two types of CompIntfPropInfoCollection object properties: *field* properties and *collection* properties.

A *field* property maps to a specific record field. You can access structural information about the field using a CompIntfPropInfoCollection object. This information includes the name of the field, whether it's required, is it based on a prompt table, and so on.

A *collection* property is just that, a collection of properties. And before you can access a CompIntfPropInfoCollection object, you must first get a CompIntfPropInfoCollection *collection.* The following are the valid types of CompIntfPropInfoCollection collections:

- CREATEKEYS, GETKEYS and FINDKEYS

  When you create a component, you must specify the search record to be used with that component. You can also specify an alternate search record to be used when the component is accessed in Add mode. The key fields from those records make up GETKEYS, FINDKEYS, and CREATEKEYS collections.

  - CREATEKEYS: A collection containing the key fields of the search record specified to be used in Add mode. Use the CreateKeyInfoCollection property to instantiate the CompIntfPropInfoCollection collection.
  - GETKEYS: A collection containing the primary required key fields from the primary search record. Use the GetKeyInfoCollection property to instantiate the CompIntfPropertInfo collection.
  - FINDKEYS: A collection containing the key fields and the alternate key fields from the primary search record. Use the FindKeyInfoCollection property to instantiate the CompIntfPropertInfo collection.

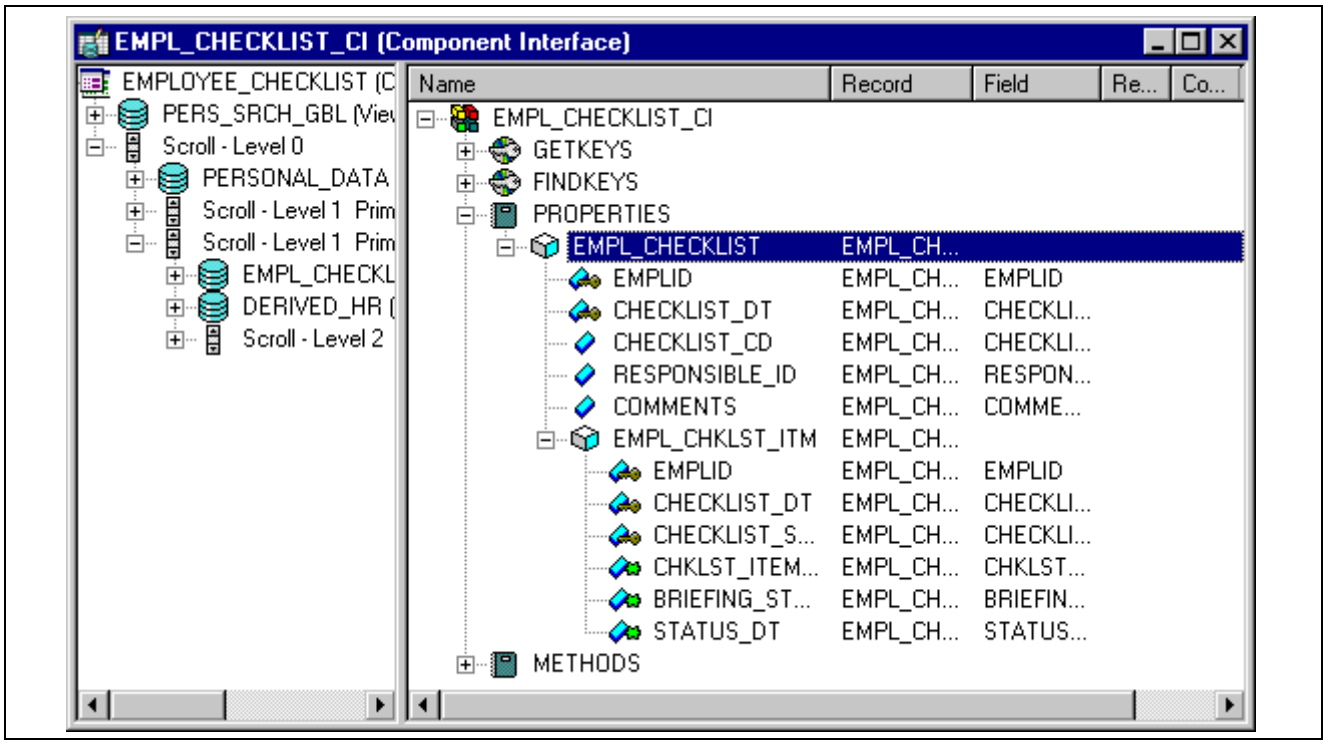
- A page scroll

  The fields associated with a page scroll are in this type of collection. This may or may not be all the fields associated with the record. Use the PropertyInfoCollection property to instantiate this kind of collection.

If the page the Component Interface is based on contains a secondary scroll, you can check the Type property to determine if the object is a CompIntfPropInfoCollection object (that is, a field), or a scroll. Then, to get the properties of the fields associated with that secondary scroll, you can use the PropertyInfoCollection property on the CompIntfPropInfoCollection object.

For example, the following Component Interface has a level zero and level one scroll.

Copyright © 1988-2007, Oracle. All rights reserved.



Component Interface with secondary scroll

The level zero scroll is made up of three fields: CHECKLIST_CD, RESPONSIBLE_ID, COMMENTS, and a level one scroll, EMPL_CHKLST_ITM. The CompIntfPropInfoCollection collection for this Component Interface would have four items in it. Three would be CompIntfPropInfoCollection objects (the three fields.) The last item, EMPL_CHKLST_ITM would *not* be a valid CompIntfPropInfoCollection object. You can use the IsCollection property to verify if an item in a collection is itself a collection or a valid CompIntfPropInfoCollection object.

To access the fields in a lower level scroll, you must use the PropertyInfoCollection property first, to return a collection of those fields.

The following example loops through a Component Interface. It pulls out the names of the properties in the first three levels of a Component Interface. If the property is a nested collection, it prefixes the ancestor collection name to the property name.

```
&MYSESSION = %Session;
&CI = &MYSESSION.GetCompIntfc(COMPINTFC.VOLNEW);
&PROPINFO_0 = &CI.PropertyInfoCollection;
For &I = 1 To &PROPINFO_0.Count;
   &PROPITEM_0 = &PROPINFO_0.Item(&I);
   Warning (&PROPITEM_0.Name);
   If (&PROPITEM_0.IsCollection) Then
      &PROPINFO_1 = &PROPITEM_0.PropertyInfoCollection;
      For &J = 1 To &PROPINFO_1.Count;
         &PROPITEM_1 = &PROPINFO_1.Item(&J);
         &S1 = &PROPITEM_0.Name | "." | &PROPITEM_1.Name;
         Warning (&S1);
         If (&PROPITEM_1.IsCollection) Then
            &PROPINFO_2 = &PROPITEM_1.PropertyInfoCollection;
            For &K = 1 To &PROPINFO_2.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
                &PROPITEM_2 = &PROPINFO_2.Item(&K);
                &S1 = &PROPITEM_0.Name | "." | &PROPITEM_1.Name | "." | &PROPITEM_⇒
2.Name;
                Warning (&S1);
             End-For;
          End-If
       End-For;
    End-If;
End-For;
```

CompIntfPropInfoCollection Collection Methods

This section explains the CompIntfPropInfoCollection collection methods.

First

Syntax

First()

Description

The First method returns the first CompIntfPropInfoCollection object in the CompIntfPropInfoCollection collection object executing the method.

Example

```
&CIINFO = &CIPROPCOLL.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns a CompIntfPropInfoCollection object that exists at the *number* position in the CompIntfPropInfoCollection collection executing the method

If the specified item is *itself* a collection, the CompIntfPropInfoCollection object that gets returned is invalid (set to NULL). You can use the PropertyInfoCollection property to return a collection of CompIntfPropInfoCollection objects for the collection.

Example

```
&SCROLL1 = &MYCI.PropertyInfoCollection;
For &I = 1 to &SCROLL1.Count;
&PROPERTY = &SCROLL1.Item(&I);
If &PROPERTY.IsCollection Then
&SCROLL2 = &PROPERTY.PropertyInfoCollection;
/*do scroll 2 processing */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Else
/* do scroll 1 processing */
End-If;
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next CompIntfPropInfoCollection object in the CompIntfPropInfoCollection collection object executing the method. You can use this method only after you have used either the First or Item methods: otherwise the system doesn't know where to start.

CompIntfPropInfoCollection Collection Properties

This section explains the Count property.

Count

Description

This property returns the number of CompIntfPropInfoCollection objects in the CompIntfPropInfoCollection collection object executing the method.

This property is read-only.

Example

```
&COUNT = &MYCIPROPINFOC.Count;
```

CompIntfPropInfoCollection Object Properties

This section explains the CompIntfPropInfoCollection Object properties.

Format

Description

This property returns the field format for the object executing the property (that is, name, phone, zip, SSN, and so on) as a number. The values are:

 Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
0	No Format
1	Name
2	Phone
3	Zip
4	SSN
5	Routine
6	Mixed Case
7	Raw Binary
8	Number only
9	SIN
10	Phone International
11	Zip International
12	Seconds
13	Microseconds
14	Custom

This property is read-only.

IsCollection

Description

This property returns True if the object executing the property is a data collection, False otherwise. If IsCollection is True, other field-oriented properties like Required, Type, Xlat, YesNo, Prompt and Format are undefined. If IsCollection is False, the object represents a field and all the previous properties are defined as described.

This property is read-only.

IsReadOnly

Description

This property returns True if the property marked Read Only in the Component Interface Definition; False otherwise.

This property is read-only.

Key

Description

This property returns True if the object executing the property is a key, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

LabelLong

Description

This property returns the record field LongName value as a string. If there is a component override for this value, it is *not* included.

This property is read-only.

LabelShort

Description

This property returns the record field ShortName value as a string. If there is a component override for this value, it is *not* included.

This property is read-only.

Length

Description

This property returns the length of the field property as a number. If the property is a collection, this property returns a zero.

The length of the field is calculated by converting it to a string then getting the length of the string. This means that *everything* in the field is counted as a character, including spaces, date time separation characters, decimal points, sign indicators, and so on.

For example, the following string has 10 characters:

```
01/01/2001
```

The following string has five characters:

```
10.10
```

The following string has six characters:

```
-10.10
```

This property is read-only.

Name

Description

This property returns the name of the object executing the property as a string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Prompt

Description

This property returns True if the object executing the property is associated with a prompt table, False otherwise.

This property is read-only.

PropertyInfoCollection

Description

This property returns another CompIntfPropInfoCollection collection if the object executing the property is a collection.

This property is read-only.

Example

```
&SCROLL1 = &MYCI.PropertyInfoCollection;
For &I = 1 to &SCROLL1.Count;
   &PROPERTY = &SCROLL1.Item(&I);
   If &PROPERTY.IsCollection Then
   &SCROLL2 = &PROPERTY.PropertyInfoCollection;
   /*do scroll 2 processing */
   Else
/* do scroll 1 processing */
   End-If;
End-For;
```

Required

Description

This property returns True if the object executing the property is a required property, False otherwise.

This property is read-only.

Type

Description

This property returns the field type, as a number, of the object. The values are:

Value	Description
0	Character
1	Long Character
2	Number
3	Signed Number

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
4	Date
5	Time
6	DateTime
7	SubRecord (Not supported with Component Interfaces)
8	Image (Limited support with Component Interfaces)
9	ImageReference (Not supported with Component Interfaces)

This property is read-only.

Xlat

Description

This property returns True if the object executing the property is associated with an XLAT table, False otherwise.

This property is read-only.

YesNo

Description

This property returns True if the object executing the property is associated with the Yes/No table, False otherwise.

This property is read-only.

Component Interface Examples

The following are examples of some of the most usual actions you're likely to perform using a Component Interface.

Create a New Instance of Data Example

The following is an example of how to create a new instance of a Component Interface.

To create a new instance of data:

In this example, you are creating a new instance of data for the EXPRESS Component Interface, which is based on the EXPRESS_ISSUE_INV component. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EXPRESS);
&MYCI.BUSINESS_UNIT = "H01B";
&MYCI.INTERNAL_FLG = "Y";
&MYCI.ORDER_NO = "NEXT&rsquo;;
&MYCI.Create();
&MYCI.CUSTOMER = "John&rsquo;s Chicken Shack";
&MYCI.LOCATION = "H10B6987";
.
.
.
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EXPRESS);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the CREATEKEYS.

   These key values are required to open a new instance of the data. If the values you specify aren't unique, that is, if an instance of the data already exists in the database with those key values, you receive a runtime error.

   ```
   &MYCI.BUSINESS_UNIT = "H01B";
   &MYCI.INTERNAL_FLG = "Y";
   &MYCI.ORDER_NO = "NEXT&rsquo;;
   ```

4. Create the instance of data for the Component Interface.

   After you set the key values, you must use the Create method to populate the Component Interface with the key values you set.

   ```
   &MYCI.Create();
   ```

Copyright © 1988-2007, Oracle. All rights reserved.

This creates an instance of this data. However, it hasn't been saved to the database. You must use the Save method before the instance of data is committed to the database.

5. Set the rest of the fields.

   Assign values to the other fields.

```
&MYCI.CUSTOMER = "John&rsquo;s Chicken Shack";
&MYCI.LOCATION = "H10B6987";
.
.
.
```

   If you have specified InteractiveMode as True, every time you assign a value or use the InsertItem or DeleteItem methods, any PeopleCode programs associated with that field (either with record field or the component record field) fires (FieldEdit, FieldChange, RowInsert, and so on.)

6. Save the data.

   When you execute the Save method, the new instance of the data is saved to the database.

```
If NOT(&MYCI.Save()) Then
```

   The Save method returns a Boolean value: True if the save was successful, False otherwise. You can use this value to do error checking.

   The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on.) are executed. If you didn't specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time for all fields that had values set.

Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

---

7. Check Errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

   The Text property of the PSMessage returns the text of the error message. At the end of this text is a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{BusinessComponentName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```

 Copyright © 1988-2007, Oracle. All rights reserved.

For example, if you specified the incorrect format for a date field of the Component Interface named ABS_HIST, the Text property would contain the following string:

```
Invalid Date {ABS_HIST.BEGIN_DT} (90), (1)
```

The contextual string (by itself) is available using the Source property of the PSMessage.

Note. If you've called your Component Interface from an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Chapter 36, "Session Class," Error Handling, page 1512 and Chapter 36, "Session Class," Source, page 1526.

Getting an Existing Instance of Data Example

The following is an example of how to get an existing instance of a Component Interface.

To get an existing instance of data:

In this example, you are getting an existing instance of data for the EMPL_CHKLST_BC Component Interface, which is based on the EMPLOYEE_CHECKLIST component. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_BC);
&MYCI.EMPLID= "8001";
&MYCI.Get();

/* Get checklist Code */

&CHECKLIST_CD = &MYCI.CHECKLIST_CD;

/* Set Effective date */

&MYCI.EFFDT = "05-01-1990";
.
.
.
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

1. Get a session object.

Copyright © 1988-2007, Oracle. All rights reserved.

Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

```
&MYSESSION = %Session;
```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_BC);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the GETKEYS.

   These are the key values required to return a unique instance of existing data. If the keys you specify allow for more than one instance of the data to be returned, or if no instance of the data matching the key values is found, you receive a runtime error.

   ```
   &MYCI.EMPLID = "8001";
   ```

4. Get the instance of data for the Component Interface.

   After you set the key values, you have to use the Get method.

   ```
   &MYCI.Get();
   ```

   This populates the Component Interface with data, based on the key values you set.

5. Get field values or set field values.

   At this point, you can either get values or set values.

   ```
   &CHECKLIST_CD = &MYCI.CHECKLIST_CD;

   /* OR */
   &MYCI.EFFDT = "05-01-1990";
   ```

   If you have specified InteractiveMode as True, every time you assign a value, any PeopleCode programs associated with that field (either with record field or the component record field) fires (FieldEdit, FieldChange, and so on.)

6. Save or Cancel the Component Interface, as appropriate.

   If you've changed values and you want to save your changes to the database, you must use the Save method.

   ```
   If NOT(&MYCI.Save()) Then
   ```

   The Save method returns a Boolean value: True if the save was successful, False otherwise. Use this value to do error checking.

   The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on.) are executed. If you didn't specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

If you don't want to save any changes to the data, use the Cancel method. The Component Interface is reset to the state it was in just after you used the GetCompIntfc method.

```
&MYCI.Cancel();
```

This is similar to a user pressing ESC while in a component, and choosing to not save any of their changes.

7. Check Errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

   The Text property of the PSMessage returns the text of the error message. At the end of this text is a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{ComponentInterfaceName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```

   For example, if you specified the incorrect format for a date field of the Component Interface named ABS_HIST, the Text property would contain the following string:

```
Invalid Date {ABS_HIST.BEGIN_DT} (90), (1)
```

   The contextual string (by itself) is available using the Source property of the PSMessage.

Note. If you've called your Component Interface from an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Chapter 36, "Session Class," Error Handling, page 1512 and Chapter 36, "Session Class," Source, page 1526.

Retrieving a List of Instance of Data Example

The following is an example of how to retrieve a list of instances of data.

To retrieve a list of instances of data:

In this example, you are getting a list of existing instances of data for the EMPL_CHKLST_CI Component Interface, which is based on the EMPLOYEE_CHECKLIST component. The following is the complete code sample: the steps break it up and explain each line.

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;
Local ApiObject &MYNEWCI;


&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_CI);
&MYCI.EMPLID= "8";
&MYCI.LAST_NAME_SRCH = "S";
&MYLIST = &MYCI.Find();
For &I = 1 to &MYLIST.Count;

   /* note: do not reuse the CI used to create the list, or the list will be⇒
 destroyed */

&MYNEWCI = &MYLIST.Item(&I);

/* CI from list still must be instantiated to use it */

&MYNEWCI.Get();

/* do some processing */

End-For;
```

1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.EMPL_CHKLST_CI);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the FINDKEYS values.

   These can be alternate key values or partial key values. If no instance of the data matching the key values is found, you receive a runtime error.

   ```
   &MYCI.EMPLID = "8";
   &MYCI.LAST_NAME_SRCH = "S";
   ```

4. Get a list of data instances for the Component Interface.

   After you set the alternate or partial key values, use the Find method to return a list of data instances for the Component Interface.

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYLIST = &MYCI.Find();
```

Note. The Find method can be executed only at level zero in a Component Interface.

5. Select an instance of the data.

   To select an instance of the data, you can use any of the following standard data collection methods:

   - First
   - Item
   - Next

For example, you could loop through every instance of the data to do some processing:

```
For &I = 1 to &MYLIST.Count;
   /* note: do not reuse the Component Interface used to */
   /* create the list here, or the list will be destroyed */
&MYNEWCI = &MYLIST.Item(&I);
/* CI from list must still be instantiated to use it */
&MYNEWCI.Get();
/* do some processing */
End-For;
```

After you have a specific instance of the data, you can get values, set values, and so on.

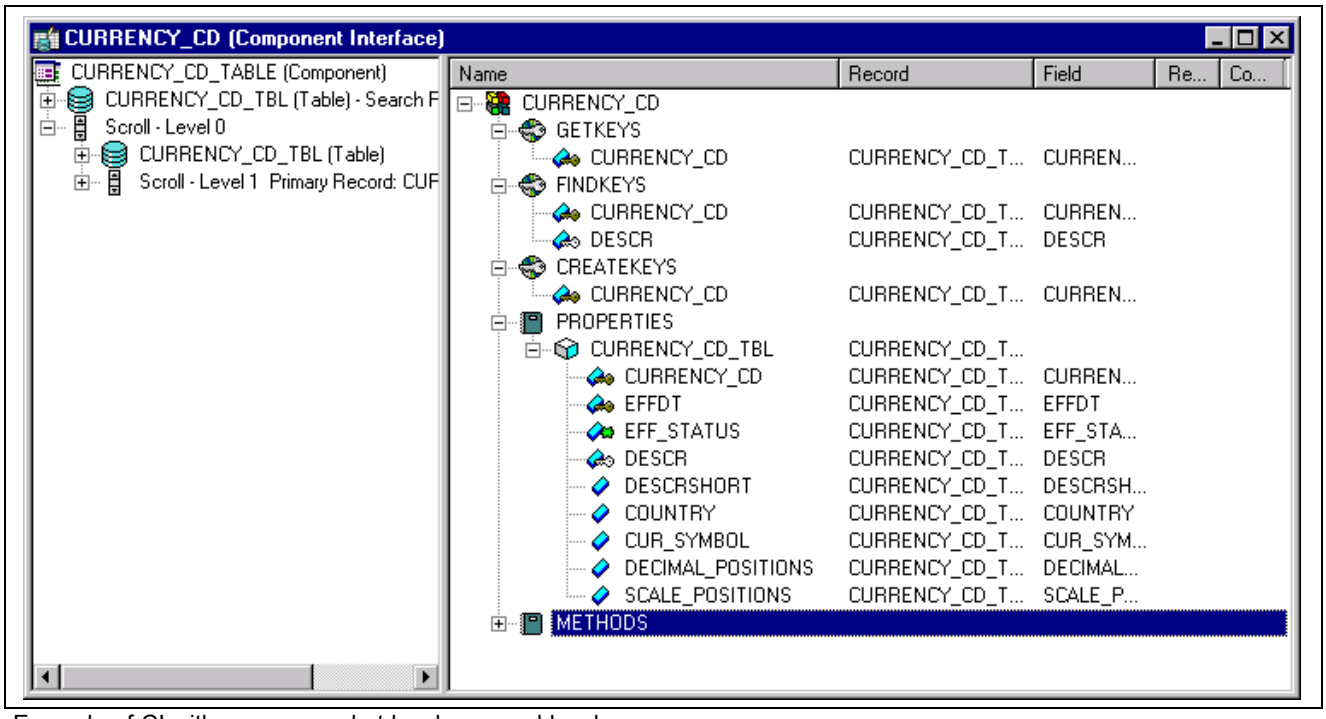
See Chapter 12, “Component Interface Classes,” Data Collection Methods, page 443.

Inserting Effective-Dated Data Example

You can rename a collection in a Component Interface. For example, suppose you had the same record at level zero and at level one. You may want to rename the level one data collection to reflect this. The data in a data collection is associated with the primary database record of a scroll, *not* with the name you supply.

Here is an example of using a Component Interface that has the same record at level zero and level one. The Component Interface is based on the CURRENCY_CD_TBL component.

Copyright © 1988-2007, Oracle. All rights reserved.



Example of CI with same record at level zero and level one

The following code example is based on this Component Interface and does the following:

1. Gets an existing Component Interface.
2. Finds the current effective-dated item index.
3. Inserts a new row before the current effective-dated item using the InsertItem method.

```
Local ApiObject &Session;
Local ApiObject &CURRENCY_CD;
Local ApiObject &CURRENCY_CD_TBLCol;
Local ApiObject &CURRENCY_CD_TBLItm;
Local ApiObject &PSMessages;
Local string &ErrorText, &ErrorType;
Local number &ErrorCount;
Local Boolean &Error;

Function CheckErrorCodes()

   &PSMessages = &Session.PSMessages;
   &ErrorCount = &PSMessages.Count;
   For &i = 1 To &ErrorCount
      &ErrorText = &PSMessages.Item(&i).Text;
      &ErrorType = &PSMessages.Item(&i).Type;
   End-For;

End-Function;

/* Initialize Flags */
&Error = False;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&Session = %Session;

If &Session <> Null Then

   CheckErrorCodes();
   /* Application Specific Error Processing */

Else

   &CURRENCY_CD = &Session.GetCompIntfc(CompIntfc.CURRENCY_CD);

   If &CURRENCY_CD = Null Then

      CheckErrorCodes();
      /* Application Specific Error Processing */

   Else

      /* Set Component Interface Get Keys */
      &CURRENCY_CD.CURRENCY_CD = "USD";

      If Not &CURRENCY_CD.Get() Then

         CheckErrorCodes();
         /* Application Specific Error Processing */
         &Error = True;

      End-If;

      If Not &Error Then

         &CURRENCY_CD_TBLCol = &CURRENCY_CD.CURRENCY_CD_TBL;
         &CURRENCY_CD_TBLItm = &CURRENCY_CD_TBLCol.InsertItem(&CURRENCY_CD_⇒
TBLCol.CurrentItemNum());
         &CURRENCY_CD_TBLItm.EFFDT = %Date;
         &CURRENCY_CD_TBLItm.EFF_STATUS = "A";
         &CURRENCY_CD_TBLItm.DESCR = "NewCurrencyCode";
         &CURRENCY_CD_TBLItm.DESCRSHORT = "New";
         &CURRENCY_CD_TBLItm.COUNTRY = "USA";
         &CURRENCY_CD_TBLItm.CUR_SYMBOL = "?";
         &CURRENCY_CD_TBLItm.DECIMAL_POSITIONS = 4;
         &CURRENCY_CD_TBLItm.SCALE_POSITIONS = 0;

         /* Save Instance of Component Interface */
         If Not &CURRENCY_CD.Save() Then

            CheckErrorCodes();
            /* Application Specific Error Processing */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
            End-If;

         End-If;
         /* End: Set Component Interface Properties */

         /* Cancel Instance of Component Interface */
         &CURRENCY_CD.Cancel();

      End-If;

End-If;
```

Inserting Effective-Dated Data Example Using Visual Basic

Here's a code example in Visual Basic that does the same thing as the previous code example.

```
Private Sub CURRENCY_CD()

        On Error GoTo eMessage

        '***** Set Object References *****
        Dim oCISession As Object
        Dim oCURRENCY_CD As Object
        Dim oCURRENCY_CD_TBL As Object
        Dim oCURRENCY_CD_TBLItem As Object

        '***** Set Connect Parameters *****
        strAppSeverPath = "//PSOFT101999:9000"
        strOperatorID = "PTDMO"
        strPassword = "PTDMO"

        '***** Create PeopleSoft Session Object *****
        Set oCISession = CreateObject("PeopleSoft.Session")

        '***** Connect to the App Sever *****
        oCISession.Connect 1, strAppSeverPath, strOperatorID, strPassword, 0

        '***** Get the Component *****
        Set oCURRENCY_CD = oCISession.GetCompIntfc("CURRENCY_CD")

        '***** Set the Component Interface Mode *****
        oCURRENCY_CD.InteractiveMode = False
        oCURRENCY_CD.GetHistoryItems = True

        '***** Set Component Get/Create Keys *****
        oCURRENCY_CD.CURRENCY_CD = "USD"

        '***** Execute Create *****
        oCURRENCY_CD.Get
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
        'Set CURRENCY_CD_TBL Collection Field Properties --
        'Parent: PS_ROOT Collection
        Set oCURRENCY_CD_TBL = oCURRENCY_CD.CURRENCY_CD_TBL
        Set oCURRENCY_CD_TBLItem = oCURRENCY_CD_TBL.InsertItem(oCURRENCY_CD_⇒
TBL.CurrentItemNum())

        oCURRENCY_CD_TBLItem.EFFDT = Date
        oCURRENCY_CD_TBLItem.EFF_STATUS = "A"
        oCURRENCY_CD_TBLItem.DESCR = "NewCurrencyCode"
        oCURRENCY_CD_TBLItem.DESCRSHORT = "New"
        oCURRENCY_CD_TBLItem.COUNTRY = "USA"
        oCURRENCY_CD_TBLItem.CUR_SYMBOL = "?"
        oCURRENCY_CD_TBLItem.DECIMAL_POSITIONS = 4
        oCURRENCY_CD_TBLItem.SCALE_POSITIONS = 0

        '***** Save Component Interface *****
        oCURRENCY_CD.Save
        oCURRENCY_CD.Cancel

        Exit Sub

eMessage:
        '***** Display VB Runtime Errors *****
        MsgBox Err.Description

        '***** Display PeopleSoft Error Messages *****
        If oCISession.PSMessages.Count > 0 Then
                For i = 1 To oCISession.PSMessages.Count
                        MsgBox oCISession.PSMessages.Item(i).Text
                Next i
        End If

End Sub

Sub MAIN()
    CURRENCY_CD
End Sub
```

Inserting or Deleting a Row of Data Example

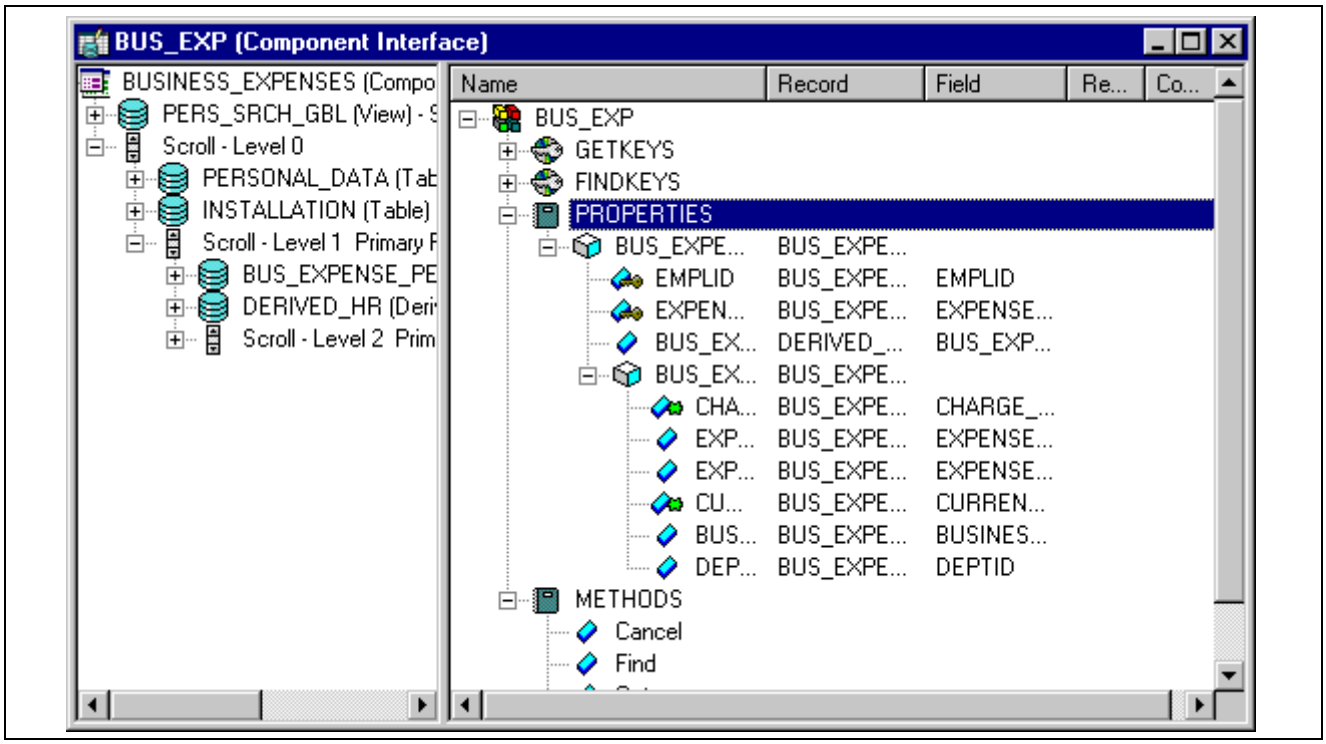
The CopyRowset and CopyRowsetDelta methods use the primary database name of a scroll, not the name you may give a collection.

A data collection represents a row of data. You often insert or delete a row of data.

To insert or delete a row of data:

In this example, you are getting a existing instance of data for the BUS_EXP Component Interface, which is based on the BUSINESS_EXPENSES component, then inserting (or deleting) a row of data in the second level scroll.

Copyright © 1988-2007, Oracle. All rights reserved.



BUS_EXP Component Interface definition

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MYSESSION;
Local ApiObject &MYCI;

&MYSESSION = %Session;
&MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.BUS_EXP);
&MYCI.EMPLID= "8001";
&MYCI.Get();
&MYLEVEL1 = &MYCI.BUS_EXPENSE_PER;
/* get appropriate item in lvl1 collection */
For &I = 1 to &MYLEVEL1.Count
&ITEM = &MYLEVEL1.Item(&I);
&MYLEVEL2 = &ITEM.BUS_EXPENSE_DTL;
&COUNT = &MYLEVEL2.Count
/* get appropriate item in lvl2 collection */
For &J = 1 to &COUNT
&LVL2ITEM = &MYLEVEL2.Item(&J);
&CIDATE = &LVL2ITEM.CHARGE_DT;
If &CIDATE <> %Date Then
      /* insert row */
&NEWITEM = &MYLEVEL2.InsertItem(&COUNT);
   /* set values for &NEWITEM */
Else
      /* delete last row */
   &MYLEVEL2.DeleteItem(&COUNT);
End-If;
   End-For;
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
End-For;

If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

1. Get a session object.

   Before you can get a Component Interface, you have to get a session object. The session controls access to the Component Interface, provides error tracing, enables you to set the runtime environment, and so on.

   ```
   &MYSESSION = %Session;
   ```

2. Get a Component Interface.

   Use the GetCompIntfc method with a session object to get the Component Interface. You must specify a Component Interface definition that has already been created. You receive a runtime error if you specify a Component Interface that doesn't exist.

   ```
   &MYCI = &MYSESSION.GetCompIntfc(COMPINTFC.BUS_EXP);
   ```

   After you execute the GetCompIntfc method, you have only the *structure* of the Component Interface. You haven't populated the Component Interface with data yet.

3. Set the GETKEYS.

   These are the key values required to return a unique instance of existing data. If the keys you specify allow for more than one instance of the data to be returned, or if no instance of the data matching the key values is found, you receive a runtime error.

   ```
   &MYCI.EMPLID = "8001";
   ```

4. Get the instance of data for the Component Interface.

   After you set the key values, you must use the Get method.

   ```
   &MYCI.Get();
   ```

   This populates the Component Interface with data, based on the key values you set.

5. Get the level one scroll.

   The name of the scroll can be treated like a property. It returns a data collection that contains all the level one data.

   ```
   &MYLEVEL1 = &MYCI.BUS_EXPENSE_PER
   ```

6. Get the appropriate item in the level one data collection.

   Remember, scroll data is hierarchical. You must get the appropriate level one row before you can access the level two data.

Copyright © 1988-2007, Oracle. All rights reserved.

```
For &I = 1 to &MYLEVEL1.Count
&ITEM = &MYLEVEL1.Item(&I);
```

7. Get the level two scroll.

   This is done the same way as you accessed the level one scroll, by using the scroll name as a property.

```
&MYLEVEL2 = &ITEM.BUS_EXPENSE_DTL;
```

8. Get the appropriate item in the level two data collection.

   A data collection is made up of a series of items (rows of data.) You have to access the appropriate item (row) in this level also.

```
&COUNT = &MYLEVEL2.Count
/* get appropriate item in lvl2 collection */
For &J = 1 to &COUNT
&LVL2ITEM = &MYLEVEL2.Item(&J);
```

9. Insert or delete a row of data.

   You can insert or delete a row of data from a data collection. The following example finds the last item (row of data) in the second level scroll. If it matches the value of %Date, the last item is deleted. If it doesn't match, a new row is inserted.

```
&CIDATE = &LVL2ITEM.CHARGE_DT;
If &CIDATE <> %Date Then
   /* insert row */
&NEWITEM = &MYLEVEL2.InsertItem(&COUNT);
/* set values for &NEWITEM */
Else
   /* delete last row */
&MYLEVEL2.DeleteItem(&COUNT);
End-If;
```

10. Save the data.

    When you execute the Save method, the new instance of the data is saved to the database.

```
If NOT(&MYCI.Save()) Then
```

    The Save method returns a Boolean value: True if the save was successful, False otherwise. Use this value to do error checking.

    The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with SaveEdit, SavePreChange, WorkFlow, and so on) are executed. If you did not specify the Component Interface to run in interactive mode, FieldEdit, FieldChange, and so on, also run at this time.

Note. If you're running a Component Interface from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

---

11. Check Errors.

    You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MYCI.Save()) Then
   /* save didn&rsquo;t complete */
   &COLL = &SESSION.PSMessages;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

The Text property of the PSMessage returns the text of the error message. At the end of this text is a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{BusinessComponentName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```

For example, if you specified the incorrect format for a date field of the Component Interface named ABS_HIST, the Text property contains the following string:

```
Invalid Date {ABS_HIST.BEGIN_DT} (90), (1)
```

The contextual string (by itself) is available using the Source property of the PSMessage.

Note. If you've called the Component Interface from an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Chapter 36, "Session Class," Error Handling, page 1512 and Chapter 36, "Session Class," Source, page 1526.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 13

Crypt Class

This chapter provides an overview of the crypt class and discusses:

- Create a crypt object
- Error handling
- Data type of a crypt object
- Scope of a crypt object
- Crypt class reference

Understanding the Crypt Class

The crypt class is used with pluggable cryptography. After you create an encryption profile, use PeopleCode to invoke the encryption profile for encrypting, decrypting, or signing a field, depending on the profile

See Also

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, "Securing Data with Pluggable Cryptography"

Creating a Crypt Object

The crypt class does not have a separate function for instantiating an object (such as CreateCrypt.) Instead, you instantiate a crypt object using the CreateObject function, using the keyword **Crypt**.

```
&cry = CreateObject("Crypt");
```

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CreateObject.

Declaring Crypt Objects

Crypt objects are declared by using the Crypt type name.

```
Local Crypt &MyCrypt;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Note. Crypt objects cannot be serialized, and so can only be declared as Local.

Scope of a Crypt Object

A crypt object can only be instantiated from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

Crypt Class Methods

In this section, we discuss the crypt class methods. The methods are discussed in alphabetical order.

FirstStep

Syntax

```
FirstStep()
```

Description

Use the FirstStep method to access the first step in the encryption chain.

You must use either the FirstStep or GoToStep methods before you use the NextStep or SetParameter methods.

Parameters

None.

Returns

None.

See Also

Chapter 13, “Crypt Class,” GoToStep, page 476 , Chapter 13, “Crypt Class,” NextStep, page 477 and Chapter 13, “Crypt Class,” SetParameter, page 478.

GoToStep

Syntax

```
GoToStep(StepNum)
```

Description

Use the GoToStep method to access a specific step in the encryption chain.

You must use either the GoToStep or FirstStep methods before you use the NextStep or SetParameter method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
StepNum	Specify the step number that you want to access, as a number.

Returns

None.

See Also

Chapter 13, "Crypt Class," FirstStep, page 476 , Chapter 13, "Crypt Class," NextStep, page 477 and Chapter 13, "Crypt Class," SetParameter, page 478.

LoadLibrary

Syntax

LoadLibrary(*LibraryFile, LibraryID*)

Description

Use the LoadLibrary method to specify the encryption library to be used. This method is generally used when either your underlying library changes (such as, a new version, added algorithms, and so on) or you have your written own library and you need to load the metadata into the PeopleSoft system.

Parameters

Parameter	Description
LibraryFile	Specify the name of the file containing the encryption library as a string. You do not have to specify a full path name. The delivered OpenSSL library is pspetssl.dll. The delivered PGP library is pspetpgp.dll.
LibraryID	Specify the name of the library, as a string.

Returns

None.

NextStep

Syntax

NextStep()

Description

Use the NextStep method to access the next step in the encryption chain.

You must use the FirstStep or GoToStep method before using NextStep.

Your program terminates if you call NextStep when you are already at the last step in the encryption chain.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 13, "Crypt Class," FirstStep, page 476 and Chapter 13, "Crypt Class," GoToStep, page 476.

Open

Syntax

Open(*ProfileName*)

Description

Use the Open method to open the encryption profile identified by *ProfileName*. You must open an encryption profile before you can add data to the encryption profile. Your program terminates if you specify an encryption profile that doesn't exist.

Parameters

Parameter	Description
ProfileName	Specify the name of the encryption profile you want to access, as a string. You can store the name of the encryption profile in a field, and specify a *fieldname.recordname*.

Returns

None.

Example

```
Local Crypt &cry;

&cry = CreateObject("Crypt");
&bar = QE_CRYPT_WRK.CRYPT_PRFL_ID;
&cry.Open(&bar);
&cry.UpdateData(QE_CRYPT_WRK.DESCRLONG);
QE_CRYPT_WRK.LARGECHAR = &cry.Result;
```

SetParameter

Syntax

SetParameter(*Name*, *Value*)

Description

Use the SetParameter method to set the parameter specified by *Name* to a value specified by *Value.*

You must have already used the FirstStep, NextStep, or GoToStep methods to specify a step before using this method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of the parameter that you want to change, as a string.
Value	Specify the value for the parameter that you want to change.

Returns

None.

See Also

Chapter 13, “Crypt Class,” FirstStep, page 476 , Chapter 13, “Crypt Class,” GoToStep, page 476 and Chapter 13, “Crypt Class,” NextStep, page 477.

UpdateData

Syntax

`UpdateData``(`*`Data`*`)`

Description

Use the UpdateData method to add data to the encryption chain. This method can be called multiple times after opening a profile to add data.

Parameters

Parameter	Description
Data	Specify the data you want to add to the encryption chain, as a string.

Returns

None.

Crypt Class Properties

This section describes the crypt class properties. The properties are described in alphabetical order.

Result

Description

After updating the encryption chain, the Result property contains the end result of the encryption chain. Once the result has been retrieved, it is no longer possible to update the object anymore.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Verified

Description

For algorithms that check a signature, the system sets the Verified property to true if the signature is valid, and false if the signature is invalid. For algorithms that do not check a signature, it always returns false.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 14

Exception Class

This chapter provides an overview of Exception class and discusses the following topics:

- How exceptions are thrown.
- When to use exceptions.
- Try-Catch sections.
- Data Type for Exception class objects.
- Scope of Exception class objects.
- Exception class built-in function and language constructs.
- Exception class methods.
- Exception class properties.

Understanding Exception Class

An *exception* can be defined as any unusual event that requires special handling in your PeopleCode program. This could include some kind of hardware failure, such as trying to write to a printer that's been turned off, or a software failure, such as trying to divide by zero. You can also detect your own errors, and use the Throw statement to construct an exception.

Exception handling is the processing you initiate when an exception occurs. You can handle errors in PeopleCode using the Catch statement. The places in your code where you want exceptions handled must be enclosed by the Try and End-Try statements.

The Exception class encapsulates exceptional runtime conditions. It can be used with most PeopleCode programs.

Note. The Exception class does *not* work with any of the PeopleSoft APIs, that is, the classes whose objects are declared as type ApiObject. This includes the Tree classes, the Query classes, Search classes, and so on.

The process of error handling can be broken down as follows:

- An error occurs (either a hardware or software error).
- The error is detected and an exception is thrown (either by the system or by your program).
- Your exception handler provides the response.

Copyright © 1988-2007, Oracle. All rights reserved.

How Exceptions Are Thrown

Exceptions get thrown either by the runtime system or by your program.

In general, the kinds of errors that cause the runtime system to throw an exception are errors that terminate your PeopleCode program. For example, dividing by zero, referencing a method or property that doesn't exist, or trying to use a method or property on a null object.

To throw your own exceptions, you can use the Throw statement.

When to Use Exceptions

If the error is something that you can easily check, you shouldn't use exceptions. For example, suppose your page had begin and end date fields. In your SavePreChange PeopleCode program, you would want to check to verify that the end date was after the begin date. This kind of error produces incorrect data. It doesn't terminate your PeopleCode program. This is a simple check, one that you don't need to throw an exception for.

Errors that you might want to use exceptions for are the kinds that you are going to check for often. It might make more sense to have a single exception class for that kind of error, so you'd only have to write your error handling code once.

For example, suppose that a function you called was supposed to return an array object that you'd then manipulate using the array class methods and properties. When the function fails and doesn't return an array object, you can either check for Null, or check for an exception (as this kind of error terminates your PeopleCode program.) If you must often check for Nulls, perhaps instead of constantly checking, you could rely on the runtime system throwing exceptions for you.

Pseudo code example:

```
Import ExceptionPackage:*;
/* this package contains all the exception classes for your app */

try
   &MyArray = GetArrayData(&Param1, &Param2, &Param3);
   /* Code to manipulate &MyArray here */
catch ExceptionNull &Ex1
   if &Ex1.MessageSetNumber = 2
      and &Ex1.MessageNumber = 236 Then

/* this is the message set and message number for &MyArray being Null */

   /* do data error handling */

   end-if;
end-try;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Considerations Using Exceptions and Transfer Functions

Using functions that transfer the end user to a new component, such as the DoModal, DoModalComponent, or Transfer functions (in some cases) inside a try block do *not* catch PeopleCode exceptions thrown in the new component. Starting a new component starts a brand new PeopleCode evaluation context. Catches are only caught for exceptions thrown within the *current* component.

In the following code example, the catch statement only catches exceptions thrown in the code *prior to* using the DoModal function, but not any exceptions that are thrown within the new component.

```
/* Set up transaction */
If %CompIntfcName = "" Then
   try
      &oTrans = &g_ERMS_TransactionCollection.GetTransactionByName(RB_EM_⇒
WRK.DESCR);
      &sSearchPage = &oTrans.SearchPage;
      &sSearchRecord = &oTrans.SearchRecord;
      &sSearchTitle = &oTrans.GetSearchPageTitle();
      If Not All(&sSearchPage, &sSearchRecord, &sSearchTitle) Then
         Error (MsgGetText(17834, 8081, "Message Not Found"));
      End-If;
      &c_ERMS_SearchTransaction = &oTrans;

      /* Attempt to transfer to hidden search page with configurable filter */
      &nModalReturn = DoModal(@("Page." | &sSearchPage), &sSearchTitle, - 1, - 1);
   catch Exception &e
      Error (MsgGetText(17834, 8082, "Message Not Found"));
   end-try;
```

Try-Catch Sections

The statements that immediately follow the Try statement are called the *protected statements*. These are the only statements that are 'protected' by the catch clauses in the Try-Catch statements. The catch clauses in a Try-Catch statement can be executed only if an exception is thrown by the protected statements. In addition, a catch clause is executed only when handling an exception that matches the type given on the catch. Any exceptions thrown by the catch clauses are not caught by their own Try-Catch statement.

The execution of the Try-Catch statement starts by executing the protected statements. If none of these statements, as well as none of the statements called by them, causes an exception to be thrown, the try-catch is done. The statements in the catch clauses are not executed.

Each catch clause in the Try-Catch section declares a local variable of class Exception or a subclass of Exception. A catch clause local variable has the same scope and lifetime as a local variable declared at the start of the catch clause.

The following is the general order of exception execution and handling in the Try-Catch statements.

1. The exception is thrown.
2. The exception is considered by the first enclosing Try-Catch statement.

   Try-catch statements can be nested.

Copyright © 1988-2007, Oracle. All rights reserved.

This is a dynamic enclosure, that is, tracking back through method and function callers. Any intervening non-Try-Catch statements are skipped.

3. Within this Try-Catch statement, the catch clauses are considered in program text order, that is, first the first catch clause is considered, then the next, and so on to the last. At each consideration, the type (class) of the exception being thrown is compared to the type of the local variable declared by the Catch clause. If the exception being thrown is the same as the Catch type, or is a subtype of it (so the exception could be assigned to the Catch's local variable) the Catch clause matches the exception being thrown.
4. If a matching catch clause is found, that catch clause handles the exception. The exception is considered 'caught', which means that it does not propagate further. Execution proceeds by assigning the thrown object to the catch clause's local variable and executing the statements in the catch clause. When one of the catch clauses handles the exception, (and the statements in the catch clause don't throw any further exceptions), execution continues normally after the Try-Catch statement.

   If a statement in the catch clause throws another exception, that exception begins to propagate from the point of the throw.
5. If a matching catch clause is not found, that is, if no catch clauses can accept the thrown object, then the thrown object is still uncaught. The thrown object is considered by the next dynamically enclosing Try-Catch statement (return to step 3).
6. If a thrown object is not caught by any catch clause in any dynamically enclosing Try-Catch statements, a fatal PeopleCode evaluation error is produced.

Data Type for Exception Class Objects

Exception Class objects are declared as type Exception. For example:

```
Local Exception &Ex1;
```

Scope of Exception Class Objects

Exception objects can be instantiated only from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

Exception Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateException

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Throw

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Try

 Copyright © 1988-2007, Oracle. All rights reserved.

Exception Class Methods

In this section, we discuss the Exception class methods, in alphabetical order.

GetSubstitution

Syntax

GetSubstitution(*Index*)

Description

When you create a message in the message catalog, you can specify values in the message text, such as %1, %2, and so on, that are replaced with values by the system at runtime. The value that gets put into the message text value is called the *substitution string*. The GetSubstitution method returns the substitution string in the error message specified by *Index*, with 1 being the first substitution string, 2 being the second, and so on.

Parameters

Parameter	Description
Index	Specify the substitution string number that you'd like to return. The first one is 1, the second is 2, and so on. You must specify a valid substitution string number, that is, you can't specify a 4 if there are only three substitution strings.

Returns

The substitution text as a string.

Example

```
If &Ex1.MessageSetNumber = 2 and &Ex1.MessageNumber = 170 Then
   /* Get Sendmail error */
      &String = &Ex1.GetSubstitution(1);
   /* do processing according to type of Sendmail error */
   . . . .
End-if;
```

See Also

Chapter 14, "Exception Class," SetSubstitution, page 486

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Message Catalog

Output

Syntax

Output()

Description

Use the Output method if you want to display the message associated with the exception to the user (in an online context), or have it logged (if offline). This is analogous to using the MessageBox function to log a message.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

If called in an online context, display the message associated with the exception to the user. If called offline, record the exception for later display or analysis.

Example

```
catch Except2 &Ex2
   &Ex2.Output(); /* tell about it */
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Throw

SetSubstitution

Syntax

SetSubstitution(*Index*, *String*)

Description

When you create a message in the message catalog, you can specify values in the message text, such as %1, %2, and so on, that are replaced with values by the system at runtime. The value that gets put into the message text value is called the *substitution string*. The SetSubstitution method sets the substitution string in the error message specified by *Index*, with 1 being the first substitution string, 2 being the second, and so on.

Parameters

Parameter	Description
Index	Specify the substitution string number that you'd like to replace. The first one is 1, the second is 2, and so on. You must specify a valid substitution string number, that is, you can't specify a 4 if there are only three substitution strings.
String	Specify the text of the substitution string.

Returns

None.

See Also

Chapter 14, "Exception Class," GetSubstitution, page 485

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Message Catalog

ToString

Syntax

ToString([*AddContext*])

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The ToString method returns the expanded message text associated with the exception in the current user's current language. The string returned is the message text (in the current user's language) with the substitution string replacing the substitution markers (%1 and so on) in the text. It always returns the context information about where the error occurred at the end of the message. To suppress this, specify *AddContext* as false.

Parameters

Parameter	Description
AddContext	Specify whether to have the context information about where the error occurred added at the end of the message. This parameter takes a Boolean value. The default is true, that is, to add the context information.

Returns

A string.

See Also

Chapter 14, "Exception Class," Output, page 485

Exception Class Properties

In this section, we discuss the Exception class properties, in alphabetical order.

Context

Description

This property returns a string description of the location of the condition in PeopleCode or other processing. This might contain a stack backtrace at the point of the exceptional condition.

This property is read-only.

Example

This is the "At" part of the error message display. For example, for an exception thrown in Record.Field Event QE_31DIGREC1.QE_31DIGFLD5 FieldChange function EachComp at PeopleCode program counter 671, statement 11 of the source program, the Context would be:

```
At QE_31DIGREC1.QE_31DIGFLD5.FieldChange EachComp   PCPC:671  Statement:11
```

DefaultText

Description

This property sets a string to use as the basic message text for the exception when the message can't be found in the message catalog.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

MessageNumber

Description

This property is the message number for a message describing the exceptional condition.

This property is read-only.

MessageSetNumber

Description

This property is the message set number for a message describing the exceptional condition.

This property is read-only.

MessageSeverity

Description

This property is the message severity as a string for a message describing the exceptional condition. The message severity is set for the message when it's created in the message catalog. The values are:

Numeric Value	Constant Value	Description
C	%MsgSeverity_Cancel	Message severity is Cancel.
E	%MsgSeverity_Error	Message severity is Error.
M	%MsgSeverity_Message	Message severity is Message.
W	%MsgSeverity_Warning	Message severity is Warning.

This property is read-only.

StackTrace

Description

This property returns a string with the complete stack trace. This includes the whole context, (that is, current location + called from X + called from. . .)

You can process the returned string by breaking it into pieces using the Split function, and using "Called from" as the value to use for separating the string. For example,

```
Local Array of String &Pieces;

/* get exception */

&Pieces = Split(&E.StackTrace, "Called from:");
```

This property is read-only.

Example

The following is an example of a trace stack.

```
In C (0,0) AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Name:C  PCPC
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
:136  Statement:2
Called from:AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Name:B  Sta
tement:6
Called from:AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Name:A  Sta
tement:13
Called from:AEMINITEST.MAIN.GBL.default.1900-01-01.Step02.OnExecute  Statement:1
8
```

SubstitutionCount

Description

This property returns the number of substitution strings in the message text as a number.

The number of substitutions comes from the *subslist* of the CreateException function.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateException

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 15

Field Class

This chapter provides an overview of Field class and discusses the following topics:

- Shortcut considerations
- Data Type for a field object
- Scope of a field object
- Field class built-in function
- Field class methods
- Field class properties

Understanding the Field Class

A *field* object, instantiated from the field class, is a single instance of data within a record object, and is based on a field definition.

Accessing or changing the value of a field using the Value property is a common action.

SetDefault is a frequently used method. Name, Enabled, and Type are several commonly used field properties. If you are working with message objects, EditError is also a commonly used field property.

The field class is one of the data buffer access classes.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Considerations Using User Interface Properties

All of the field properties related to the user interface, such as the Label property, only return the value previously set in PeopleCode (false, or blank if nothing was set). They do not return the values set for the field in PeopleSoft Application Designer.

Shortcut Considerations

An expression of the form

```
FIELD.fieldname.property
```

Copyright © 1988-2007, Oracle. All rights reserved.

or

```
FIELD.fieldname.method(. . .)
```

is converted to an object expression by using `GetField(`**`FIELD.`***`fieldname`*`)`. For example, the next two lines of code are identical:

```
FIELD.CHECKLIST_DT.Enabled = False;
GetField(FIELD.CHECKLIST_DT).Enabled = False;
```

An expression of the form

```
recname.fieldname.property
```

or

```
recname.fieldname.method(.  .  .)
```

is converted to an object expression by using `GetField(`*`recname.fieldname`*`)`. For example, the next two lines of code are identical:

```
EMPL_CHECKLINST.CHECKLIST_DT.Enabled = False;
GetField(EMPL_CHECKLIST.CHECKLIST_DT).Enabled = False;
```

Data Type for a Field Object

Field objects are declared as type Field. For example,

```
Local Field &MyField;
```

Scope of a Field Object

A field can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Field Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetField

Field Class Methods

In this section, we discuss the Field class methods. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

AddDropDownItem

Syntax

AddDropDownItem(*CodeString*, *DescriptionString*)

Description

The AddDropDownItem method adds an item to the dropdown list in the control for the field. The first time this method is called, it overrides the prompt table or translate table used to populate the list. Those items no longer appear in the list. Only the items added using this method display.

Subsequent calls to this method adds additional items to the dropdown list. The items added with the first call to the method also display.

If there is an existing value and the dropdown list is changed with these functions, the selection shows as (Invalid value) unless the new list contains an entry with the same code as the existing value.

Considerations Using AddDropDownItem

If the data for the dropdown is language sensitive, the values for the dropdown should come from the message catalog or from a database field that has a related language record, and should not be hard-coded.

A good place for your PeopleCode program to populate a dropdown list is in the RowInit event. This event executes before the page is shown for the first time, so it prevents unnecessary SQL.

Parameters

Parameter	Description
CodeString	Specify the value used to set the field value if this item is selected. Codes longer than the size of the field are truncated.
DescriptionString	Specify the value the end-user sees in the dropdown list.

Returns

None.

Example

Using a hardcoded list is not appropriate for this function because translations do not work. The data must come from the Translate Table (or other record) directly so that the data is translated correctly.

```
Local Rowset &Xlat;

&FLD = GetRecord(Record.JOB).GetField(Field.ACTION);
&FLD.ClearDropDownList();

Evaluate %Component
When Component.JOB_DATA_CONCUR
   &Xlat = CreateRowset(Record.PSXLATITEM);
   &Xlat.Fill("WHERE FILL.FIELDNAME = 'ACTION' AND Fill.FIELDVALUE in ⇒
('ADL','HIR') and EFFDT = (select max(EFFDT) from PSXLATITEM B where B.FIELDNAME =⇒
 'ACTION' and B.FIELDVALUE in ('ADL','HIR') and EFFDT <= JOB.EFFDT)");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      &Xlat_cnt = &Xlat.ActiveRowCount;
      For &I = 1 To &Xlat_cnt
         &CodeIn = &Xlat.GetRow(&I).GetRecord(1).FIELDVALUE.Value;
         &DescIn = &Xlat.GetRow(&I).GetRecord(1).XLATLONGNAME.Value;

         &FLD.AddDropDownItem(&CodeIn, &DescIn);
      End-For;

      Break;
   When-Other
   End-Evaluate;
```

See Also

Chapter 15, "Field Class," ClearDropDownList, page 494

ClearDropDownList

Syntax

ClearDropDownList()

Description

The ClearDropDownList method clears all items added to the dropdown list using the AddDropDownItem method. In addition, this method causes the prompt table or translate table values defined for the list to come back into effect again (unless they're subsequently overridden again with AddDropDownItem.)

Parameters

None.

Returns

None.

Example

```
&FLD = GetRecord(Record.ABSENCE_HIST).GetField(Field.ABSENCE_TYPE);
&FLD.AddDropDownItem("CNF", "Conference");
&FLD.AddDropDownItem("VAC", "Vacation");
&FLD.AddDropDownItem("SCK", "Sick");
. . . .
&FLD.ClearDropDownList();
```

See Also

Chapter 15, "Field Class," AddDropDownItem, page 493

GetAuxFlag

Syntax

GetAuxFlag(*FlagNumber*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetAuxFlag method to determine whether the Auxiliary Flag Mask specified by *FlagNumber* has been set for a field.

Currently, only one flag comes preset from PeopleSoft: a 1 indicates a ChartField.

Parameters

Parameter	Description
FlagNumber	Specify the flag number. 1 is a ChartField. Valid values are between 1 and 32.

Returns

A Boolean value: True, the flag is set, False if the flag hasn't been set.

Example

```
&field = GetField(Field.RC_TEST_PB);
&ret = &field.GetAuxFlag(1);
If (&ret = True) Then
   MessageBox(0, "Metadata Fn Status", 0, 0, "Aux Flag 1 is SET - ChartField");
Else
   MessageBox(0, "Metadata Fn Status", 0, 0, "Aux Flag 1 is NOT SET - Not a Chart⇒
Field");
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetDBFieldAuxFlag

GetLongLabel

Syntax

GetLongLabel(*LabelID*)

Description

The GetLongLabel method returns the long name for a field given a label ID. If the given label ID isn't found, a null string is returned. *LabelID* takes a string value.

Note. If a button is defined as an HTML button or hyperlink, and if it has an associated record field, the label associated with it is only the text of the button. The mouse-over text can only be changed by the lable if the button is defined as a button with an Image label.

Returns

A text string containing the long name of the field for the specified label ID.

Example

The following code sets the label for a field to one of two different texts, based on the page.

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local Field &FIELD;

&FIELD = GetField(RECORD.MYFIELD);
   If %Page = PAGE.PAGE1 Then
      &FIELD.Label = &FIELD.GetLongLabel("LABEL1");
   Else If %Page = PAGE.PAGE2 Then
      &FIELD.Label = &FIELD.GetLongLabel("LABEL2");
End-If;
```

If the Label ID is the same as the name of the field, you could use the following:

```
&LABELID = &FIELD.Name;
&FIELD.Label = &FIELD.GetLongLabel(&LABELID);
```

See Also

Chapter 15, "Field Class," GetShortLabel, page 497, Chapter 15, "Field Class," Label, page 510 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," SetLabel.

GetRelated

Syntax

GetRelated(*recname.fieldname*)

Description

The GetRelated method returns a field object for a related field *recname.fieldname* that has the field executing the method as its control field.

This method is similar to the GetRelField built-in function, however, the built-in works only in the current context, while this method can be applied to a field from any position in the buffer structure.

Using GetRelated With a Control Field

PeopleCode events on the Control Field can be triggered by the Related Edit field. When this happens, there can be different behavior than with other types of fields:

- If the events are called from FieldEdit of the Control Field, and that FieldEdit is triggered by a change in the Related Edit field, the functions return the previous value.
- If the events are called from FieldChange of the Control Field, and that FieldChange is triggered by a change in the Related Edit field, the functions return the value entered into the Related Edit. This may be a partial value, that will subsequently be expanded to a complete value when the processing is complete.

Parameters

Parameter	Description
recname.fieldname	Specifies a field in the same row as the current field object, that has the field executing the method as its control field.

Returns

The field object for the field with the specified name that is related to the field executing the method.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

In the following example, the field object is instantiated, then the related display field object. The Value property for the related display is changed, it is disabled, and another variable is assigned its value.

```
Local Field &FIELD, &REL_FIELD;

&FIELD = GetField(OPC_9A2FIELDS.COMPANY);
/* control field object  */
&REL_FIELD = &FIELD.GetRelated(COMPANY_TBL.DESCR);
/* related display object */
&REL_FIELD.Value = "Change";
&REL_FIELD.Enabled = False;
&TMP = &REL_FIELD.Name;
&REL_FIELD.Value = &TMP;
```

If you were not going to use the &FIELD variable later, the first two lines of code in the previous example could be combined:

```
&REL_FIELD = GetField(OPC_9A2FIELDS.COMPANY).GetRelated(COMPANY_TBL.DESCR);
```

Suppose you had two control fields, EMPLID and MANAGER_ID, and both use the NAME field on the PERSONAL_DATA table as their related display. If you need to access both related display fields, you could do the following:

```
&NAME_EMPLID = GetField(PERSONAL_DATA.EMPLID).GetRelated(PERSONAL_DATA.NAME);
&NAME_MANAGER = GetField(PERSONAL_DATA.MANAGER_ID).GetRelated(PERSONAL_DATA.NAME);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetField

GetShortLabel

Syntax

GetShortLabel(*LabelID*)

Description

The GetShortLabel method returns the short name for a field given a label ID. If the given label ID isn't found, a null string is returned. *LabelID* takes a string value.

Note. If a button is defined as an HTML button or hyperlink, and if it has an associated record field, the label associated with it is only the text of the button. The mouse-over text can only be changed by the lable if the button is defined as a button with an Image label.

Returns

A text string containing the short name of the field for the specified label ID.

Example

The following code sets the label for a field to one of two different texts, based on the page.

```
Local Field &FIELD;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&FIELD = GetField(RECORD.MYFIELD);
   If %Page = PAGE.PAGE1 Then
      &FIELD.Label = &FIELD.GetShortLabel("LABEL1");
   Else If %Page = PAGE.PAGE2 Then
      &FIELD.Label = &FIELD.GetShortLabel("LABEL2");
End-If;
```

If the Label ID is the same as the name of the field, you could use the following:

```
&LABELID = &FIELD.Name;
&FIELD.Label = &FIELD.GetShortLabel(&LABELID);
```

See Also

Chapter 15, "Field Class," GetLongLabel, page 495, Chapter 15, "Field Class," Label, page 510 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," SetLabel.

SearchClear

Syntax

SearchClear()

Description

The SearchClear method clears the field values if the field is a search key.

Considerations Using SearchClear and SetDefault

The SetDefault method causes a field to be set to its default value (if there is one) immediately after SearchInit PeopleCode finishes. SetDefault overrides SearchClear. If you call SearchClear for a record, then use SetDefault for a field, the field is set to its default value and the search key values for the rest of the record are cleared.

Parameters

None.

Returns

None.

Example

```
Local Field &FIELD;

&FIELD = GetField(PERSONAL_DATA.EMPLID);
&FIELD.SearchClear();
```

See Also

Chapter 15, "Field Class," SetDefault, page 499 and Chapter 31, "Record Class," SearchClear, page 1394.

SetCursorPos

Syntax

SetCursorPos(**PAGE.***pagename* | **%Page**)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The SetCursorPos method enables you to set the focus to the page field corresponding to the field object (on the specified page). The current page may be specified as %Page.

Restrictions on Use with a Component Interface

This function is ignored (has no effect) when used by a PeopleCode program that's been called by a Component Interface.

Parameters

Parameter	Description
Pagename	The name of the page, preceded by the keyword **Page**. The *pagename* page must be in the current component. You can also pass the %page system variable in this parameter (without the **Page** reserved word).

Returns

None.

Example

The following pseudo-code enables you to set the focus to a related field:

```
GetField(ControlRec.ControlField).GetRelated(RelatedRec.RelatedField).SetCursorPos⇒
(PAGE.pagename);
```

See Also

Chapter 15, "Field Class," SetDefault, page 499 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," SetCursorPos.

SetDefault

Syntax

```
SetDefault()
```

Description

SetDefault sets the value of the field to a null value, or to a default, depending on the type of field.

- If this method is used against data from the component buffers, the next time default processing occurs, it is set to its default value: either a default specified in its record field definition or one set programmatically by PeopleCode located in a FieldDefault event. If neither of these defaults exist, the Component Processor leaves the field blank.
- If this method is used with a field that isn't part of the data buffer (for example, a field in a record object instantiated with CreateRecord) the field is automatically set to its default value if one is set for the *field*, not for the record field. Any FieldDefault PeopleCode will not be run on these types of fields. If you want to set the default values for all the fields in a record, use the SetDefault record class method.

Blank numbers correspond to zero on the database. Blank characters correspond to a space on the database. Blank dates and long characters correspond to NULL on the database. SetDefault gives each field data type its proper value.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

None.

Example

```
&CHARACTER.SetDefault();
```

See Also

Chapter 31, "Record Class," SetDefault, page 1398 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide*, "PeopleCode and the Component Processor," Default Processing.

Field Class Properties

In this section, we discuss the Field class properties. The properties are discussed in alphabetical order.

DataAreaCollapsed

Description

This property specifies whether the default initial view of the data area of a group box is collapsed or expanded.

Note. You must set the Collapsible Data Area on the properties for the group box in PeopleSoft Application Designer for this property to have any effect.

This property changes to reflect the current state of the data area, according to whether the user has collapsed or expanded it. Changing the value collapses or expands the data area, but it does *not* prevent the user from collapsing (or expanding) it themselves.

Note. Because the user can change the value of this property, whatever value is set in PeopleCode isn't guaranteed to be still set the next time it is checked, because the user may have collapsed or expanded the data area in the meantime.

This property overwrites the value of the Default Initial View to Expanded field set in PeopleSoft Application Designer. For example, if Default Initial View to Expanded is selected in PeopleSoft Application Designer, then the value for the DataAreaCollapsed property is set to True, the control initially displays collapsed.

This property takes a Boolean value: True, initially display the data area collapsed, False, initially display the data area expanded.

This property is read-write.

Note. If you want to collapse an entire level-based control, such as a scroll area or a grid, use the DataAreaCollapsed Rowset method.

See Also

Chapter 33, "Rowset Class," DataAreaCollapsed, page 1445

 Copyright © 1988-2007, Oracle. All rights reserved.

DecimalPosition

Description

Use this property to specify the number of digits after the decimal point to be displayed for a field defined as number or signed number.

This property *overwrites* the Decimal Positions value defined for a field in PeopleSoft Application Designer.

Setting the DecimalPosition property to a smaller number than the record field's decimal position causes the displayed data to be truncated. For example, suppose a numeric field in the database has a value 0.005. Setting DecimalPosition to 2 displays 0.00 on page.

This property takes an integer value. The default value is -1, which indicates that the property is not set.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Field Definitions," Specifying Number Field Attributes

DisplayFormat

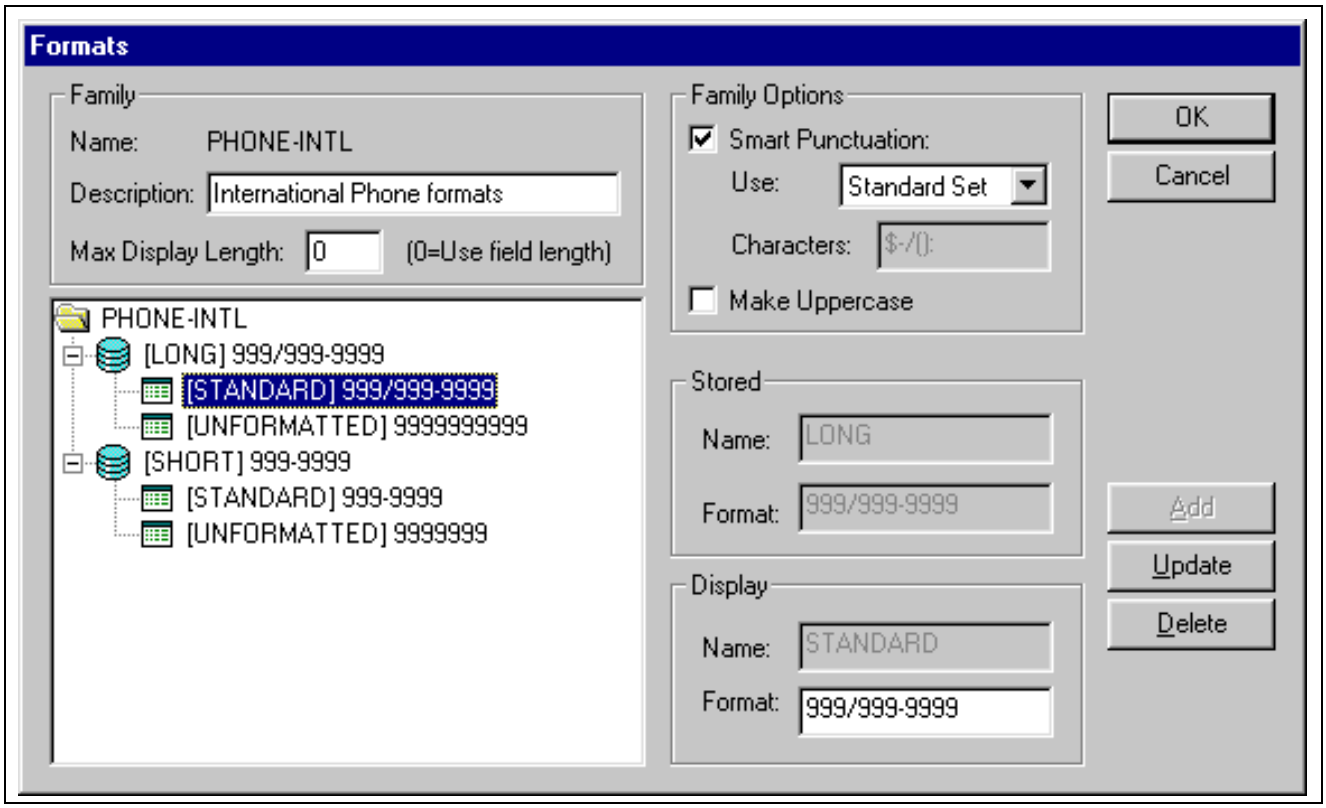
Description

Use this property to specify a custom format to use for the field.

This property is read-write.

The custom format for a field is specified in the field definition. This property enables you to switch between display formats that are defined as part of a custom format. For example, suppose your field used the PHONE-INTL custom format:

Copyright © 1988-2007, Oracle. All rights reserved.



PHONE-INTL format

Both the LONG and the SHORT stored formats have two display formats: STANDARD and UNFORMATTED.

Using this property, you could select either of the display formats. For example:

```
If %Component = PAGE.INTERNATIONAL
   &CHAR.DisplayFormat = "STANDARD";
Else
   &CHAR.DisplayFormat = "UNFORMATTED";
End-If;
```

Note. You *cannot* change the Stored format, but you can find its value using the StoredFormat property.

Example

```
&CHARACTER.DisplayFormat = "STANDARD";
/* this assumes that &CHARACTER is a custom formatted field,
and a display format option is STANDARD */
```

See Also

Chapter 15, “Field Class,” StoredFormat, page 518

DisplayOnly

Description

This property is set to true if the field property DisplayOnly in the PeopleSoft Application Designer is set. May be set to false to change how the field displays.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. This property *overwrites* whatever value is set in PeopleSoft Application Designer.

This property is read-write.

DisplayZero

Description

Use this property to specify if, in a numeric field, zeros or blank are displayed on a page.

This property *overwrites* the Display Zero value on the page field properties dialog box in PeopleSoft Application Designer.

This property takes a Boolean value: true, display zeros, false, display blank. The default values is false.

The value of the DisplayZero property can be changed by the SmartZero property.

If SmartZero is set to true, and your application sets DisplayZero to false in a PeopleCode program, the application server changes the value of DisplayZero to true to display zeros, not blanks. When this occurs, the DisplayZeroChanged property is set to true.

This property is read-write.

See Also

Chapter 15, "Field Class," DisplayZeroChanged, page 503 and Chapter 15, "Field Class," SmartZero, page 517.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Field Definitions," Specifying Attributes for a New Field Definition

DisplayZeroChanged

Description

Use this property to determine if the original DisplayZero property has been changed from its original setting.

The DisplayZeroChanged property is only relevant when the SmartZero property has been set to true. When SmartZero equals true, the application server adjusts the DisplayZero property to display a blank or zero based on the user's input. You can then check the DisplayZeroChanged property to find out if the application server has changed the DisplayZero property.

This property takes a Boolean value: true, the application server has changed the field's DisplayZero property, false otherwise.

For example, suppose you have a PeopleCode program in the RowInit event that sets the DisplayZero property for a field to True, and also sets the SmartZero property to True. When a user deletes the numeric zero in that field on the page, and the page is submitted to the application server, the application server sets the field's DisplayZero property to False so that the new page displays a blank, not a numeric zero. The DisplayZeroChanged property is set to true in this example.

This property is read-only.

See Also

Chapter 15, "Field Class," DisplayZero, page 503 and Chapter 15, "Field Class," SmartZero, page 517.

Copyright © 1988-2007, Oracle. All rights reserved.

EditError

Description

This property is True if an error for this field has been found after executing the ExecuteEdits method with either a message object or a record object. This property can be used with the MessageSetNumber and MessageNumber properties to find the error message set number and error message number.

This property is read-write. After you have fixed the errors, you must set this property to False before running ExecuteEdits again.

Example

The following is an example showing how EditError, along with the ExecuteEdits method could be used:

```
&REC.ExecuteEdits();
If &REC.IsEditError Then
   For &I = 1 to &REC.FieldCount
      If &REC.GetField(&I).EditError Then
         LOG_ERROR();  /* application specific pgm */
      End-If;
   End-For;
End-If;
```

See Also

Chapter 31, “Record Class,” ExecuteEdits, page 1386, Chapter 22, “Message Classes,” ExecuteEdits, page 748, Chapter 15, “Field Class,” MessageNumber, page 512 and Chapter 15, “Field Class,” MessageSetNumber, page 513.

Enabled

Description

This property is True if this field is enabled. May be set False to disable any control that displays this field.

This property is read-write.

Example

```
&CHARACTER.Enabled = True;
&CHARACTER.Enabled = False;
```

FieldLength

Description

This property returns the length of the field as a number.

This property is read-only.

Example

```
&MyRec = CreateRecord(Record.BKREC1);
&MyField = &MyRec.GetField(Field.BKNAME);
&length = &MyField.FieldLength;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
MessageBox(0, "Field Length", 0, 0, "The field BKREC1.BKNAME is length " |⇒
 &length);
```

FormatLength

Description

This property returns the length of the format for the field as a number.

This property is read-only.

Example

```
&field = GetField(Field.BKTEST);
&formatlength = &field.FormatLength;
&length = &field.FieldLength;
&mymsg = "Field.BKTEST length is " | &length | " and format length is " |⇒
 &formatlength;
MessageBox(0, "MetaData Fn Status", 0, 0, &mymsg);
```

FormattedValue

Description

This property returns the value of a field as a string, formatted exactly as it would be displayed on an edit field on a page. This is useful when you're using a prompt field for the label of another field.

Because a record field can be bound to more than one page field, FormattedValue takes the first associated page field, looking first on the current page, then through all the pages in the component. This property returns a null string ("") if used with an Image or ContentReference field. It also returns a null string if no associated page field is found.

This property is read-write.

Example

This property could be used to set up a hyperlink labeled by data generated on a page. To do this, do the following:

1. Define a work field.
2. Associate the work field with the hyperlink page field.
3. Define a hidden page field with the formatting that you want.
4. Associate the hidden page field with the data field.
5. Write PeopleCode (probably for RowInit) to set the label on the work field to be the FormattedValue of the data field.

For example, suppose you wanted a hyperlink labeled by the QE_POSITION_ID field on the QE_PLYR_POSITN record. This field is a prompt table field, edited by table QE_POSITION.

Copyright © 1988-2007, Oracle. All rights reserved.

You want to display the DESCRSHORT field from the prompt table, instead of the QE_POSITION_ID code value. Create a work field, POS_LINK on the WORK_REC record, and create a hyperlink page field associated with the DESCRSHORT field. For the label, create an invisible drop-down list page field, associated with QE_PLYR_POSITN.QE_POSITION_ID. Set its Prompt Table Field to DESCRSHORT. Then put a PeopleCode program on the component RowInit for either the QE_PLYR_POSITN record or the WORK_REC.

```
WORKREC.POS_LINK.Label = QE_PLYR_POSITN.QE_POSITION_ID.FormattedValue;
```

IsAltKey

Description

This property is True if this field references a field definition that is defined as an Alternate Search key in the associated record definition.

This property is read-only.

IsAuditFieldAdd

Description

This property is True if this field references a field definition that is defined as an audit field in the associated record definition, and the field is audited whenever new data is added.

This property is read-only.

IsAuditFieldChg

Description

This property is True if this field references a field definition that is defined as an audit field in the associated record definition, and the field is audited whenever data is changed.

This property is read-only.

IsAuditFieldDel

Description

This property is True if this field references a field definition that is defined as an audit field in the associated record definition, and the field is audited whenever data is deleted.

This property is read-only.

IsAutoUpdate

Description

This property is True if this field references a field definition that is defined as an Auto-Update field in the associated record definition.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

IsChanged

Description

This property returns True if the value for the field has been changed.

Note. Use this property with the primary database record only. It will not return valid results if used with a work record.

This property is read-only.

Example

```
If &CHARACTER.IsChanged Then
    Warning ("The character field has been changed");
End-If;
```

IsDateRangeEdit

Description

This property is True if this field references a field definition that is defined as requiring a Reasonable Date in the associated record definition.

This property is read-only.

IsDescKey

Description

This property is True if this field references a field definition that is defined as a descending key in the associated record definition.

This property is read-only.

IsDuplKey

Description

This property is True if this field references a field definition that is defined as a duplicate key in the associated record definition.

This property is read-only.

IsEditTable

Description

This property is True if this field references a field definition that has a table edit type of a Prompt Table with Edit.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

IsEditXlat

Description

This property is True if this field references a field definition that has a table edit type of a Translate Table Edit.

This property is read-only.

IsFromSearchField

Description

This property is True if this field references a field definition that is defined as a From Search Field in the associated record definition.

This property is read-only.

IsInBuf

Description

This property returns True if the data of the field is present in the data buffers. For example, all of the fields of a derived record are not present in the data buffer.

This property is read-only.

Example

The following example iterates over all the fields in a record. The code verifies that the data for the field is accessible before it tries to assign the value to a variable.

```
For &I = 1 to &REC.FieldCount
   &FIELD = &REC.GetField(&I);
   If &FIELD.IsInBuf Then
      &VALUE = &FIELD.Value;
      /* do other processing */
   End-If;
End-For;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Referencing Data in the Component Buffer," Record Fields and the Component Buffer

IsKey

Description

This property is True if this field references a field definition that is defined as a primary key of the associated record definition.

This property is read-only.

Example

```
If &CHARACTER.IsKey Then
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
    Warning ("The field " | &Character.Name | " is a key");
End-If;
```

IsListItem

Description

This property is True if this field references a field definition that is defined as a List Box item of the associated record definition.

This property is read-only.

IsNotUsed

Description

This property is True if this field has been set as NotUsed with the SetDBFieldNotUsed function.

This property is read-only.

Example

```
&field = GetField(Field.RC_TEST_PB);
&ret = &field.IsNotUsed;
If (&ret = True) Then
   MessageBox(0, "MetaData Fn Status", 0, 0, "Field is NOTUSED");
Else
   MessageBox(0, "MetaData Fn Status", 0, 0, "Field is used.");
End-If;
```

IsRequired

Description

This property is True if this field references a field definition that is defined as Required in the associated record definition.

This property is read-only.

IsSearchItem

Description

This property is True if this field references a field definition that is defined as a Search key in the associated record definition.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

IsSystem

Description

This property is True if this field references a field definition that is defined as System Maintained field in the associated record definition.

This property is read-only.

IsThroughSearchField

Description

This property is True if this field references a field definition that is defined as a Through Search Field in the associated record definition.

This property is read-only.

IsUseDefaultLabel

Description

This property is True if this field references a field definition that has its label defined as Use Default Label in the associated record definition.

This property is read-only.

IsYesNo

Description

This property is True if this field references a field definition that has a table edit type of a Yes/No Table Edit.

This property is read-only.

Label

Description

This property returns a reference to the label you have set for the field. You can use this property to change the label text for the field when it's displayed on a page.

This property returns a blank string if you use it before you've set the label with either the SetLabel function or with this property.

To return the text of a label before you set it, use the GetShortLabel or GetLongLabel methods.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters.

The "tool tip," or mouse over text, that appears with a hyperlink at runtime comes from the RFT long label assigned to the record field. However, the RFT long label displays only if it is different from the assigned display value of the hyperlink and it is not null. If the link is an image button, the tool tip is derived from the label text if there is any. Otherwise, the RFT long label is used.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Example

The following code sample changes all the field labels for a record to the short label. It assumes that there is a label with the same name as the name of the field for all fields in the record.

```
Local Field &FLD;
Local Record &REC;

If CHECK_FIELD Then
   &REC = GetRecord();
   For &I = 1 to &REC.FieldCount
      &FLD = &REC.GetField(&I);
      &LABELID = &FLD.Name;
      &FLD.Label = &FLD.GetShortLabel(&LABELID);
   End-For;
End-If;
```

See Also

Chapter 15, “Field Class,” GetShortLabel, page 497 and Chapter 15, “Field Class,” GetLongLabel, page 495.

LabelImage

Description

Use this property to set the image for a button. This property is only valid for button controls. This property can be set even when the button is disabled or display-only.

This property only *overrides* the existing image. It doesn’t change it. The next time the page is accessed, the original image is used.

The value for this property can be either of the following:

```
Image.imagename
```

or

```
"Imagename"
```

PeopleSoft strongly recommends that if the PeopleCode program sets the LabelImage for an active button, it also sets the image appropriately if the button is disabled.

This property is read-write.

Example

```
Local Field &MyField;

&MyField = GetField(BUTTON_BOTTOM);
&MyField.LabelImage = Image.BTN_REFRESH;
```

Copyright © 1988-2007, Oracle. All rights reserved.

LongTranslateValue

Description

This property returns a string that contains the Long translate (XLAT) value of the field if the field is based on a translate table.

If the field has a null value, a null string is returned. If the field isn't based on a translate table, or the value isn't in the translate table, the field's current value is returned. Because the current value can be of any type, this property has a type of Any.

Note. If you're accessing a field based on the Translate table, the Value property returns only the one or two letter XLAT value.

Use the ShortTranslateValue property to return the short translate value of a field.

This property is read-only.

Example

```
Local Any &VALUE;
Local Field &MYFIELD;

&MYFIELD = GetField();
&VALUE = &MYFIELD.LongTranslateValue;
If ALL(&VALUE) Then
   /* do processing */
End-if;
```

See Also

Chapter 15, "Field Class," Value, page 521 and Chapter 15, "Field Class," ShortTranslateValue, page 516.

MessageNumber

Description

This property returns the error message number (as a number) if an error for this field is found after executing ExecuteEdits method. You can use this property in conjunction with the EditError property (which can be used to determine whether there are any errors) and the MessageSetNumber property (which contains the error message set number if an error is found.)

This property is read-only.

Example

```
For &I = 1 to &RECORD.FieldCount
   &MYFIELD = &RECORD.GetField(&I);
   If &MYFIELD.EditError Then
      &MSGNUM = &MYFIELD.MessageNumber;
      &MSGSET = &MYFIELD.MessageSetNumber;
      /* Do processing */
   End-If;
End-For;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

MessageSetNumber

Description

This property returns the error message set number (as a number) if an error for this field is found after executing ExecuteEdits method. You can use this property in conjunction with the EditError property (which can be used to determine whether there are any errors) and the MessageNumber property (which contains the error message number if an error is found.)

This property is read-only.

Example

```
For &I = 1 to &RECORD.FieldCount
   &MYFIELD = &RECORD.GetField(&I);
   If &MYFIELD.EditError Then
      &MSGNUM = &MYFIELD.MessageNumber;
      &MSGSET = &MYFIELD.MessageSetNumber;
      /* Do processing */
   End-If;
End-For;
```

Name

Description

This property returns the name of the field definition that the field object is based on as a string value.

This property is read-only.

Example

```
WinMessage("The character field's name is : " | &CHARACTER.Name);
```

OriginalValue

Description

This property returns the value of a field, that is, the value that from the database. If the value hasn't been changed and saved by the user, it is the original value from the database. If the value has been changed and saved by the user, it is the existing value in the database.

Note. This property does *not* work for derived records. Original values are the database values, and derived records do not have a corresponding database value.

This property is read-only.

Example

```
&Orig = &MyField.OriginalValue;
If &Orig = &Date Then
   /* do current day processing */
Else
   /* do other processing */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
End-If;
```

ParentRecord

Description

This property returns a reference to the record object for the record containing the field.

This property is read-only.

Example

```
&NUMBER_OF_FIELDS = &CHARACTER.ParentRecord.Fieldcount;
/* note that FieldCount is a property of the Record class */
```

PromptTableName

Description

This property returns the name of the prompt table (if any) associated with this field. This property returns a string value. This property returns Null if no prompt table is associated with the field.

This property is read-only.

SearchDefault

Description

If this property is set to True, system defaults (the default values set in the record field definitions) are enabled on search dialogs for the field. Setting this property to True *does not* cause the FieldDefault event to fire.

The system default is done only once, when the search dialog first starts, immediately after any SearchInit PeopleCode. If the user subsequently blanks out a field, the field isn't reset to the default value. Setting SearchDefault to False disables default processing for the field executing the property.

SearchDefault is effective only when used in SearchInit PeopleCode programs.

This property is read-write.

Example

```
&CHARACTER.SearchDefault = True;
/* assuming &CHARACTER is a search key, and has a default value */
```

This example turns on edits and system defaults for the SETID field in the search dialog box:

```
&SETID = GetField(INV_ITEMS.SETID);
&SETID.SeachDefault = True;
&SETID.SearchEdit = True;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

SearchEdit

Description

If this property is set to True, SearchEdit enables system edits (edits specified in the record field definition) for the field, for the life of the search dialog box. Setting SearchDefault to False disables system edits. In the Add mode search dialog box, the following edits are performed when the end-user clicks the Add button. In any other mode, the following edits are performed when the end-user clicks the Search button:

- Formatting
- Required Field
- Yes/No Table
- 1/0 Table
- Translate Table
- Prompt Table

SearchEdit does not cause the FieldEdit, FieldChange, or SaveEdit PeopleCode events to fire during the search dialog.

You might use SearchEdit to control access to the system. For example, you can apply this function to the SETID field of a dialog box and require the user to enter a valid SETID before they are able to click OK on the search dialog box.

This property is read-write.

Considerations Using SearchEdit

If you use this method in the SearchInit event, the search page options are limited to the "=" and "IN" operators.

Example

```
&CHARCTER.SearchEdit = True;
/* assuming &CHARACTER is a search key, and contains an edit table
such as translate table defined in its field properties.  */
```

This example turns on edits and system defaults for the SETID field in the search dialog box:

```
&SETID = GetField(INV_ITEMS.SETID);
&SETID.SeachDefault = True;
&SETID.SearchEdit = True;
```

SetComponentChanged

Description

This property determines whether a change to a field (using PeopleCode) marks the component as changed and the data written to the database if the field is a database field.

This property takes a Boolean value: true, changes to the field mark the component as changed, false, the component is treated as if unchanged. The default value is true for database fields and false for derived fields.

Copyright © 1988-2007, Oracle. All rights reserved.

The Set Component Changed checkbox for a page field in PeopleSoft Application Designer determines if a user's action on a page marks the component as changed. For example, if Set Component Changedfor an edit box control is cleared in PeopleSoft Application Designer, then user's editing on the edit box does not mark the component as changed and no save warning message is displayed if the user leaves the page without saving. The SetComponentChanged property for a field determines if a change on the field value using PeopleCode marks the component as changed and a save warning is issued when the user tries to exit the page without saving.

This property is only applicable after the Page Activate event has run. If you try to use this property before or during Page Activate, the user does not receive a warning if data has been changed.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Page Definitions," Setting Use Properties

Chapter 33, "Rowset Class," SetComponentChanged, page 1451

ShortTranslateValue

Description

This property returns a string that contains the Short translate (XLAT) value of the field if the field is based on a translate table.

If the field has a null value, a null string is returned. If the field isn't based on a translate table, or the value isn't in the translate table, the field's current value is returned. Because the current value can be of any type, this property has a type of Any.

Note. If you're accessing a field based on the translate table, the Value property returns only the one or two letter XLAT value.

Use the LongTranslateValue property to return the long translate value of a field.

This property is read-only.

Example

```
Local Any &VALUE;
Local Field &MYFIELD;

&MYFIELD = GetField();
&VALUE = &MYFIELD.ShortTranslateValue;
If ALL(&VALUE) Then
   /* do processing */
End-if;
```

See Also

Chapter 15, "Field Class," Value, page 521 and Chapter 15, "Field Class," LongTranslateValue, page 512.

 Copyright © 1988-2007, Oracle. All rights reserved.

ShowRequiredFieldCue

Description

With PeopleTools 8, an asterisk (*) is displayed on pages beside fields that are defined as Required in PeopleSoft Application Designer. You can use this property to specify whether this asterisk, also called the *required field cue*, is displayed for a particular field.

For example, many fields are made required or non-required either procedurally or through PeopleCode. This means they aren't defined as Required in PeopleSoft Application Designer, and the end-user may be confused. For these fields, you can use this property.

Note. This property affects only fields where a required field cue is otherwise permissible. That is, regardless of the setting of the property, no cue is ever shown on a pushbutton, a display-only field, and so on.

This property is read-write.

SmartZero

Description

Use this property to specify if FieldChange PeopleCode programs are run when a user changes a zero to a blank, or a blank to a zero, for a field on a page.

This property takes a Boolean value: true, run FieldChange PeopleCode programs, false, do not. The default value is false.

When you set this property to true, in addition to treating a user change of blank-to-zero and zero-to-blank as a field change, the application server also adjusts the DisplayZero property for the runtime field object to display a blank or zero as entered by the user.

This property is read-write.

See Also

Chapter 15, "Field Class," DisplayZero, page 503 and Chapter 15, "Field Class," DisplayZeroChanged, page 503.

SqlText

Description

This property is valid only for fields that have a dynamic view as their prompt record. If you set SqlText to a non-null value, that text is used instead of the dynamic view's normal text used for prompting.

Suppose you wanted to have a different prompt table depending on the settings of other fields in the row. Normally you could use %EDITTABLE to dynamically specify the prompt table you want. However in this case there are too many possible combinations of values, which would require too many views. Furthermore, the values are customizable by the end-user or the application, which means even if you, the developer, wanted to, you couldn't provide all the combinations of views necessary. However you can generate the desired SQL text for the view in PeopleCode based on what the user enters.

Copyright © 1988-2007, Oracle. All rights reserved.

If you use a dynamic view as the prompt table, and have the dynamic view contain a SQL object that is updated from PeopleCode, you could achieve this functionality. However, a SQL object is a shared object, so if multiple users used the same page, they overwrite each other's settings and the SQL object contains the SQL for the most recent user. Similarly if a single user had multiple rows on a page, the SQL object is valid only for the most recent row. This means if the user went to another row and did a prompt, they would get the wrong values again.

The purpose of this property is to enable you to specify the generated SQL text independently for each occurrence in each transaction. It enables you to override the text of a dynamic view being used as a prompt table on a field by field basis.

It is up to the developer to verify that the text specified for this property is valid, that is, that it selects the correct number of fields for the record definition, and so on.

This property is read-write.

StoredFormat

Description

This property returns the custom character format (as a string) for the field executing the property.

If the field doesn't have a custom format associated with it, the user receives a runtime error message.

If the field has a display format associated with it, you can change that using the DisplayFormat property.

This property is read-only.

Example

```
WinMessage("The character field's custom stored format is : " | &CHARACTER.Stored⇒
Format);

/* this assumes that &CHARACTER is a custom formatted */
/* field */
```

See Also

Chapter 15, "Field Class," DisplayFormat, page 501

Style

Description

If a page has a style sheet associated with it, this property can be used to specify a different style class with a field. This property overrides only the existing style. It does not change it. The next time the page is accessed, the original style is used.

This property takes a string value.

This property is read-write.

Example

The following example associates a test field first with a style class depending on the value of the field.

```
Local Field &field;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&field = GetField();

If TESTFIELD1 = 1 Then;
   &field.Style = "PSHYPERLINK";
End-If;

If TESTFIELD1 = 2 Then;
   &field.Style = "PSIMAGE";
End-If;
```

Field with PSHYPERLINK style

Field with PSIMAGE style

SystemIDFieldName

Description

This property returns the name of the System ID field as a string. If there is no defined value (the default) the value is a null string ("").

This property accesses the System ID field name value specified on the Use tab of the Record Properties dialog box.

The System ID field is used to create a unique way to identify the record for mobile synchronization purposes. This property is used exclusively for mobile applications. This property is not available for subrecords.

This property is read-only.

Example

```
Local Record &REC;

&REC = GetRecord();
&sSystemID = &REC.SystemIDFieldName;
```

See Also

Chapter 15, “Field Class,” TimeStampFieldName, page 520

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, “Creating Record Definitions,” Setting Record Properties

Copyright © 1988-2007, Oracle. All rights reserved.

TimeStampFieldName

Description

This property returns the name of the Timestamp field as a string. If there is no defined value (the default) the value is a null string ("").

This property accesses the Timestamp field name value specified on the Use tab of the Record Properties dialog box.

The Timestamp field is used to specify a field that is automatically updated with the date and time when there's a change to the record. This property is used exclusively for mobile applications.

This property is not available for subrecords. This property is read-only.

Example

```
Local Record &REC;

&REC = GetRecord();
&sTimeStamp = &REC.TimeStampFieldName;
```

See Also

Chapter 15, "Field Class," SystemIDFieldName, page 519

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Record Definitions," Setting Record Properties

Type

Description

This property returns the type of field. The values can be one of the following strings:

- CHAR
- DATE
- DATETIME
- IMAGE (for static images)
- IMAGEREFERENCE
- LONGCHAR
- NUMBER
- SIGNEDNUMBER
- TIME

Note. Fields of type Attachment have a type of IMAGE.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
If &CHARACTER.Type = "NUMBER" Then
   /* perform processing */
Else
   /* error processing */
End-If;
```

Value

Description

This property contains the current value of the field, converted to an appropriate PeopleCode data type.

In most contexts, the Value property can be used to assign a new value to the field. However, there are some restrictions:

- You cannot assign the Value property in any RowSelect PeopleCode program.
- You cannot assign the current record field value to the Value property in any FieldEdit PeopleCode program.

Note. If you're accessing a field based on the translate table, Value returns only the one letter XLAT value. To get the full XLAT value, use the LongTranslateValue or ShortTranslateValue properties.

This property is read-write.

Considerations Using INSTALLATION or OPTIONS Tables

When using the INSTALLATION or OPTIONS table, you cannot use the Value property. You must use the old style format, not the field object format. For example, the following code is invalid:

```
If %Page = Page.GP_RUN_TYPE And
      INSTALLATION.TL.Value = "N" Then
   &RS2.HideAllRows();
```

While the following code is valid:

```
If %Page = Page.GP_RUN_TYPE And
      INSTALLATION.TL = "N" Then
   &RS2.HideAllRows();
```

Considerations Using Character Values

Previously, if you had an edit-box field, and if the end-user selected the value in it and deleted the value, leaving the field empty, the value of the field in PeopleCode was not an empty (zero-length) string.

Now, the value of such a field *is* an empty (zero-length) string.

In addition, if the user adds one or more space characters to a field, the field still returns a Null string.

The following is how to check for this:

```
If (fieldname.Value = "") Then
```

Example

```
&CHARACTER.Value = "Hello";
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 15, “Field Class,” LongTranslateValue, page 512 and Chapter 15, “Field Class,” ShortTranslateValue, page 516.

Visible

Description

This property is True if this field is visible in the page displaying it. Setting this property to False hides the field. Because every field is implicitly associated with a rowset, row, and record, setting the Visible property for a field on the first page of a component hides only that field. If that field is repeated on other pages in the component, the other occurrences of the field aren’t hidden.

This property is read-write.

Example

The following code hides the field &ROUND_OPTION based on the value of AVG_DFLT_OPTION:

```
Local field &ROUND_OPTION;

&ROUND_OPTION = GetField();

If AVG_DFLT_OPTION = "Y" Then
   &ROUND_OPTION.SetDefault();
   &ROUND_OPTION.Visible = True;
Else
   &ROUND_OPTION.Visible = False;
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 16

File Class

This chapter describes the File class, which provides methods and properties for reading from and writing to external files and discusses the following topics:

- Data type of a file object
- Scope of a file object
- File security considerations
- File access interruption recovery
- Plain text files
- Automatic PeopleCode generation
- File layout error processing
- File class built-in functions
- File class methods
- File class properties
- File layout examples
- Multiple file layouts

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Constructing File Layouts and Performing Data Interchanges"

Understanding File Layout

PeopleTools supports reading and writing to plain text files, and to files that have a format based on a File Layout definition that has been created in PeopleSoft Application Designer.

- If the file is a plain text file, data is read or written using text strings.
- If the file is based on a File Layout, you can use text strings, rowset, or record objects.

This simplifies reading, writing, and manipulating hierarchical transaction data with PeopleCode.

File layout methods and properties are noted as such in their descriptions.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 16, “File Class,” Plain Text Files, page 526

Chapter 16, “File Class,” File Layout Examples, page 557

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, “Constructing File Layouts and Performing Data Interchanges”

Data Type of a File Object

The File Object is an instance of the File class. A file object is declared using the File data type. For example,

```
Local File &MYFILE;
```

This creates an object &MYFILE of the class File.

Scope of a File Object

A file object can only be instantiated from PeopleCode. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

A file object is passed to and returned from PeopleCode functions and methods as a reference, so the file object is never copied; rather, alternate identifiers are used to refer to the same object. Any change made to a file using one identifier has the same effect as it would with any other identifier for that file.

In the following code example, two variables, &F1 and F2, are declared by using the data type File. The file is opened using the GetFile built-in function. Next, &F1 is assigned to &F2. This does *not* copy &F1 to &F2. Instead, &F1 and &F2 both refer to the same object. Therefore, in the last step when &F2 is closed, &F1 is closed too.

```
Local File &F1, &F2;

&F1 = GetFile("somefile.txt", "R");
If &F1.IsOpen Then
   &F2 = &F1;
   &F2.Close();  /* Now &F1 is also closed. */
End-if;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetFile

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, “Understanding Objects and Classes in PeopleCode,” Assigning Objects

 Copyright © 1988-2007, Oracle. All rights reserved.

File Security Considerations

When you're using file objects in PeopleCode that might run on a server, you must be aware of some security concerns. The underlying system doesn't provide security checks on access to the files. Instead, the PeopleCode programs use whatever authority the server process has, and *not* that of the user. This means you must write your PeopleCode program to prevent the user from reading or writing files that they should not. In particular, be especially wary of opening files where any part of the filename is derived from user input.

File Access Interruption Recovery

You can use the GetPosition or SetPosition methods to minimize the loss of work in the event of a system failure during file access. To use them and recover from access interruptions, you must access the file in Update mode using the GetFile built-in function or the Open method, specifying the mode parameter "U". You need to use this mode in anticipation of possible interruptions if you want to recover from them later.

To start reading or writing from the beginning of a file, use SetPosition to set a read/write position of 0, immediately after you open the file in Update mode.

Warning! In Update mode, any write operation clears the file of all data that follows the position you set.

When reading from or writing to a file, use GetPosition periodically to determine your current byte position in the file. This establishes a checkpoint to which you can return after a failure. You must save the checkpoint value in a separate file or a database so you can retrieve it later. How often you save a checkpoint depends on your requirements: the less work you want to redo, the more frequent your checkpoints should be. You may also want to save information identifying the data you're reading or writing at the time.

After a failure interrupts your file access, reopen the file in Update mode. You can then retrieve the last checkpoint value you saved, and use SetPosition to apply that value. Your read/write position in the file is the position of the last checkpoint, and you can continue reading or writing from there.

Use the Update mode with GetPosition and SetPosition only for recovering from interruptions as described here, or for starting at the beginning of the file again.

Note. For fixed-width file layouts, the field start positions and lengths are computed in 8-bit bytes, except in the case of UCS2 or UTF-16 where they are computed in 16-bit double-bytes. This may create complications when using variable width encodings (for example, UTF-8 or Shift-JIS), when the data is elsewhere calculated in characters (rather than bytes), because the data involved may contain characters of varying byte-width.

See Also

Chapter 16, "File Class," Open, page 534

Chapter 16, "File Class," SetPosition, page 543

Chapter 16, "File Class," GetPosition, page 532

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

Copyright © 1988-2007, Oracle. All rights reserved.

Plain Text Files

To read and write from plain text files involves reading and writing strings of data. These text strings can be manipulated with built-in string functions, like RTrim, Find, Replace, and so on.

Note. If your data is hierarchical in nature, or based on existing PeopleSoft records or pages, use a File Layout definition for reading and writing your data, rather than doing it line by line (or field by field.)

The following example creates an array of array of string, then reads in two files, one into each "column" of the array. The Names file contains names; the Numbers file contains employee numbers. The ReadLine method reads each successive line in a file, until it reaches the end of the file.

Notice that the first file is opened using the GetFile function. The second file is not opened using GetFile, but rather with the Open method. After the data is read into the array, you can do processing on the data. The end of the program writes the changes back to the files, using the WriteLine method, which includes a system end of line character at the end of every line.

```
Local array of array of string &BOTH;
Local File &MYFILE;
Local string &HOLDER;

/* Create empty &BOTH array */
&BOTH = CreateArrayRept(CreateArrayRept("", 0), 0);

/* Read first file into first column */

&MYFILE = GetFile("names.txt", "R");
While &MYFILE.ReadLine(&HOLDER);
   &BOTH.Push(&HOLDER);
End-While;

/* read second file into second column */

&MYFILE.Open("numbers.txt", "R");
&LINENO = 1;
While &MYFILE.ReadLine(&HOLDER);
   If &LINENO > &BOTH.Len Then
      /* more number lines than names, use a null name */
      &BOTH.Push(CreateArray("", &HOLDER));
   Else
      &BOTH[&LINENO].Push(&HOLDER);
   End-If;
   &LINENO = &LINENO + 1;
End-While;

/* if more names than numbers, add null numbers */
For &LINENO = &LINENO to &BOTH.Len
      &BOTH[&LINENO].Push("");
End-For;
&MYFILE.Close();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/* do processing with array */
/* write data back to files */

&MYFILE1 = GetFile("names.txt", "A");
&MYFILE2 = GetFile("numbers.txt", "A");

/* loop through array and write to files */

For &I = 1 To &BOTH.Len
   &STRING1 = &BOTH[&I][1];
   &MYFILE1.writeline(&STRING1);
   &STRING2 = &BOTH[&I][2];
   &MYFILE2.writeline(&STRING2);
End-For;

&MYFILE1.Close();
&MYFILE2.Close();
```

See Also

Chapter 16, “File Class,” File Layout Examples, page 557

Automatic PeopleCode Generation

After you create a File Layout definition, you can use PeopleCode to access it. This PeopleCode can be long and complex. Rather than write it directly, you can drag and drop the File Layout definition from Application Designer Project View into an open PeopleCode edit pane. This is primarily used for importing data. Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to meet your requirements.

The following is just a snippet of the code that is generated:

```
Function EditRecord(&REC As Record) Returns boolean ;
   Local integer &E;
REM   &REC.ExecuteEdits(%Edit_Required + %Edit_DateRange + %Edit_YesNo + %Edit_⇒
TranslateTable + %Edit_PromptTable + %Edit_OneZero);
   &REC.ExecuteEdits(%Edit_Required + %Edit_DateRange + %Edit_YesNo + %Edit_One⇒
Zero);
   If &REC.IsEditError Then
      For &E = 1 To &REC.FieldCount
         &MYFIELD = &REC.GetField(&E);
         If &MYFIELD.EditError Then
            &MSGNUM = &MYFIELD.MessageNumber;
            &MSGSET = &MYFIELD.MessageSetNumber;
            &LOGFILE.WriteLine("****Record:" | &REC.Name | ", Field:" |⇒
 &MYFIELD.Name );
            &LOGFILE.WriteLine("****" | MsgGet(&MSGSET, &MSGNUM, ""));
         End-If;
      End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      Return False;
   Else
      Return True;
   End-If;
End-Function;
.
.
.
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Constructing File Layouts and Performing Data Interchanges"

End Of Line Considerations

Input files for file layouts of type fixed or CSV must not contain embedded CR/LF with the field data.

File Layout Error Processing

If an error occurs on any field in any record of a rowset object populated with the ReadRowset method, the rowset object's property IsEditError returns True. For example, you can use the method ExecuteEdits on a record, to verify that the data in the record is valid (has the correct format, the right data type, and so on.) This type of error is indicated by the IsEditError.

In some instances, however, the rowset object won't receive the error; in that case the file object's IsError property returns True. To discover all field errors, check both properties after executing ReadRowset.

To determine which field has the error, you must examine the EditError property of every field in the rowset to find the one returning True. You can then examine that field's MessageSetNumber and MessageNumber properties to determine the relevant error message. The following example shows how this might be done:

```
&MYFILE.Open(&SOMENAME, "R");
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT);
&MYROWSET = &MYFILE.ReadRowset();
If &MYFILE.IsError Then
  For &I = 1 to &MYROWSET.ActiveRowCount
    If &MYROWSET.GetRow(&I).IsEditError Then
      &ROW = &MYROWSET.GetRow(&I);
      For &J = 1 to &ROW.RecordCount
        If &ROW.GetRecord(&J).IsEditError Then
          &REC = &ROW.GetRecord(&J);
          For &K = 1 to &REC.FieldCount
            If &REC.GetField(&K).EditError Then
              /* Examine the field&rsquo;s
                 MessageSetNumber and MessageNumber properties,
                 and respond accordingly */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
            End-If;
             End-For;
          End-If;
          End-For;
        End-If;
      End-For;
    End-If;

    &MYFILE.Close();
```

Note. Only field errors set the IsError, IsEditError, or EditError properties. All other errors triggered by File class methods terminate the PeopleCode program.

See Also

Chapter 16, "File Class," IsError, page 552

Chapter 16, "File Class," ReadRowset, page 538

Chapter 16, "File Class," IgnoreInvalidId, page 551

Chapter 33, "Rowset Class," IsEditError, page 1449

Chapter 31, "Record Class," ExecuteEdits, page 1386

Chapter 22, "Message Classes," ExecuteEdits, page 748

Chapter 15, "Field Class," MessageNumber, page 512

Chapter 15, "Field Class," MessageSetNumber, page 513

File Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FileExists

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FindFiles

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

File Class Methods

In this section, we discuss the File class methods. The methods are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

Close

Syntax

```
Close()
```

Description

The Close method discards any changes that haven't been written to the external file, disassociates the file object from the external file, releases all resources connected with the file object, and causes the file object to go out of scope.

You cannot use any methods or properties on the object after it is closed. You must get another file object (using the GetFile function) and instantiate another file object after you use Close.

Parameters

None.

Returns

None.

Example

```
&MYFILE.Open("somefile.txt", "W", %FilePath_Relative);
&MYFILE.WriteLine("Some text.");
&MYFILE.Close();
```

See Also

Chapter 16, "File Class," Open, page 534 and Chapter 16, "File Class," IsOpen, page 553.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

CreateRowset

Syntax

```
CreateRowset()
```

Description

The CreateRowset method is a file layout method. It instantiates a PeopleCode rowset object containing one unpopulated row, based on the file layout definition specified with SetFileLayout.

Note. You must specify a file layout using the SetFileLayout method before using CreateRowset, otherwise it returns NULL.

See Chapter 16, "File Class," SetFileLayout, page 542.

After the empty rowset object has been created, you can use Rowset class methods such as Select or Fill, and built-in functions such as GetLevel0 or GetRowset, to populate the rowset with data. After the rowset contains data, you can use the method WriteRowset to write the data to a file.

Don't use CreateRowset when reading from a file. Instead, use the ReadRowset method to instantiate and populate rowsets with data from the file.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

An empty rowset object.

Example

```
&SOMEROWSET = &MYFILE.CreateRowset();
```

The following rowset is created from a file object, then populated with data using the GetLevel0 function:

```
&FILEROWSET = &MYFILE.CreateRowset();
&FILEROWSET = GetLevel0();
&MYFILE.WriteRowset(&FILEROWSET, True);
```

The following rowset is created from a file object, then populated with data using Fill method:

```
&FILEROWSET = &MYFILE.CreateRowset();
&NUM_READ = &FILEROWSET.Fill("where MYRECORD = :1", &UVAL);
```

See Also

Chapter 16, “File Class,” ReadRowset, page 538, Chapter 16, “File Class,” SetFileLayout, page 542 and Chapter 16, “File Class,” WriteRowset, page 548.

Chapter 33, “Rowset Class,” page 1421

Delete

Syntax

```
Delete()
```

Description

Use the Delete method to delete an open file. You cannot delete a closed file.

Parameters

None.

Returns

None.

Example

```
/* Open a temporary file. Use the "N" parameter to ensure
the file is new. */

&MyFile = GetFile("temp.txt", "N");

/* verify the file was instantiated successfully */

/* do processing using the temporary file */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* delete the temporary file */

&MyFile.Delete();
```

See Also

Chapter 16, "File Class," Open, page 534 and Chapter 16, "File Class," Close, page 530.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

GetPosition

Syntax

```
GetPosition()
```

Description

The GetPosition method returns the current read or write position in the external file. GetPosition works only with a file that was opened in Update mode. This method is designed to be used in combination with the SetPosition method to establish checkpoints for restarting during file access.

Note. Currently, the effect of the Update mode and the GetPosition and SetPosition methods is not well defined for Unicode files. Use the Update mode only on files stored with a non-Unicode character set.

Parameters

None.

Returns

A number representing the current read or write position in the file.

Note. When you use ReadRowset to read from a file in Update mode, the CurrentRecord property returns, the entire record just read. The current read/write position is at the *end* of the CurrentRecord, that is, just past the end of the rowset.

Example

The following example opens a file in Update mode, and saves the current position after each read operation:

```
&MYFILE.Open(&SOMENAME, "U");
If &MYFILE.IsOpen Then
   while &MYFILE.ReadLine(&SOMESTRING)
      &CURPOS = &MYFILE.GetPosition();
      /* Save the value of &CURPOS after each read,
         and process the contents of each &SOMESTRING */
   End-While;
End-If;
&MYFILE.Close();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 16, "File Class," Open, page 534 and Chapter 16, "File Class," SetPosition, page 543.

Chapter 16, "File Class," File Access Interruption Recovery, page 525

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

GetString

Syntax

```
GetString([Strip_Line_Terminator])
```

Description

Use the GetString method to return the entire file as a single string.

Note. After this method completes successfully, the original file is deleted.

You can specify whether the resulting string is to include the line terminator characters or not by using the *Strip_Line_Terminator* parameter. The default value for this parameter is false, which means the resulting string includes the line terminator characters at the end of each line.

For example on a Unix system, with a line terminator of a LF, the resulting string includes not only the data for each line, but the LF character as well.

Parameters

Parameter	Description
Strip_Line_Terminator	Specify whether the line terminators are to be stripped or not.

Returns

A single string containing the entire contents of the file.

Example

The following example creates a file on a Windows system, then retrieves it as a single line of text.

Note. The file is destroyed on successful completion of this method.

```
Local File &File = GetFile("c:\temp\junk\something.txt", "W", %FilePath_Absolute)
/* write a bunch of > 2048 length lines */

Local string &piece, &input;
Local integer &I;

&piece = "123456789-";
While Len(&piece) < 2048
   &piece = &piece | &piece;
End-While;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&File.WriteString(&piece);
&File.WriteString(&piece);
&File.WriteString(&piece);
&input = &File.GetString( True);
&File.Close();
Local string &pieces = &piece | &piece | &piece;

/* Note that the result of this message should indicate &pieces is the same as⇒
 &input */
MessageBox(0, "", 0, 0, "&piece = &input: Len(&pieces)=" | Len(&pieces) | " Len⇒
(&input)=" | Len(&input) | " Same? " | (&pieces = &input));
```

See Also

Chapter 16, "File Class," ReadLine, page 537, Chapter 16, "File Class," WriteLine, page 545 and Chapter 16, "File Class," WriteRaw, page 546.

Open

Syntax

Open(*filespec*, *mode* [, *charset*] [, *pathtype*])

Description

The Open method associates the file object with an external file for input or output.

You can use the GetFile function to access an external file, but each execution of GetFile instantiates a new file object. If you plan to access only one file at a time, you need only one file object. Use GetFile to instantiate a file object for the first external file you access, then use Open to associate the same file object with as many different external files as you want. You need to instantiate multiple file objects with GetFile only if you expect to have multiple files open at the same time.

If the file object is currently associated with a file, that file is first closed.

Working With Relative Paths

If you specify a relative path, that path is appended to the path specified by an environmental variable, if the variable exists, in a particular order. This means if the first environmental variable in order is found, that is the one that's used. If it isn't found, the next one is searched for, and if found, is used, and so on.

If Open is called from Application Engine when running standalone, that is, when not within an application server domain), the following environmental variables are valid, and are searched for in the following order:

1. PS_FILEDIR[\relative_path]
2. PS_SERVDIR\Files[\relative_path]

If the code is *not* running on an application server, the following environmental variables are valid, and are searched for in the following order:

1. PS_FILEDIR[\relative_path]
2. PS_SERVDIR\Files[\relative_path]
3. TEMP[\relative_path]

 Copyright © 1988-2007, Oracle. All rights reserved.

The PS_FILEDIR environmental variable is *not* initialized and set to a path automatically. If you want to use this environmental variable, you must set it yourself.

Note. Your system security should verify if a user has the correct permissions before allowing access to a drive (for example, if a user changes their TEMP environmental variable to a network drive they don't normally have access to, your system security should detect this.)

Parameters

Parameter	Description
filespec	Specify the name, and optionally, the path, of the file you want to open.
mode	A string indicating the manner in which you want to access the file. The mode can be one of the following: "R" (Read mode): opens the file for reading, starting at the beginning. "W" (Write mode): opens the file for writing. ***Warning!*** When you specify Write mode, any existing content in the file is discarded and is overwritten. "A" (Append mode): opens the file for writing, starting at the end. Any existing content is retained. "U" (Update mode): opens the file for reading or writing, starting at the beginning of the file. Any existing content is retained. Use this mode and the GetPosition and SetPosition methods to maintain checkpoints of the current read/write position in the file. In Update mode, any write operation clears the file of all data that follows the position you set. **Note.** Currently, the effect of the Update mode and the GetPosition and SetPosition methods is not well defined for Unicode files. Use the Update mode only on files stored with a non-Unicode character set. "E" (Conditional "exist" read mode): opens the file for reading only if it exists, starting at the beginning. If it doesn't exist, Open has no effect. Before attempting to read from the file, use the IsOpen property to confirm that it's open. "N" (Conditional "new" write mode): opens the file for writing, only if it doesn't already exist. If a file by the same name already exists, Open has no effect. Before attempting to write to the file, use the IsOpen property to confirm that it's open. You can insert an asterisk (*) in the filename to ensure that a new file is created. The system replaces the asterisk with numbers starting at 1 and incrementing by 1, and checks for the existence of a file by each resulting name in turn. It uses the first name for which a file doesn't exist. In this way you can generate a set of automatically numbered files. If you insert more than one asterisk, all but the first one are discarded.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameter	Description
charset	A string indicating the character set you expect when you read the file, or the character set you want to use when you write to the file. You can abbreviate Unicode UTF-16 to "U" and the host operating system's default non-Unicode (sometimes referred to as the ANSI character set) to "A". All other character sets must be spelled out in full, for example, ASCII, THAI, or UTF8. If "A" is specified as the character set, or you don't specify a character set, the character set used is dependent on the application server configuration. On a Windows application server, the default non-Unicode character set is dependent on the Windows ANSI Codepage (ACP) which can be checked using the DOS command chcp. On a Unix application server, the default non-Unicode character set is specified in the application server configuration file, psappsrv.cfg, and can be modified using PSADMIN. You can also use a record field value to specify the character set (for example, `RECORD.CHARSET`.) A list of supported character set names valid for this argument can be found in *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Global Technology.* See *Enterprise PeopleTools 8.49 PeopleBook: Global Technology*, "Selecting and Configuring Character Sets and Language Input and Output," Character Sets in the PeopleSoft Pure Internet Architecture. **Note.** If you attempt to read data from a file using a different character set than was used to write that data to the file, the methods used generate a runtime error or the data returned is unusable. When a file is opened for reading using the "U" *charset* argument, GetFile expects the file to begin with a Unicode Byte Order Mark (BOM), otherwise known as a *sentinal*. This mark indicates whether the file is written in big endian order or little endian order. A BOM consisting of the hex value 0xFEFF indicates a big endian file, a BOM consisting of the hex value 0xFFEF indicates a little endian file. If the Unicode UTF-16 file being opened does not start with a BOM, an error is returned. The BOM is automatically stripped from the file when it is read into the buffers by GetFile. When a file is opened for writing using the "U" *charset* argument, the Unicode BOM is automatically written to the start of the file, dependent on whether the application server hardware platform operates in little endian or big endian mode. Byte Order Marks are not expected or supported for files in other character sets, including UTF8. **Note.** When working with XML documents, specify UTF8 for *charset*. If you are writing an XML file using a different charset, you must remember to include a charset declaration in the XML file.
pathtype	If you have prepended a path to the file name, use this parameter to specify whether the path is an absolute or relative path. The valid values for this parameter are: • %FilePath_Relative (default) • %FilePath_Absolute If you don't specify *pathtype* the default is %FilePath_Relative. If the path is an absolute path, whatever path you specify is used verbatim. You must specify a drive letter and the complete path. You can't use any wildcards when specifying a path. The Component Processor automatically converts platform-specific separator characters to the appropriate form for where your PeopleCode program is executing. On a WIN32 system, UNIX "/" separators are converted to "\", and on a UNIX system, WIN32 "\" separators are converted to "/".

Note. The syntax of the file path does *not* depend on the file system of the platform where the file is actually stored; it depends only on the platform where your PeopleCode is executing.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

The following example opens an existing Unicode file for reading:

```
&MYFILE.Open(&SOMENAME, "E", "U");
If &MYFILE.IsOpen Then
   while &MYFILE.ReadLine(&SOMESTRING)
      /* Process the contents of each &SOMESTRING */
   End-While;
   &MYFILE.Close();
End-If;
```

The following example opens a numbered file for writing in ANSI format, without overwriting any existing files:

```
&MYFILE.Open("C:\temp\item*.txt", "N", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   &MYFILE.WriteLine("Some text.");
   &MYFILE.Close();
End-If;
```

See Also

Chapter 16, "File Class," IsOpen, page 553, Chapter 16, "File Class," SetPosition, page 543, Chapter 16, "File Class," GetPosition, page 532 and Chapter 16, "File Class," Close, page 530.

Chapter 16, "File Class," File Access Interruption Recovery, page 525

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

ReadLine

Syntax

ReadLine(*string*)

Description

The ReadLine method reads one line of text from the external file. The line includes the newline character, but ReadLine strips out the newline character and inserts the result into the string variable *string*.

When ReadLine is executed, it moves the starting point for the next read operation to the end of the text just retrieved, so each line in the file can be read in turn by subsequent ReadLine operations. When no more data remains to be read from the file, ReadLine returns False, and clears the string variable of any content.

Parameters

Parameter	Description
string	A string variable that receives the input text.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the method succeeds, False otherwise. The return value is *not* optional, it is required.

Example

The following example reads a file called &MYFILE and puts each line as a separate element in an array.

```
Local File &MYFILE;
Local array of string &MYARRAY;
Local string &TEXT;

&MYFILE = GetFile("names.txt", "R");
&MYARRAY = CreateArrayRept("", 0);
While &MYFILE.ReadLine(&TEXT);
   &MYARRAY.Push(&TEXT);
End-While;
&MYFILE.Close();
```

See Also

Chapter 16, "File Class," GetString, page 533, Chapter 16, "File Class," ReadRowset, page 538, Chapter 16, "File Class," WriteString, page 550 and Chapter 16, "File Class," WriteLine, page 545.

ReadRowset

Syntax

```
ReadRowset()
```

Description

The ReadRowset method is a file layout method. It instantiates a PeopleCode rowset object based on the file layout definition, then populates the rowset with one transaction from the file. A transaction is considered to be one instance of level zero data contained in a file record, plus all of its subordinate data. If you put more than one segment at level zero, each segment is read in sequence.

Note. You must specify a file layout using the SetFileLayout method before using ReadRowset, otherwise it will return NULL.

When ReadRowset is executed, it moves the starting point for the next read operation to the beginning of the next rowset, so each transaction in the file can be read in turn by subsequent ReadRowset operations. When no more data remains to be read from the file, ReadRowset returns NULL.

If you're using the SetFileId method with ReadRowset to process an input file based on multiple file layouts, ReadRowset returns NULL when it reads a FileId file record (line) between the rowsets.

When ReadRowset returns NULL, you can use the IsFileId property to determine if you've reached the end of the file or a FileId record.

Note. When using ReadRowset, if a value in the file exceeds the defined length in the file layout, it is ignored. The given record field is flagged with an edit error which can be programmatically checked.

If the ReadRowset encounters a line in the file containing the FileId, and the lines following this are *not* a new rowset, the process considers it to be an invalid FileId. You can specify whether to ignore the invalid record or terminate the PeopleCode with the IgnoreInvalidId property.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. If you're using the SetFileId method with ReadRowset to process an input file based on multiple layouts, FileId file records between the rowsets are considered to be valid file records, and won't generate any errors, regardless of the state of the IgnoreInvalidId property.

See Chapter 16, "File Class," SetFileLayout, page 542.

Considerations for Using Dates With ReadRowset

Single digits in dates in the form MMDDYY or MMDDYYYY must be padded with zeros. That is, if the date in your data is February 3, 2000, the form must be:

```
02/03/2000
```

or

```
02/03/00
```

The following is *not* valid.

```
2/3/00
```

Considerations Using XML With ReadRowset

If all of the fields in the file layout are not present in the XML, no data is written to the database. If there are additional tags in the XML, they are ignored.

Considerations Using Nested Data with ReadRowset

If you are processing input files with a large amount of nested data, your application server may run out of memory before the system finishes processing all of the data.

This may happen because of the differences between processing a single-level rowset and a multi-level rowset. If you are reading the rows of a single-level (level 0) rowset from a file, the file is processed one row at a time, that is, only one row resides in memory at a time.

However, if you are reading the rows of a multi-level (parent-child) rowset from a file, then for each level 0 row, *all* of the associated child rows (and all of their associated child rows) simultaneously reside in memory. As a result, during the processing a large input data file that is associated with a multi-level rowset (through a multi-level file layout definition), your application server may run out of memory.

To work around this, consider doing one of the following:

- Retain your original file layout definition and split the original input file into smaller (but structurally unchanged) pieces
- Flatten out your original file layout definition (that is, split it up into several single-level definitions) as well as split the original input file into several single-level pieces.

Parameters

None.

Returns

A populated rowset.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example reads and processes an entire file. The data in the file is based on a single File layout definition:

```
&MYFILE.GetFile(&SOMENAME, "R");
If &MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT) then
   &SOMEROWSET = &MYFILE.ReadRowset();
   While &SOMEROWSET <> NULL
      /* Process the contents of each &SOMEROWSET */
      &SOMEROWSET = &MYFILE.ReadRowset();
   End-While;
End-If;
&MYFILE.Close();
```

See Also

Chapter 16, "File Class," CurrentRecord, page 551, Chapter 16, "File Class," IgnoreInvalidId, page 551, Chapter 16, "File Class," GetString, page 533, Chapter 16, "File Class," ReadLine, page 537, Chapter 16, "File Class," SetFileLayout, page 542, Chapter 16, "File Class," WriteRecord, page 546 and Chapter 16, "File Class," WriteRowset, page 548.

Chapter 16, "File Class," Multiple File Layouts, page 569

Chapter 16, "File Class," ReadRowset Example, page 565

Chapter 33, "Rowset Class," IsEditError, page 1449

SetFileId

Syntax

SetFileId(*fileid*, *position*)

Description

The SetFileId method is a file layout method. If your input file contains data based on more than one File Layout definition, you must use this method in combination with the CurrentRecord and IsNewFileId properties and the ReadRowset method to process the file correctly.

Note. SetFileId, CurrentRecord and IsNewFileId don't apply to CSV and XML format input files. You can use only FIXED format files to implement multiple file layouts.

See Chapter 16, "File Class," CurrentRecord, page 551; Chapter 16, "File Class," IsNewFileId, page 552 and Chapter 16, "File Class," ReadRowset, page 538.

At each point in the input file where the structure of the rowset changes, there must be an extra line in the file containing the file record (line), the FileId file record (line), that signifies that the change. Each FileId file record must have the following components:

- A value that designates it as a Fileid. Only one FileId value is necessary throughout the file, such as "999".
- A name field that identifies the file layout needed for the rowset that follows. This field could contain the name of the file layout as it's defined in PeopleSoft Application Designer.

Copyright © 1988-2007, Oracle. All rights reserved.

These lines containing the FileId are *not* part of any rowset. They can contain other information, which will be disregarded by the system. The FileId identifier and the file layout names aren't automatically stored anywhere; they exist only in the input file's FileId file records and in the PeopleCode.

To process an input file that requires multiple file layouts:

1. Use SetFileLayout to specify the file layout definition to use.

   If you're specifying the initial file layout for this file, it doesn't have to be the correct one. You can determine the correct file layout to use for the first rowset during a subsequent step.

2. Use SetFileId to specify the value of the FileId field, *fileid*, that's used throughout the file to designate FileId file records.

   Note. After each execution of SetFileLayout, the SetFileId setting is disabled. Be sure to re-specify the FileId value if you expect the file layout to change, so the system continues looking for the next FileId file record.

3. Use ReadRowset to read the next rowset from the file.

   The current file record is also stored in the CurrentRecord property as a string. The PeopleCode process determines whether it's the beginning of a new rowset, or a FileId file record according to the *fileid* you specified. If it's a FileId file record, the process sets the IsNewFileId property to True; if not, it sets the IsNewFileId property to False.

   Note. If this is the first execution of ReadRowset on the file, it returns NULL upon encountering the initial FileId file record, but still stores that file record in CurrentRecord and sets IsNewFileId accordingly.

4. If the rowset returned isn't NULL, process that rowset as necessary.
5. If IsNewFileId is False, go back to step 3 to continue reading rowsets from the file. If IsNewFileId is True, examine CurrentRecord to determine which new file layout to use, and go back to step 1.

You can repeat this procedure one rowset at a time, proceeding through the remainder of the input file. When ReadRowset returns NULL, no more rowset data is available.

Parameters

Parameter	Description
fileid	Specify the value of the FileId field used in the current file.
position	Specify the column in the FileId file record where the FileId field starts. The default is 1.

Returns

None.

Example

```
&rsFile = &MYFILE.ReadRowset();
&CurrentRecord = &MYFILE.CurrentRecord;
&IsNewFileID = &MYFILE.IsNewFileId;
If &MYFILE.IsNewFileId Then
   &FILELAYOUT = FindFileID(&CurrentRecord);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   &MYFILE.SetFileLayout(@("FileLayout." | &FILELAYOUT));
   &MYFILE.SetFileId("999", 1);
End-If;
```

See Also

Chapter 16, "File Class," CurrentRecord, page 551, Chapter 16, "File Class," IsNewFileId, page 552, Chapter 16, "File Class," ReadRowset, page 538 and Chapter 16, "File Class," SetFileLayout, page 542.

Chapter 16, "File Class," File Layout Examples, page 557

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Constructing File Layouts and Performing Data Interchanges"

SetFileLayout

Syntax

SetFileLayout(FILELAYOUT.*filelayoutname*)

Description

The SetFileLayout method is a file layout method. It associates a specific file layout definition with the file object executing this method, providing easy access to rowset data. The file object must be associated with an open file. With file layout definitions, you can read and write rowsets as easily as strings. If the file object isn't open or the definition doesn't exist, SetFileLayout fails.

You must execute SetFileLayout before you can use any other file layout methods or properties with a file object, otherwise those methods and properties return NULL or False.

Note. All provided PeopleTools records that have a prefix of PSFLD are related to file layout definitions.

Parameters

Parameter	Description
filelayoutname	Specify as a string the name of an existing file layout definition created in PeopleSoft Application Designer.

Returns

A Boolean value: True if the method succeeds, False otherwise.

Example

The following example opens a data file, associates a file layout definition with it, reads and processes the first rowset from it, and closes the file.

```
&MYFILE.Open(&SOMENAME, "E");
If &MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT) then
   &SOMEROWSET = &MYFILE.ReadRowset();
   /* Process the contents of &SOMEROWSET */
Else
   /* Error - SetFileLayout failed */
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYFILE.Close();
```

See Also

Chapter 16, "File Class," CurrentRecord, page 551, Chapter 16, "File Class," CreateRowset, page 530, Chapter 16, "File Class," ReadRowset, page 538, Chapter 16, "File Class," IgnoreInvalidId, page 551 and Chapter 16, "File Class," WriteRowset, page 548.

Chapter 16, "File Class," File Layout Examples, page 557

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Constructing File Layouts and Performing Data Interchanges"

SetPosition

Syntax

SetPosition(*position*)

Description

The SetPosition method sets the current read or write position in the external file associated with the file object executing this method. SetPosition works only with a file, which is opened in Update mode, and is designed to be used in combination with the GetPosition method to recover from interruptions during file access.

To start reading or writing from the beginning of a file, you must use SetPosition to set a read/write position of 0, immediately after you open the file in Update mode.

Parameters

Parameter	Description
position	The byte position in the file at which you want to continue reading or writing. This should either be 0 or a value you previously obtained with the GetPosition method and saved in a separate file or a database.

Warning! In Update mode, any write operation clears the file of all data that follows the position you set.

Note. Use SetPosition only for recovering from file access interruptions. Supplying your own value for SetPosition isn't recommended, except for position 0.

Returns

None.

Example

The following example reopens a file in Update mode, and sets the read position to the last saved checkpoint:

```
&MYFILE = GetFile(&SOMENAME, "U");
If &MYFILE.IsOpen Then
   /* Retrieve the value of the last saved checkpoint, &LASTPOS */
   &MYFILE.SetPosition(&LASTPOS);
   while &MYFILE.ReadLine(&SOMESTRING)
      /* Process the contents of each &SOMESTRING */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   End-While;
   &MYFILE.Close();
End-If;
```

Note. Currently, the effect of the Update mode and the GetPosition and SetPosition methods is not well defined for Unicode files. Use the Update mode only on files stored with a non-Unicode character set.

See Also

Chapter 16, "File Class," GetPosition, page 532 and Chapter 16, "File Class," Open, page 534.

Chapter 16, "File Class," File Access Interruption Recovery, page 525

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetFile

SetRecTerminator

Syntax

SetRecTerminator(*Terminator*)

Description

The SetRecTerminator method is a file layout method.

Use this method to specify the string of characters that indicate the end of a file record. Read operations use the value of SetRecTerminator to determine where each file record ends and the next one starts, and write operations append the value of SetRecTerminator to each record written.

This value defaults to the newline character appropriate to the platform where the file is being stored:

- a linefeed on UNIX systems
- a carriage return/linefeed combination on Windows systems

You need to specify a different record terminators only if you anticipate that part of the data in a file field that includes the newline character used on the storage platform; you must assign the record terminator a unique string that you know does not appear in a file field.

If you set the record terminator to Null ("") you eliminate linefeeds or carriage returns.

Parameters

Parameter	Description
Terminator	Specify the value you want used for the record terminator.

Returns

None.

Example

```
&File = GetFile(......);

if &File.IsOpen then
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/* set the terminator to be NULL */
Local String &Term = "";
&File.SetRecTerminator(&Term);
```

WriteLine

Syntax

WriteLine(*string*)

Description

The WriteLine method writes one string of text, *string*, to the output file associated with the file object executing this method, followed by a newline character appropriate to the platform where the file is being written. To build a single line using multiple strings, use the WriteString method.

Parameters

Parameter	Description
string	The string of text to be written.

Returns

None.

Example

The following example adds a line of text to an existing file:

```
&MYFILE.Open("somefile.txt", "A");
&MYFILE.WriteLine("This is the last line in the file.");
&MYFILE.Close();
```

The following example converts a file where the fields are separated with tabs into a file where the fields are separated with commas.

```
Local File &TABFILE, &CSVFILE;
Local string &FILE_NAME, &DATA, &NEWDATA;

&FILE_NAME = "Test.txt";
&TAB = Char(9);

&TABFILE = GetFile(&FILE_NAME, "r");
&FileName = &TABFILE.Name;
&POS = Find(".", &FileName);
&NEWFILE_NAME = Substring(&FileName, 1, &POS) | "dat";
&CSVFILE = GetFile(&NEWFILE_NAME, "N", %FilePath_Absolute);
If &TABFILE.IsOpen And
      &CSVFILE.IsOpen Then
   While &TABFILE.ReadLine(&DATA);
      &NEWDATA = Substitute(&DATA, &TAB, ",");
      &CSVFILE.WriteLine(&NEWDATA);
   End-While;
   &TABFILE.Close();
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      &CSVFILE.Close();
   End-If;
```

See Also

Chapter 16, "File Class," GetString, page 533, Chapter 16, "File Class," ReadLine, page 537, Chapter 16, "File Class," WriteString, page 550 and Chapter 16, "File Class," WriteRaw, page 546.

WriteRaw

Syntax

WriteRaw(*RawBinary*)

Description

The WriteRaw method writes the contents of *RawBinary* to a file. This can be used for writing images, messages, or other types of raw binary data to a file.

Parameters

Parameter	Description
RawBinary	Specify the raw binary to be written to the file.

Returns

None.

Example

The following example writes employee photos (GIF files) from a record to a file.

```
Local File &FILE;
Local Record &REC;
Local SQL &SQL;

&REC = CreateRecord(Record.EMPL_PHOTO);
&SQL = CreateSQL("%SelectAll(:1)", Record.EMPL_PHOTO);

&FILE = GetFile("C:\temp\EMPL_PHOTO.GIF", "w", "a", %FilePath_Absolute);

While &SQL1.Fetch(&REC)
   &FILE.WriteRaw(&REC.EMPLOYEE_PHOTO.Value);
End-While;
&FILE.Close();
```

See Also

Chapter 16, "File Class," WriteLine, page 545 and Chapter 16, "File Class," WriteString, page 550.

WriteRecord

Syntax

WriteRecord(*record*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The WriteRecord method is a file layout method. It writes the contents of the record object *record*, to the output file. (Remember, a record object contains only one row of data from an SQL table.)

You can use this method to build a transaction in the output file, one file record at a time, without having to instantiate and populate a rowset object. You can apply this method to any record whose structure and name matches that of a record defined in the current file layout.

Note. You must execute the SetFileId method from the file object before using WriteRecord, otherwise it returns False.

See Chapter 16, "File Class," SetFileLayout, page 542.

Note. When you're writing text to an XML file, special HTML characters, such as ampersands, lesser than or greater than symbols, and so on, are automatically converted to valid XML character string, such as &amp.

Considerations Using XML With File Definition Tags

File Definition Tags have no effect when importing data. However, generally, during export, the File Definition Tag is added at the start and end of the data to create a valid XML file.

The one exception is when the file to which the data is being written during export has been opened in append mode. At that time, the file definition tag is not taken into consideration.

Considerations Using XML with File Definitions

If your file layout is defined as XML, WriteRecord doesn't add the closing tag for the record. You must write it yourself using the WriteLine method. This is because the code has no way of knowing when you want to write children records following the record just written out.

The following code shows an example of using WriteLine:

```
Local File &MYFILE;

&MYFILE = GetFile("XMLrecord.txt", "A");

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.RECORDLAYOUT) Then
      &LN = CreateRecord(RECORD.QA_INVEST_LN);
      &SQL2 = CreateSQL("%Selectall(:1)", &LN);
      While &SQL2.Fetch(&LN)
         WriteRecord(&LN);
          WriteLine("</QA_INVEST_LN>"); /* Add the closing tag */
      End-While;
   Else
   /* do error processing - filelayout not correct */
   End-If;
Else
/* do error processing - file not open */
End-If;

&MYFILE.Close();
```

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
record	Specify the name of an existing record object to be written. You can use Rowset class methods such as GetField and built-in functions such as GetRecord to populate the record with data before writing it to the file.

Returns

A Boolean value: True if the method succeeds, False otherwise. This value is optional.

Example

The following example appends all the data in a record to an existing file:

```
Local File &MYFILE;

&MYFILE = GetFile("record.txt", "A");

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.VOL_TEST) Then
      &LN = CreateRecord(RECORD.VOLNTER_ORG_TBL);
      &SQL2 = CreateSQL("%Selectall(:1)", &LN);
      While &SQL2.Fetch(&LN)
         &MYFILE.WriteRecord(&LN);
      End-While;
   Else
      /* do error processing - filelayout not correct */
   End-If;
Else
   /* do error processing -; file not open */
End-If;

&MYFILE.Close();
```

See Also

Chapter 16, "File Class," CreateRowset, page 530, Chapter 16, "File Class," ReadRowset, page 538, Chapter 16, "File Class," SetFileLayout, page 542 and Chapter 16, "File Class," WriteRowset, page 548.

Chapter 31, "Record Class," page 1377

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

WriteRowset

Syntax

WriteRowset(*rowset* [, *Write_Data*])

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The WriteRowset method is a file layout method. It writes the contents of a rowset object, *rowset*, to the output file associated with the file object executing this method. Regardless of whether the rowset contains just one or more than one transaction (level zero row), executing this method once writes the entire contents of the rowset to the output file.

Note. You must execute the SetFileLayout method from the file object before using WriteRowset, otherwise it returns False.

See Chapter 16, "File Class," SetFileLayout, page 542.

WriteRowset writes a rowset to a file only if the data in the component buffer has changed. You must specify the *WriteData* parameter as True if you want the WriteRowset method to force the rowset to be written to the file even if the buffer has not been changed.

Note. When you're writing text to an XML file, special HTML characters, such as ampersands, lesser than or greater than symbols, and so on, are automatically converted to valid XML character string, such as &amp.

Considerations Using Fixed Length Files With Numeric Fields

All numeric fields are right-justified in when writing to fixed length files. In addition, zeros are padded to the right after the decimal point if required.

Considerations Using XML With File Definition Tags

File Definition Tags have no effect when importing data. However, generally, during export, the File Definition Tag is added at the starting and end of the data to create a valid XML file.

The one exception is when the file to which the data is being written during export has been opened in append mode. At that time, the file definition tag is not taken into consideration.

Parameters

Parameter	Description
rowset	Specify the name of an existing rowset object that was instantiated with the File class CreateRowset or ReadRowset method. You can use Rowset class methods such as Select and built-in functions such as GetLevel0 to populate the rowset with data before writing it to the file.
WriteData	Specify whether to write the rowset data to the file whether or not the data in the buffer has changed. This parameter takes a Boolean value: True, write the data to the buffer regardless, False, only write the data if changed. The default value for this parameter is False.

Returns

A Boolean value: True if the method succeeds, False otherwise.

Example

```
Local File &MYFILE;
Local Rowset &FILEROWSET;

&MYFILE = GetFile("c:\temp\EMP_CKLS.txt", "A", %FilePath_Absolute);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.EMPL_CHECKLIST) Then
      &FILEROWSET = &MYFILE.CreateRowset();
      &FILEROWSET = GetLevel0();
      &MYFILE.WriteRowset(&FILEROWSET, True);
   Else
      /* file layout not found, do error processing */
   End-If;
Else
   /* file not opened, do error processing */
End-if;

&MYFILE.Close();
```

See Also

Chapter 16, "File Class," CreateRowset, page 530, Chapter 16, "File Class," ReadRowset, page 538, Chapter 16, "File Class," SetFileLayout, page 542 and Chapter 16, "File Class," WriteRecord, page 546.

Chapter 31, "Record Class," page 1377

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

WriteString

Syntax

WriteString(*string*)

Description

The WriteString method writes one string of text to the output file associated with the file object executing this method, without any newline character. Each string written extends the current line in the file.

You can start a new line by using the WriteLine method to write the last part of the current line. WriteLine always adds a newline character appropriate to the platform where the file is being written, whether you supply a character string of any length, or a null string.

Parameters

Parameter	Description
string	A string variable containing the text to be written.

Returns

None.

Example

The following example opens an empty file, writes two lines of text to it without a final newline, and closes it:

```
&MYFILE.Open("somefile.txt", "W");
&MYFILE.WriteString("This is the first ");
&MYFILE.WriteString("line in the file.");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYFILE.WriteLine("");
&MYFILE.WriteString("This second line is not terminated.");
&MYFILE.Close();
```

See Also

Chapter 16, "File Class," ReadLine, page 537, Chapter 16, "File Class," WriteLine, page 545 and Chapter 16, "File Class," WriteRaw, page 546.

File Class Properties

In this section, we discuss each File Class property.

CurrentRecord

Description

This property is a file layout property. This property returns the current record as a string. The CurrentRecord property is used in combination with the SetFileId and ReadRowset methods and IsNewFileId property when reading files that contain data based on multiple file layouts.

If the ReadRowset method returns NULL, either the current record is a FileId record or the end of file has been reached. The IsNewFileId enables you to determine which. If it is a FileId, you can parse the string returned by CurrentRecord to determine the file layout definition to be used.

Note. SetFileId, CurrentRecord, and IsNewFileId don't apply to CSV and XML format input files. You can implement multiple file layouts only with FIXED format files.

This property is read-only.

Example

```
&RECSTRING = &MYFILE.CurrentRecord;
```

See Also

Chapter 16, "File Class," SetFileId, page 540, Chapter 16, "File Class," ReadRowset, page 538 and Chapter 16, "File Class," IsNewFileId, page 552.

Chapter 16, "File Class," Multiple File Layouts, page 569

IgnoreInvalidId

Description

This property is a file layout property that's used in combination with the ReadRowset method. It returns a Boolean value that specifies whether file records with invalid FileIds are ignored. Each time ReadRowset is executed, it may encounter a file record that doesn't qualify as part of the rowset because its File ID isn't part of the current file layout, or because it occurs in an invalid position in the rowset. If IgnoreInvalidId is False, the PeopleCode program terminates; if IgnoreInvalidId is True, ReadRowset ignores the invalid file record. The default value is True.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MYFILE.IgnoreInvalidId = False;
```

See Also

Chapter 16, “File Class,” ReadRowset, page 538

IsError

Description

This property is a file layout property. It returns a Boolean value indicating whether a field error condition was generated by the last file layout method executed. If an error condition was generated, IsError returns True; if not, IsError returns False. The default value is False.

This property is read-only.

Example

The following example shows where IsError would be used:

```
&MYFILE.Open(&SOMENAME, "R");
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT);
&MYROWSET = &MYFILE.ReadRowset();
If &MYFILE.IsError Then
   /* Examine the EditError property of each field in the rowset
      to find the one with the error, and respond accordingly */
End-If;
&MYFILE.Close();
```

See Also

Chapter 16, “File Class,” Multiple File Layouts, page 569

IsNewFileId

Description

This property is a file layout property. It returns a Boolean value indicating whether a FileId file record has been encountered. When the ReadRowset method reads a transaction from an input file, it examines the current file record following the transaction. If that file record is a FileId file record, IsNewFileId returns True; if not, IsNewFileId returns False.

IsNewFileId is used in combination with the SetFileId method and CurrentRecord property when reading files that require multiple file layouts.

Note. SetFileId, CurrentRecord, and IsNewFileId don’t apply to CSV and XML format input files. You can use only FIXED format files to implement multiple file layouts.

This property is read-only.

Example

```
&MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT) then  /* Set the first layout */
&MYFILE.SetFileId("999", 1);  /* Set the FileId */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&SOMEROWSET = &MYFILE.ReadRowset();  /* Read the first rowset */
While &SOMEROWSET <> NULL
   If &MYFILE.IsNewFileId Then
      /* Examine &MYFILE.CurrentRecord for the next layout */
      /* Set the next layout */
      &MYFILE.SetFileId("999", 1);  /* Set the FileId */
   End-If;
   /* Process the current &SOMEROWSET */
   &SOMEROWSET = &MYFILE.ReadRowset();  /* Read the next rowset */
End-While;
```

See Also

Chapter 16, “File Class,” ReadRowset, page 538, Chapter 16, “File Class,” SetFileId, page 540 and Chapter 16, “File Class,” CurrentRecord, page 551.

IsOpen

Description

This property returns a Boolean value indicating whether the file is open. If the file object is open, IsOpen returns True; if not, IsOpen returns False.

This property is read-only.

Example

The following example opens a file, writes a line to it, and closes it:

```
&MYFILE.Open("item.txt", "W");
If &MYFILE.IsOpen Then
   &MYFILE.WriteLine("Some text.");
   &MYFILE.Close();
End-If;
```

Name

Description

This property returns as a string the name of the external file. If no file is currently associated with the file object, Name returns a NULL string.

This property is read-only.

Example

```
&TMP = &MYFILE.Name;
```

Copyright © 1988-2007, Oracle. All rights reserved.

UseSpaceForNull

Description

Use the UseSpaceForNull property to specify whether the WriteRowset method writes <qualifier> <qualifier> (<qualifier>-space-<qualifier>) for all the Null or empty character fields of a CSV-type file layout, or if the method writes <qualifier><qualifier> (<qualifier>-<qualifier>).

This property takes a boolean value: false if WriteRowset writes <qualifier>-<qualifier>, true if WriteRowset writes <qualifier>-space-<qualifier>. The default is False.

Note. The state of this property has no effect on the behavior of the ReadRowset method. It also has no effect when the associated file layout is of type fixed or XML.

This property is read-write.

Example

```
Local File &LOGFILE;
Local File &FILE1;
Local File &FILE2;
Local Record &REC1;
Local SQL &SQL1;
Local Rowset &RS1;
Local Row &ROW1;

&LOGFILE = GetFile("C:\Temp\currency.log", "W", "A", %FilePath_Absolute);
If (&LOGFILE = Null) Then
   Exit;
End-If;
If Not (&LOGFILE.SetFileLayout(FileLayout.CURRENCY_FL)) Then
   &LOGFILE.WriteLine("SetFileLayout() on LOGFILE failed.");
End-If;

&FILE1 = GetFile("C:\Temp\currency_nonspaced.csv", "W", "A", %FilePath_Absolute);
If (&FILE1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  GetFile() on non-spaced output file failed.");
   &LOGFILE.Close();
   Exit;
End-If;
If Not (&FILE1.SetFileLayout(FileLayout.CURRENCY_FL)) Then
   &LOGFILE.WriteLine("FATAL ERROR:  SetFileLayout() on non-spaced output file⇒
 failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   Exit;
End-If;

&FILE2 = GetFile("C:\Temp\currency_spaced.csv", "W", "A", %FilePath_Absolute);
If (&FILE2 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  GetFile() on spaced output file failed.");
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
   &LOGFILE.Close();
   &FILE1.Close();
   Exit;
End-If;
If Not (&FILE2.SetFileLayout(FileLayout.CURRENCY_FL)) Then
   &LOGFILE.WriteLine("FATAL ERROR:  SetFileLayout() on spaced output file⇒
 failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   &FILE2.Close();
   Exit;
End-If;

&REC1 = CreateRecord(Record.CURRENCY_CD_TBL);
If (&REC1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  CreateRecord() on record failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   &FILE2.Close();
   Exit;
End-If;

&RS1 = CreateRowset(Record.CURRENCY_CD_TBL);
If (&RS1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  CreateRowset() on record failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   &FILE2.Close();
   Exit;
End-If;

&SQL1 = CreateSQL("%Selectall(:1)", &REC1);
If (&SQL1 = Null) Then
   &LOGFILE.WriteLine("FATAL ERROR:  CreateSQL() failed.");
   &LOGFILE.Close();
   &FILE1.Close();
   &FILE2.Close();
   Exit;
End-If;

If (&FILE1.UseSpaceForNull) Then
   &LOGFILE.WriteLine("UseSpaceForNull is True on non-spaced output, by default.");
Else
   &LOGFILE.WriteLine("UseSpaceForNull is False on non-spaced output, by⇒
 default.");
End-If;
If (&FILE2.UseSpaceForNull) Then
   &LOGFILE.WriteLine("UseSpaceForNull is True on spaced output, by default.");
Else
   &LOGFILE.WriteLine("UseSpaceForNull is False on spaced output, by default.");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
End-If;

&FILE1.UseSpaceForNull = True;
If (&FILE1.UseSpaceForNull) Then
   &LOGFILE.WriteLine("Setting UseSpaceForNull to True succeeded on non-spaced⇒
 output.");
Else
   &LOGFILE.WriteLine("Setting UseSpaceForNull to True failed on non-spaced⇒
 output.");
End-If;
&FILE2.UseSpaceForNull = True;
If (&FILE2.UseSpaceForNull) Then
   &LOGFILE.WriteLine("Setting UseSpaceForNull to True succeeded on spaced⇒
 output.");
Else
   &LOGFILE.WriteLine("Setting UseSpaceForNull to True failed on spaced output.");
End-If;

&FILE1.UseSpaceForNull = False;
If (&FILE1.UseSpaceForNull) Then
   &LOGFILE.WriteLine("Setting UseSpaceForNull to False failed on non-spaced⇒
 output.");
Else
   &LOGFILE.WriteLine("Setting UseSpaceForNull to False succeeded on non-spaced⇒
 output.");
End-If;
&FILE2.UseSpaceForNull = False;
If (&FILE2.UseSpaceForNull) Then
   &LOGFILE.WriteLine("Setting UseSpaceForNull to False failed on spaced output.");
Else
   &LOGFILE.WriteLine("Setting UseSpaceForNull to False succeeded on spaced⇒
 output.");
End-If;

&FILE1.UseSpaceForNull = False;
&FILE2.UseSpaceForNull = True;
&I = 1;
While &SQL1.Fetch(&REC1)
   &ROW1 = &RS1.GetRow(1);
   &REC1.CopyFieldsTo(&ROW1.CURRENCY_CD_TBL);
   &FILE1.WriteRowset(&RS1, True);
   &FILE2.WriteRowset(&RS1, True);
   REM   &LOGFILE.WriteLine("Got row " | String(&I));
   &I = &I + 1;
End-While;

&FILE1.Close();
&FILE2.Close();
&LOGFILE.Close();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 16, “File Class,” WriteRowset, page 548

File Layout Examples

If your data is hierarchical in nature, or based on existing PeopleSoft records or pages, you want to use a File Layout definition for reading and writing your data, rather than doing it line by line (or field by field.)

For example, suppose you wanted to write all the information from a record to a file. You can use the WriteRecord method to write all the data from the record, instead of having to loop through every field, find the value, and write it to the file.

In addition, you could write all the information from a transaction (or several transactions) from a page to a file. Each transaction can be considered a *rowset*. A rowset can contain more than one record and is generally composed in a hierarchical structure. You could create a File Layout definition that has the same structure as the page (or component), and use the WriteRowset method. If you have a file that contains data in the correct format, you can use the ReadRowset method to read the data from the file to the page.

Each file layout is associated with a *format*. This format specifies the type of data in the files. You specify the format as part of the File Layout Properties. You can only specify one format for a file layout. Available formats are:

- FIXED (default)
- CSV
- XML

The file layout methods and properties use this information to handle each file type in a transparent manner. Generally, you don’t need to do anything different based on file type. Any exceptions are noted in the documentation.

Note. Unlike other PeopleTools definitions, records and field are *copied* to a File Layout definition. There are no pointers. This means if you change a record definition (add or remove a field) you must change the File Layout definition also. The changes are not automatically propagated. This is why the documentation refers to these elements as file records, file fields, and so on, to show that they are no longer part of the original definition they were created from.

PeopleSoft recommends regenerating all file layout definitions after any upgrade, to avoid any corruption caused by changes to the database.

See Using Standalone Rowsets for more examples of writing from and reading to files using File Layout and standalone rowsets.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

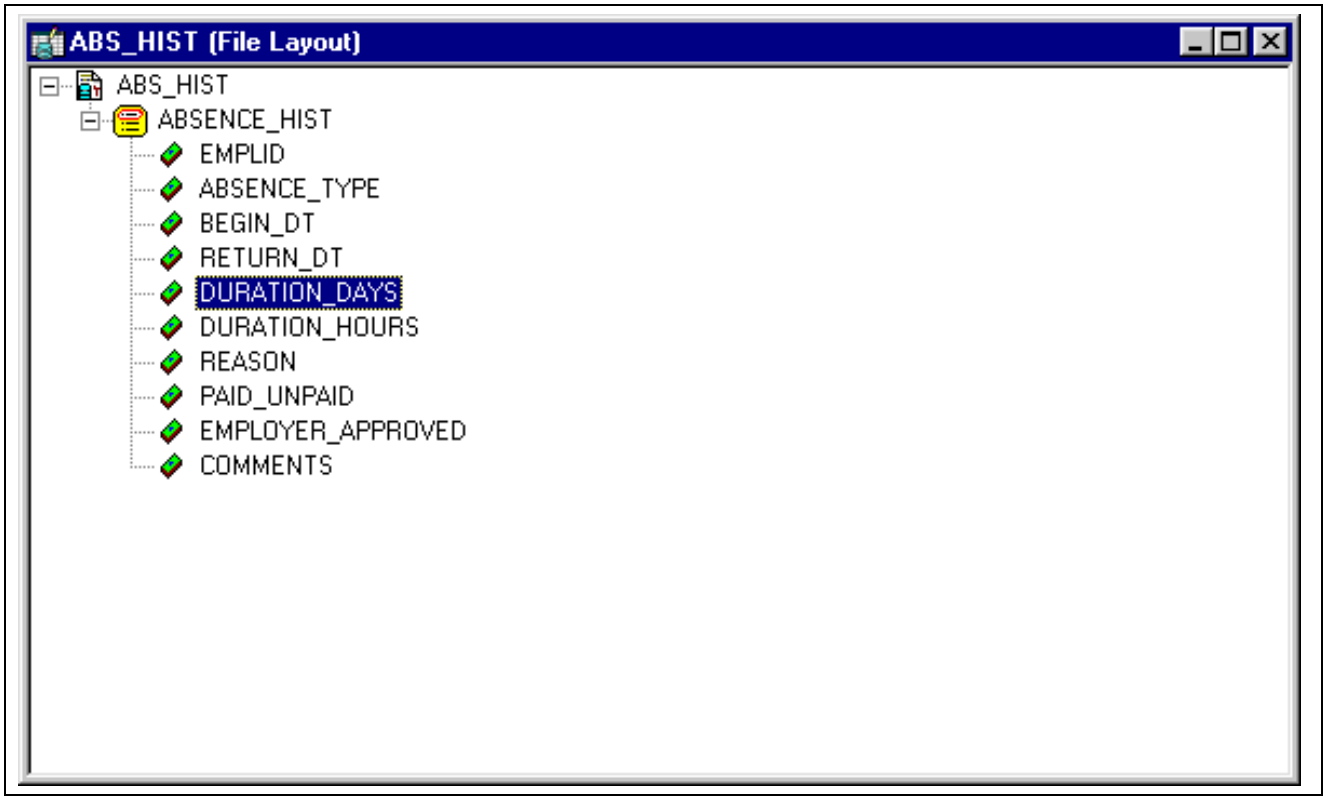
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Constructing File Layouts and Performing Data Interchanges"

WriteRecord Example

In the following example, the File Layout definition is based on the record ABSENCE_HISTORY, and looks like this:



Example File Layout definition (ABS_HIST)

You should note the following about the using the WriteRecord method:

- Not all the fields in the File Layout definition and the record have to match. The WriteRecord method, like all File Layout methods, applies only to the *like-named* fields. If there are additional fields in the record or in the File Layout definition, they are ignored.
- The WriteRecord method writes only to *like-named* records. If you rename a record after you use it to create a File Layout definition, you must rename it to the exact same name in your File Layout. Because WriteRecord writes like-named records, the same file layout definition can contain more than one record.
- The WriteRecord method takes a *record object* as its parameter. A populated record object references a *single row* of data in the SQL table. This is why a SQL Fetch statement is used in a condition around the WriteRecord method. This fetches every row of data from the SQL table, then writes it to the file.

The following code writes the ABSENCE_HIST record to the file record.txt.

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local Record &RecLine;
Local File &MYFILE;
Local SQL &SQL2;

&MYFILE = GetFile("record.txt", "A");

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FileLayout.ABS_HIST) Then
      &RecLine = CreateRecord(RECORD.ABSENCE_HIST);
      &SQL2 = CreateSQL("%Selectall(:1)", &RecLine);
      While &SQL2.Fetch(&RecLine)
         &MYFILE.WriteRecord(&RecLine);
      End-While;
   Else
      /* do error processing -; filelayout not correct */
   End-If;
Else
   /* do error processing -; file not open */
End-If;

&MYFILE.Close();
```

If you wanted to write only changed records to the file, you could add the following code that is set in bold font:

```
While &SQL2.Fetch(&RecLine)
   If &RecLine.IsChanged Then
         &MYFILE.WriteRecord(&RecLine);
   End-If;
End-While;
```

The following is the first part of a sample data file created by the previous code. The field format is FIXED:

```
8001       VAC 09/12/1981 09/26/1981 14  0                                   P Y
8001       VAC 03/02/1983 03/07/1983 5   0                                   P Y
8001       VAC 08/26/1983 09/10/1983 13  0                                   P Y
8105       CNF 02/02/1995 ??/??/     0   0                                   U N
8516       MAT 06/06/1986 08/01/1986 56  0                                   P Y
8516       SCK 08/06/1988 08/07/1988 1   0                                   P Y
8516       VAC 07/14/1987 07/28/1987 14  0                                   P Y
8553       JUR 12/12/1990 12/17/1990 5   0   Local Jury Duty                 P N
8553       MAT 02/20/1992 10/01/1992 224 0   Maternity Leave                 U N
8553       MAT 08/19/1994 03/01/1995 194 0   Maternity-2nd child             U Y
8553       PER 04/15/1993 04/19/1993 4   0                                   U N⇒
 Personal Day required
8553       SCK 01/28/1987 01/30/1987 2   0   Hong Kong Flu                   P N
8553       SCK 08/02/1988 08/03/1988 1   0   Sick                            P N
8553       SCK 09/12/1995 09/13/1995 1   0                                   P N
8641       VAC 06/01/1988 06/15/1988 14  0                                   P Y
8641       VAC 07/01/1989 07/15/1989 14  0                                   P Y
G001       MAT 07/02/1991 09/28/1991 88  0   3-month Maternity leave          P Y⇒
 Maternity will be paid as 80% of Claudia's current salary.
```

Copyright © 1988-2007, Oracle. All rights reserved.

If a record in the File Layout definition has a File Record ID, each line in the file referencing this record is prefaced with this number. The File Record ID is not a field in the data itself. It is also referred to as the *rowid*. File Record IDs can be useful when the file you're producing refers to more than one record.

File Record IDs can be used only with File Layout definitions that have a type of FIXED. If a File Layout definition has only one level then File Record IDs can be omitted. But for File Layout definitions with more than one level, you *must* use File Record IDs.

The following is sample file, produced with the same code, but with a File Record ID of 101 added:

```
101 8001       VAC 09/12/1981 09/26/1981 14  0                              P Y
101 8001       VAC 03/02/1983 03/07/1983 5   0                              P Y
101 8001       VAC 08/26/1983 09/10/1983 13  0                              P Y
101 8105       CNF 02/02/1995 ??/??/     0   0                              U N
101 8516       MAT 06/06/1986 08/01/1986 56  0                              P Y
101 8516       SCK 08/06/1988 08/07/1988 1   0                              P Y
101 8516       VAC 07/14/1987 07/28/1987 14  0                              P Y
101 8553       JUR 12/12/1990 12/17/1990 5   0   Local Jury Duty            P N
101 8553       MAT 02/20/1992 10/01/1992 224 0   Maternity Leave            U N
101 8553       MAT 08/19/1994 03/01/1995 194 0   Maternity-2nd child        U Y
101 8553       PER 04/15/1993 04/19/1993 4   0                              U N⇒
 Personal Day required
101 8553       SCK 01/28/1987 01/30/1987 2   0   Hong Kong Flu              P N
101 8553       SCK 08/02/1988 08/03/1988 1   0   Sick                       P N
101 8553       SCK 09/12/1995 09/13/1995 1   0                              P N
101 8641       VAC 06/01/1988 06/15/1988 14  0                              P Y
101 8641       VAC 07/01/1989 07/15/1989 14  0                              P Y
101 G001       MAT 07/02/1991 09/28/1991 88  0   3-month Maternity leave     P Y⇒
 Maternity will be paid as 80% of Claudia's current salary.
```

Note. File positions start at 1, not 0. When you specify a File Record ID, make sure that the starting position of the Record ID is greater than 0.

ReadRecord Example

This following example uses the same File Layout definition as the previous example. The file format is CSV. The fields are separated by commas and delimited by double-quotes.

```
"8001","VAC","09/12/1981","09/26/1981","14","0","","P","Y",""
"8001","VAC","03/02/1983","03/07/1983","5","0","","P","Y",""
"8001","VAC","08/26/1983","09/10/1983","13","0","","P","Y",""
"8105","CNF","02/02/1995","??/??/","0","0","","U","N",""
"8516","MAT","06/06/1986","08/01/1986","56","0","","P","Y",""
"8516","SCK","08/06/1988","08/07/1988","1","0","","P","Y",""
"8516","VAC","07/14/1987","07/28/1987","14","0","","P","Y",""
"8553","JUR","12/12/1990","12/17/1990","5","0","Local Jury Duty","P","N",""
"8553","MAT","02/20/1992","10/01/1992","224","0","Maternity Leave","U","N",""
"8553","MAT","08/19/1994","03/01/1995","194","0","Maternity-2nd child","U","Y",""
"8553","PER","04/15/1993","04/19/1993","4","0","","U","N","Personal Day required"
"8553","SCK","01/28/1987","01/30/1987","2","0","Hong Kong Flu","P","N",""
"8553","SCK","08/02/1988","08/03/1988","1","0","Sick","P","N",""
"8553","SCK","09/12/1995","09/13/1995","1","0","","P","N",""
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
"8641","VAC","06/01/1988","06/15/1988","14","0","","P","Y",""
"8641","VAC","07/01/1989","07/15/1989","14","0","","P","Y",""
"G001","MAT","07/02/1991","09/28/1991","88","0","3-month Maternity leave","P","Y","⇒
Maternity will be paid as 80% of Claudia's current salary."
```

To read in the previous CSV file we use the following PeopleCode. It reads the file into a temporary record. First each line of the file is read into a string. The string is split into an array, with the value of each field in the array becoming an element in the array. The value of each field in the record is assigned a value from the array. After additional processing (for example, converting strings into dates or numbers, verifying data, and so on) the record can be inserted into the database. To insert the final data into the database, this code must be associated with a PeopleCode event that allows database updates, that is, SavePreChange, WorkFlow, SavePostChange, and so on. This code could also be used as part of an Application Engine program.

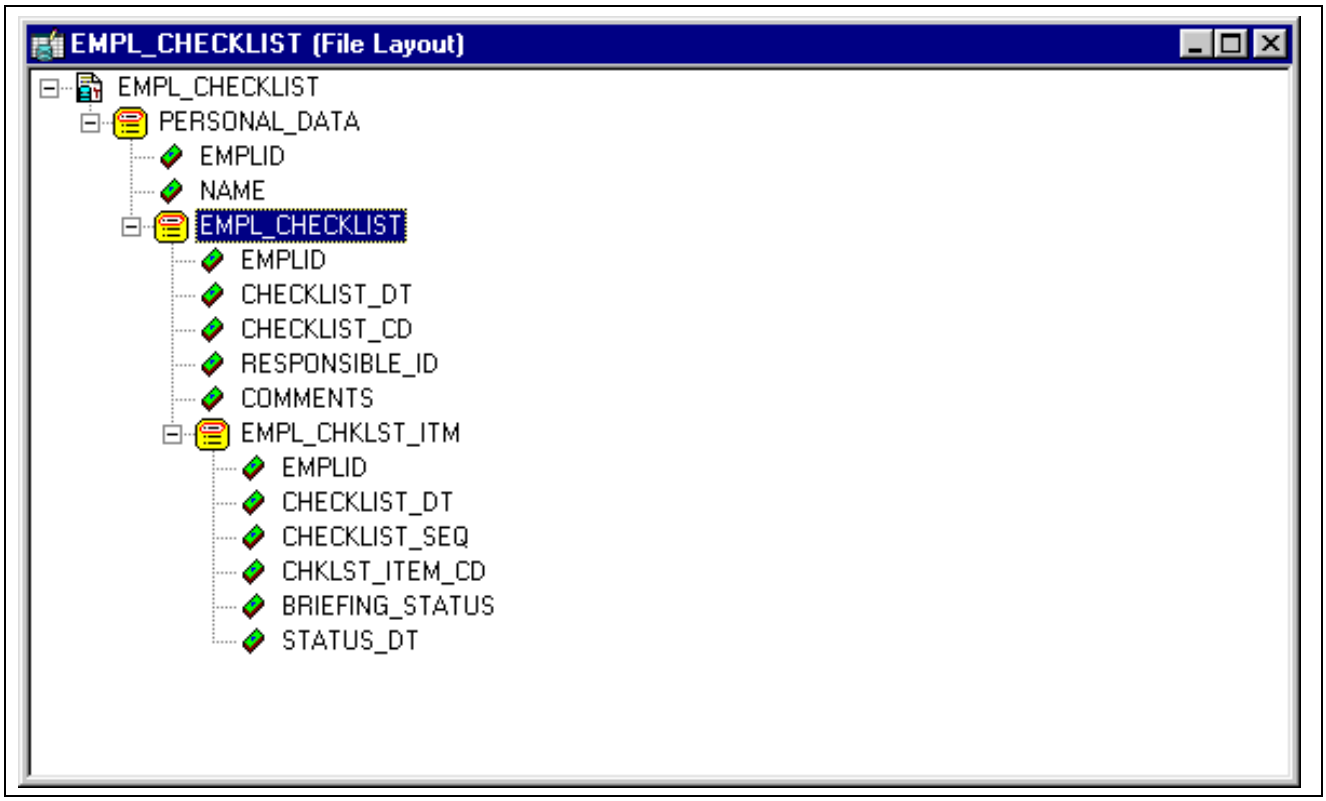
```
Local File &MYFILE;
Local Record &REC;
Local array of string &ARRAY;

&MYFILE = GetFile("c:\temp\vendor.txt", "R", %FilePath_Absolute);
&REC = CreateRecord(RECORD.ABS_HIST_TEST);
&ARRAY = CreateArrayRept("", 0);

If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.ABS_HIST) Then
      While &MYFILE.ReadLine(&STRING);
         &ARRAY = Split(&STRING, ",");
         For &I = 1 To &REC.FieldCount
            &REC.GetField(&I).Value = &ARRAY[&I];
         End-For;
      /* do additional processing here for converting values */
         &REC.Insert();
      End-While;
   Else
      /* do error processing - filelayout not correct */
   End-If;
Else
   /* do error processing - file not open */
End-If;

&MYFILE.Close();
```

Note. You can't read a file that contains a thousands separator for numeric fields. You must strip out the separator before you try to read in the file.

WriteRowset Example

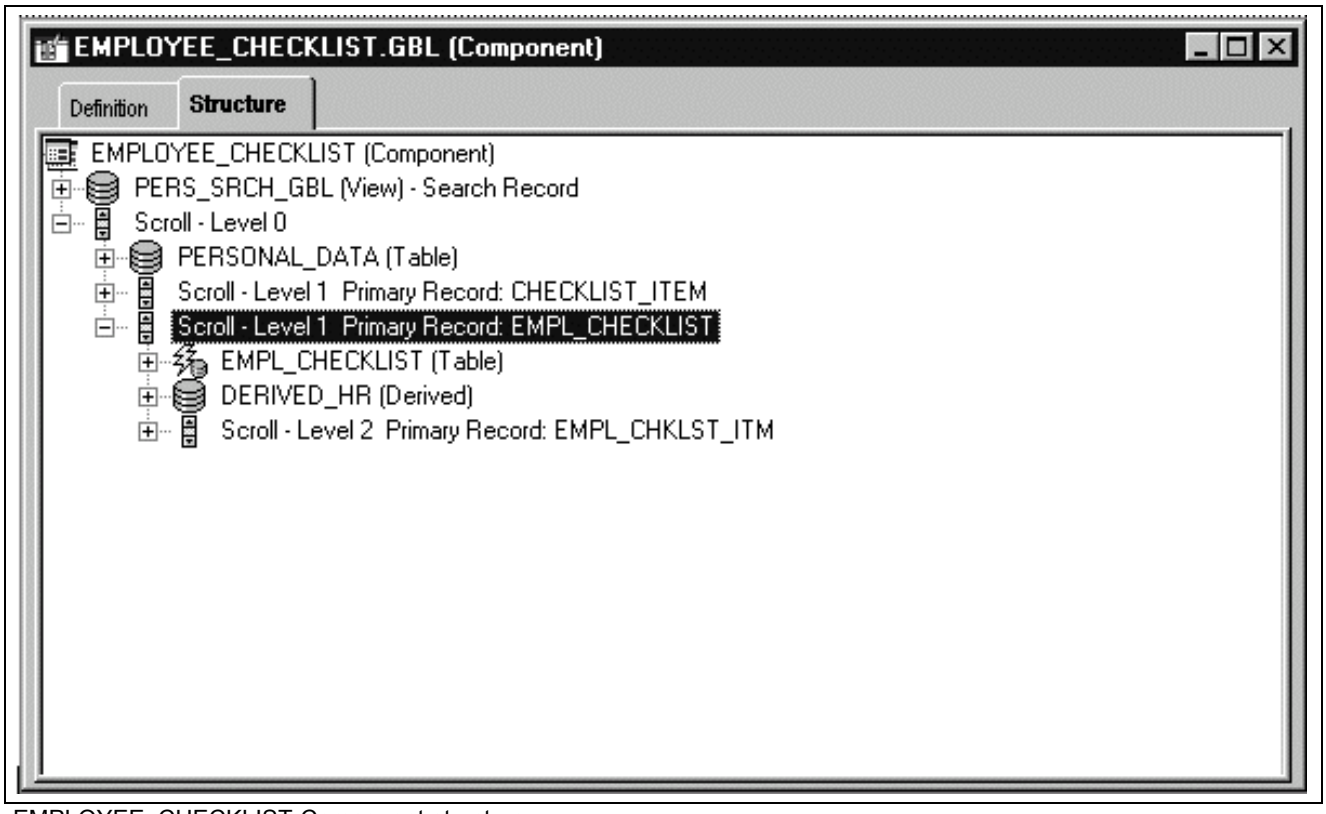
In the following example, the File Layout definition is based on the component EMPL_CHECKLIST, and looks like this:

Copyright © 1988-2007, Oracle. All rights reserved.



Example File Layout definition (EMPL_CHECKLIST)

Here's the structure of the component EMPLOYEE_CHECKLIST:



EMPLOYEE_CHECKLIST Component structure

Copyright © 1988-2007, Oracle. All rights reserved.

Note that:

- Every *field* in the two structures don't have to match (that is, every field or record that's in the file layout doesn't have to be in the component, and vice versa.)
- The two *structures* must be the same. That is, if the component has PERSONAL_DATA at level zero, and EMPL_CHECKLIST at level one, the file layout must have the same hierarchy.

The following example uses the previous File Layout definition to copy data from the EMPL_CHECKLIST page into a file.

The CreateRowset function creates an empty rowset that has the structure of the file layout definition. The GetRowset function is used to get all the data from the component and copy it into the rowset. The GetLevel0 function copies all *like-named* fields to *like-named* records. The WriteRowset method writes all the component data to the file. Because this code runs on the server, an absolute file path is used.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetRowset.

Example

The following is the PeopleCode for this example.

```
Local File &MYFILE;
Local Rowset &FILEROWSET;

&MYFILE = GetFile("c:\temp\EMP_CKLS.txt", "A", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FILELAYOUT.EMPL_CHECKLIST) Then
      &FILEROWSET = &MYFILE.CreateRowset();
      &FILEROWSET = GetLevel0();
      &MYFILE.WriteRowset(&FILEROWSET, True);
   Else
      /* file layout not found, do error processing */
   End-If;
Else
   /* file not opened, do error processing */
End-if;

&MYFILE.Close();
```

The following is a sample data file created by the previous code:

```
8113       Frumman,Wolfgang
           08/06/1999 000001           8219        Going to London office
                     100     000015 I 08/06/1999
                     200     000030 I 08/06/1999
                     300     000009 I 08/06/1999
                     400     000001 I 08/06/1999
                     500     000011 I 08/06/1999
                     600     000002 I 08/06/1999
                     700     000021 I 08/06/1999
                     800     000024 I 08/06/1999
                     900     000004 I 08/06/1999
                     1000    000006 I 08/06/1999
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
            09/06/1999 000004           7707        What to do after he arrives
                       100     000022 I 08/06/1999
                       200     000008 I 08/06/1999
                       300     000018 I 08/06/1999
                       400     000019 I 08/06/1999
8101        Penrose,Steven
            07/06/1999 000006           8229        New hire
                       1       000033 I 08/06/1999
                       2       000034 I 08/06/1999
                       3       000035 I 08/06/1999
                       4       000036 I 08/06/1999
                       5       000037 I 08/06/1999
                       6       000038 I 08/06/1999
                       7       000039 I 08/06/1999
                       8       000040 I 08/06/1999
                       9       000041 I 08/06/1999
                       10      000042 I 08/06/1999
```

When you create the File Layout definition, you can choose for each field whether to inherit a value from a higher level record field. If a value is inherited, it is written only *once* to the file, the first time it's encountered.

In the previous data example, there are three records: PERSONAL_DATA, EMPL_CHECKLIST, and EMPL_CHKLST_ITM. The field EMPLID is on all three records, but is written to the file only the first time a new value is encountered. So, the value for EMPLID is inherited by EMPL_CHECKLIST, and the value for EMPL_CHKLST_ITM is inherited from PERSONAL_DATA. The field CHECKLIST_DT is on EMPL_CHECKLIST and EMPL_CHKLST_ITM. However, the field value appears only once, in the parent record, and not in the child record.

If a record in the File Layout definition has a File Record ID, each line in the file referencing this record will be prefaced with this number. The File Record ID is not a field in the data itself. It is also referred to as the *rowid*. File Record IDs can be useful when the file being created refers to more than one record. File Record IDs can be used only with File Layout definitions that have a type of FIXED.

The following is sample file, produced with the same code, but with a File Record IDs added to all the records. 001 was added to the first level, 002 to the second, and 003 to the third.

```
001 8113        Frumman,Wolfgang
002             08/06/1999      000001 8219        Going to London office
003                         100    000015 I 10/13/1999
003                         200    000030 I 10/13/1999
003                         300    000009 I 10/13/1999
003                         400    000001 I 10/13/1999
003                         500    000011 I 10/13/1999
003                         600    000002 I 10/13/1999
003                         700    000021 I 10/13/1999
003                         800    000024 I 10/13/1999
003                         900    000004 I 10/13/1999
003                         1000   000006 I 10/13/1999
002             09/06/1999      000004 7707        What to do after he arrives
003                         100    000022 I 10/13/1999
003                         200    000008 I 10/13/1999
003                         300    000018 I 10/13/1999
003                         400    000019 I 10/13/1999
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
001 8101       Penrose,Steven
002            10/13/1999      000006 8229        New hire
003                       1      000033 I 10/13/1999
003                       2      000034 I 10/13/1999
003                       3      000035 I 10/13/1999
003                       4      000036 I 10/13/1999
003                       5      000037 I 10/13/1999
003                       6      000038 I 10/13/1999
003                       7      000039 I 10/13/1999
003                       8      000040 I 10/13/1999
003                       9      000041 I 10/13/1999
003                       10     000042 I 10/13/1999
```

ReadRowset Example

The following program reads all the rowsets from a file and updates a work scroll with that data. The work scroll isn't hidden on the page: it's created in the Component buffer from existing records using CreateRowset. The structure of the work scroll and the file layout are identical: that is, they're composed of two records, and the names of the records in the file layout are exactly the same as the names of the record definitions.

```
Local File &CHARTINPUT_F;
Local Rowset &INPUT_ROWSET, &TEMP_RS, &WORK_DATA;
Local Record &WRK_DATA;

&filename = "c:\temp\test.txt";
If FileExists(&filename, %FilePath_Absolute) Then
   &CHARTINPUT_F = GetFile(&filename, "R", "A", %FilePath_Absolute);
Else
   Exit;
End-If;

&CHARTINPUT_F.SetFileLayout(FileLayout.CHART_INFO);

/* Create rowset to be read into
NOTE that you have to start at LOWEST level of rowset */

&TEMP_RS = CreateRowset(RECORD.CHART_ITEM);
&WORK_DATA = CreateRowset(RECORD.CHART_DATA, &TEMP_RS);
&INPUT_ROWSET = CreateRowset(RECORD.CHART, &WORK_DATA);

While &INPUT_ROWSET <> Null
   &INPUT_ROWSET = &CHARTINPUT_F.ReadRowset();
   &INPUT_ROWSET.CopyTo(&WORK_DATA);
   /* do processing -- Though file may contain more than one level zero
   Component processor only allows one level zero at a time */
End-While;
```

File Rowset Considerations

The following are considerations for when you use rowsets with files.

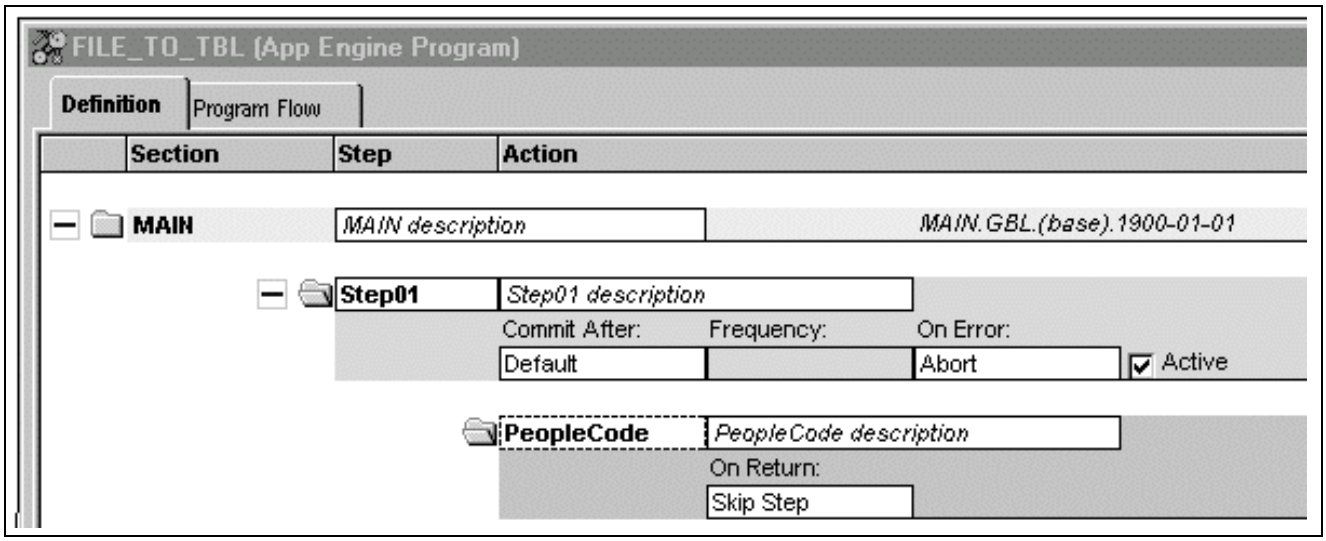
Copyright © 1988-2007, Oracle. All rights reserved.

- Although you can create a File Layout definition with more than four levels of hierarchy, a rowset created from Component buffer data can contain only four levels (level zero through 3). Any additional levels of data are ignored.
- ReadRowset populates the rowset with one transaction from the file. (A transaction is considered to be one instance of level zero data contained in a file record, plus all of its subordinate data.) WriteRowset writes one transaction to a file.

Application Engine Example

You can also use PeopleCode in an Application Engine program to either write to or read from files. This example isn't a proper Application Engine program: it contains the PeopleCode only for opening and reading from a file. However, it's included here as starting point for your own Application Engine programs.

Here is the Application Engine program:



Application Engine example program

Here is the PeopleCode in the step 1.

```
Local File &FILE;
Local Record &REC;
Local Rowset &FRS;

&FILE = GetFile("TEST.txt", "R");
&REC = CreateRecord(Record.QEPC_FILE_REC);
&SQL = CreateSQL("%Insert(:1)");

If Not &FILE.IsOpen Then
   Error ("TEST: failed file open");
Else
   If Not &FILE.SetFileLayout(FileLayout.QEPC_FILE_REC) Then
      Error ("TEST: failed SetFilelayout");
   Else
      &FRS = &FILE.ReadRowset();
      While &FRS <> Null
         &FRS.GetRow(1).QEPC_FILE_REC.CopyFieldsTo(&REC);
         &SQL.execute(&REC);
```


Copyright © 1988-2007, Oracle. All rights reserved.

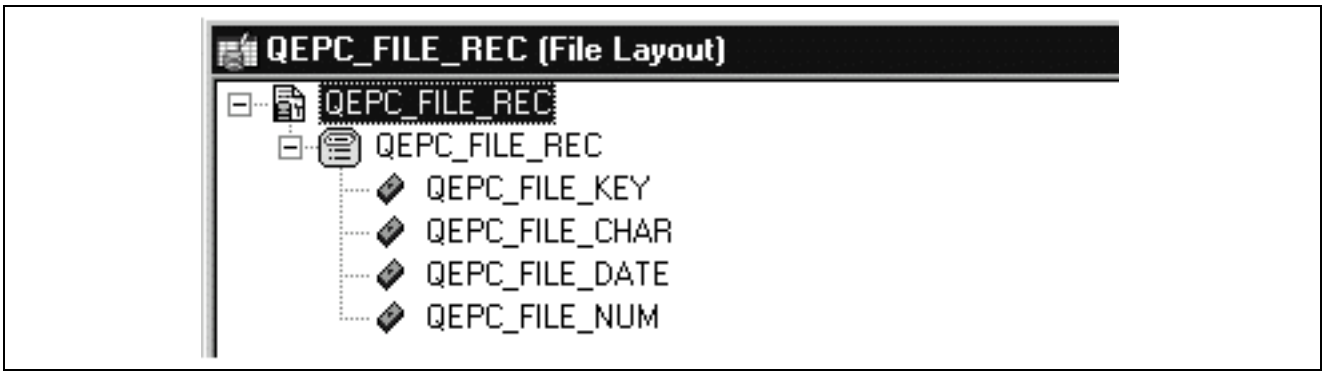
```
            &FRS = &FILE.ReadRowset();
         End-While;
      End-If;
      &FILE.Close();
   End-If;
```

The example Application Engine program reads the following CSV file:

```
"TEST2","1ST","01/01/1901",10
"TEST2","2ND","01/01/1902",20
"TEST2","3RD","01/01/1903",30
"TEST2","4TH","01/01/1904",40
```

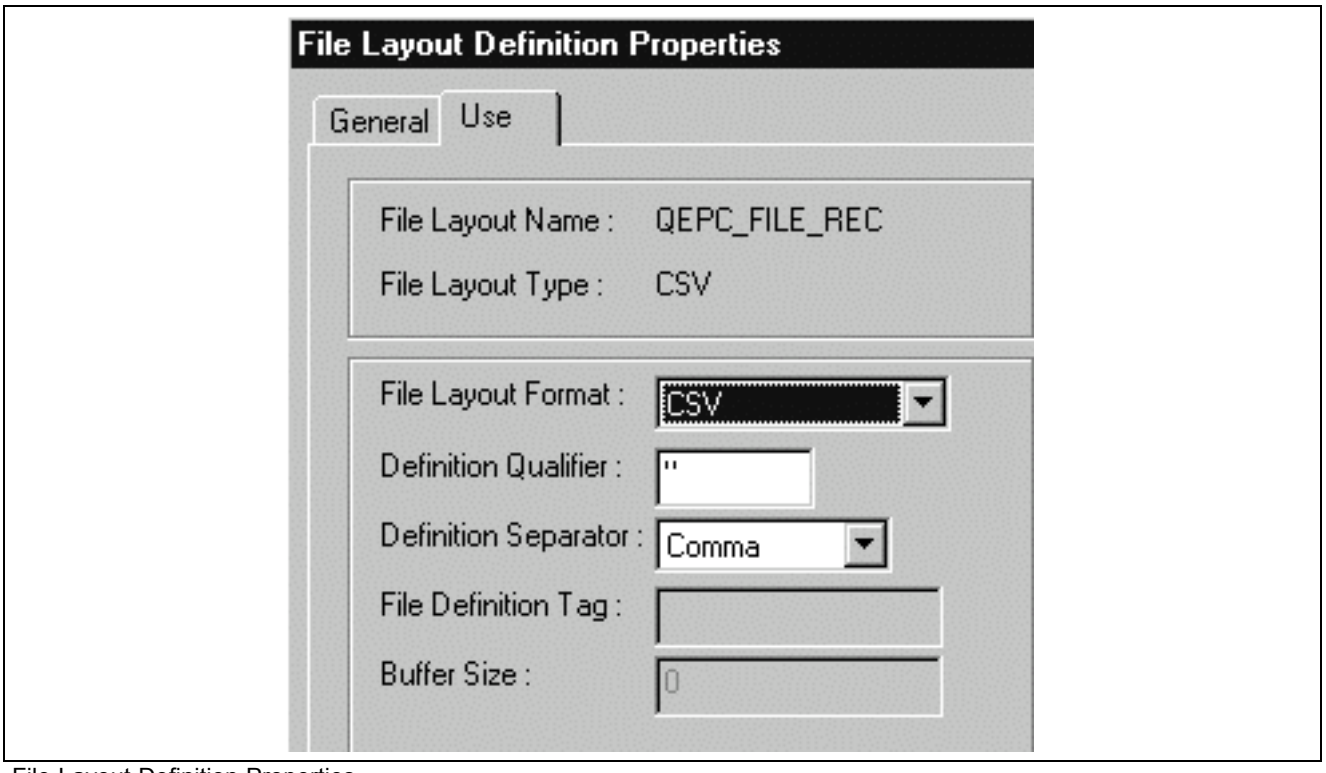
Note that the last field has no qualifier.

The File Layout used to read this record has the following form:



QEPC_FILE_REC File Layout

The properties for the QEPC_FILE_REC File Layout are as follows:

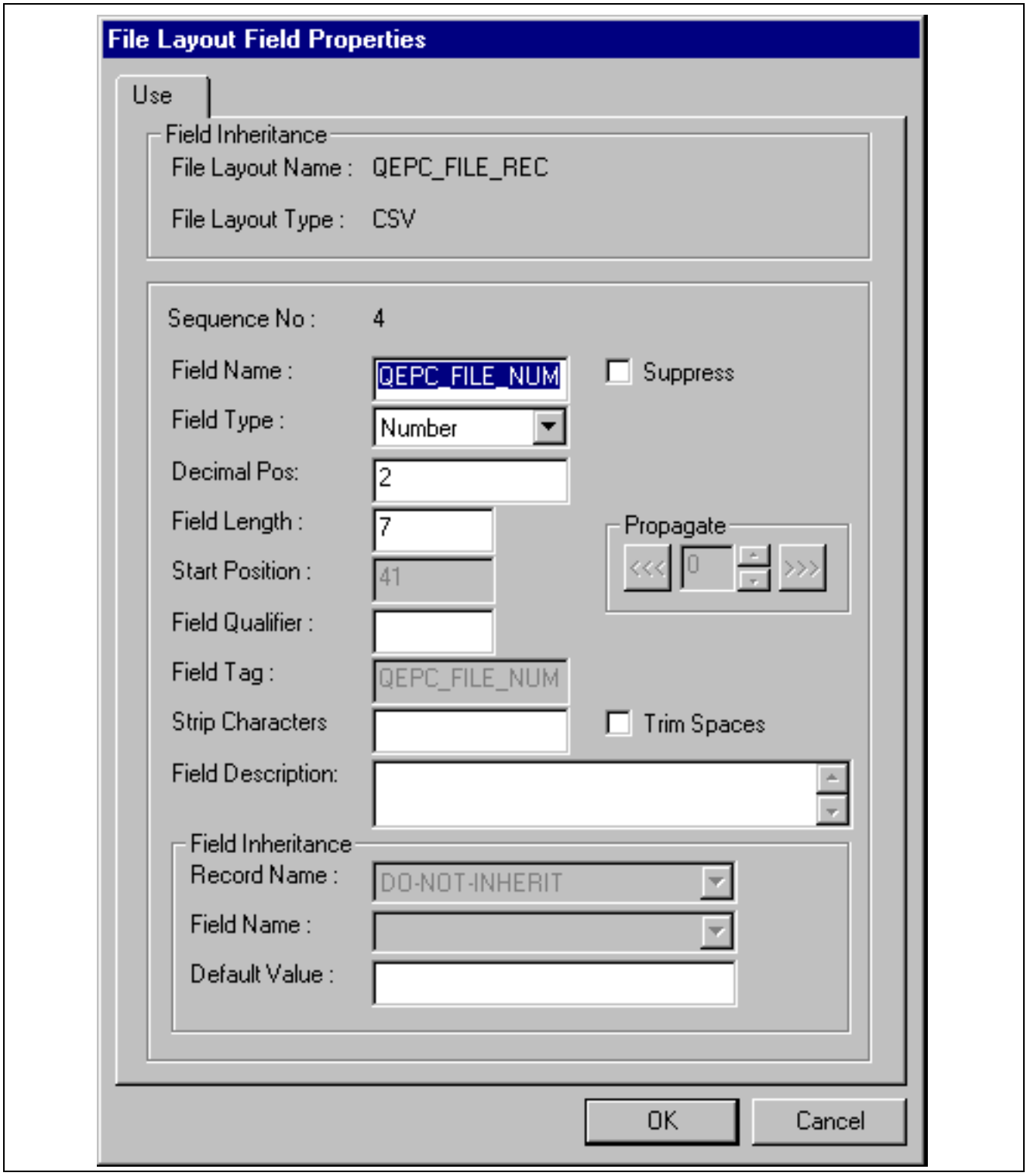


File Layout Definition Properties

Copyright © 1988-2007, Oracle. All rights reserved.

Note that the Definition Qualifier is double-quotes ("), while the separator is a comma.

Remember, the last field didn't have a qualifier. To account for that, the field properties for this field must be edited, and a blank must be put in the Field Qualifier property.



File layout Field Properties

Copyright © 1988-2007, Oracle. All rights reserved.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Engine*, “Understanding PeopleSoft Application Engine”.

Multiple File Layouts

In the previous examples, the input file contained rowsets based on a single File Layout definition. However, PeopleTools provides the functionality to process input files containing rowsets that require several *different* File Layouts.

Note. You can use only FIXED format files to implement multiple file layouts.

See the SetFileId method for details about handling multiple file layouts.

In this section, we discuss how to:

- Read multiple file layouts.
- Write multiple file layouts.

See Chapter 16, “File Class,” SetFileId, page 540.

Reading Multiple File Layouts

If your input file contains data based on more than one File Layout, it must contain an indicator, called a FileId that specifies:

- When a different File Layout definition should be used.
- Which File Layout definition should be used.

The FileId must be specified on a separate line and must precede *every* rowset that requires a layout different from the previous rowset. It isn’t considered part of the rowset.

In the following example, the file contains two FileId lines; they use a file record ID that distinguishes them from the rowset data—in this case, "999".

```
999 PRODUCT  /* The following rowset uses the PRODUCT layout */
001  /* Level 0 record data */
101  /* level 1 record data */
201  /* Level 2 record data */
201  /* Level 2 record data */
999 ORDER  /* The following two rowsets use the ORDER layout */
001  /* Level 0 record data */
111  /* Level 1 record data */
111  /* Level 1 record data */
001  /* Level 0 record data */
111  /* Level 1 record data */
111  /* Level 1 record data */
```

The FileId can contain any information you want that indicates which file layout to use; the "PRODUCT" and "ORDER" fields shown are just examples.

To read this file, you should do the following in your program:

Copyright © 1988-2007, Oracle. All rights reserved.

1. Use the SetFileId method to specify the file record ID value.
2. Use the ReadRowset method to read the data.
3. Check if the rowset is NULL.

   NULL indicates you've reached either the end of the file or a new rowset.
4. Use the IsNewFileId property to check if the next line is a FileId file record (line).

- If IsNewFileId is False, you've reached the end of the file.
- If IsNewFileId is True, use the CurrentRecord property to determine which File Layout to use next.

The following example reads rowsets from a file. When it finds a new rowset (indicated by &IsNewFileId returning True) the value of &CurrentRecord is passed to a function that reads and evaluates the line, then returns the name of the new file layout. (The code for the function FindFileId is included at the start of the example.)

```
Local File &MYFILE;
Local Rowset &rsFile;
Local Record &rSomeRec1, &rSomeRec2;
Local SQL &SQL1;

Function FindFileID(&CurrentRecord As string) Returns string ;
   Evaluate RTrim(Substring(&CurrentRecord, 5, 50))
   When "SOME_REC1"
      &FILELAYOUT = "SOME_REC1";
   When "SOME_REC2"
      &FILELAYOUT = "SOME_REC2";
   End-Evaluate;
   Return &FILELAYOUT;
End-Function;

&rSomeRec1 = CreateRecord(Record.SOME_REC1);
&rSomeRec2 = CreateRecord(Record.SOME_REC2);
&SQL1 = CreateSQL("%Insert(:1)");
&MYFILE = GetFile("c:\temp\MULTI_FILE.out", "R", %FilePath_Absolute);

rem Set temporary first file layout;
&MYFILE.SetFileLayout(FileLayout.SOME_REC1);
&MYFILE.SetFileId("999", 1);

rem Read rowset to find actual first file ID;
&rsFile = &MYFILE.ReadRowset();
&CurrentRecord = &MYFILE.CurrentRecord;
&IsNewFileID = &MYFILE.IsNewFileId;
If &MYFILE.IsNewFileId Then
   &FILELAYOUT = FindFileID(&CurrentRecord);
   &MYFILE.SetFileLayout(@("FileLayout." | &FILELAYOUT));
   &MYFILE.SetFileId("999", 1);
End-If;

rem Read first 'real' rowset;
&rsFile = &MYFILE.ReadRowset();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&CurrentRecord = &MYFILE.CurrentRecord;
&IsNewFileID = &MYFILE.IsNewFileId;
While &rsFile <> Null Or
      &IsNewFileID
   If &MYFILE.IsNewFileId Then
      &FILELAYOUT = FindFileID(&CurrentRecord);
      &MYFILE.SetFileLayout(@("FileLayout." | &FILELAYOUT));
      &MYFILE.SetFileId("999", 1);
      If &IsNewFileID Then
         &rsFile = &MYFILE.ReadRowset();
         &CurrentRecord = &MYFILE.CurrentRecord;
         &IsNewFileID = &MYFILE.IsNewFileId;
      End-If;
   End-If;

   Evaluate &FILELAYOUT

   When "SOME_REC1"
      &rsFile(1).SOME_REC1.CopyFieldsTo(&rSomeRec1);
      &rSomeRec1.ExecuteEdits(%Edit_Required);
      If Not &rSomeRec1.IsEditError Then
         &SQL1.Execute(&rSomeRec1);
      End-If;
      Break;
   When "SOME_REC2"
      &rsFile(1).SOME_REC2.CopyFieldsTo(&rSomeRec2);
      &rSomeRec2.ExecuteEdits(%Edit_Required);
      If Not &rSomeRec2.IsEditError Then
         &SQL1.Execute(&rSomeRec2);
      End-If;
   End-Evaluate;

   &rsFile = &MYFILE.ReadRowset();
   &CurrentRecord = &MYFILE.CurrentRecord;
   &IsNewFileID = &MYFILE.IsNewFileId;
End-While;

&MYFILE.Close();
```

See Chapter 16, “File Class,” SetFileId, page 540; Chapter 16, “File Class,” ReadRowset, page 538 and Chapter 16, “File Class,” IsNewFileId, page 552.

Writing Multiple File Layouts

If you’re writing files that contain data based on more than one File Layout definition, consider the following points:

- If the file is going to a third-party vendor, you should work with the third-party to determine what their requirements are for specifying the different data formats.
- If the file is going to be used by another PeopleSoft system, you must add the FileId between each rowset that requires a different layout. FileId file records are *not* part of any rowset. They should be designed so

Copyright © 1988-2007, Oracle. All rights reserved.

they won't be mistaken for part of a rowset. You can create and write them to the file in many ways. The following are suggestions:

- Build each line as a string, using any of the built-in string manipulation functions, then write them to the file using the File class WriteLine or WriteString methods.
- Design a file layout consisting of a single file record definition for the FileId file records, then build the records using Record Class methods and functions, and write them to the file using the WriteRecord method.

The following code example writes each record from the level one scroll on a page to the file using a different File Layout. Between each WriteRowset the File ID file record is written to the file, describing the new File Layout being used.

```
Local File &MYFILE;
Local Rowset &FILEROWSET;
Local Record &REC1, &REC2;
Local SQL &SQL;

&MYFILE = GetFile("c:\temp\Records.txt", "W", %FilePath_Absolute);
If &MYFILE.IsOpen Then
   If &MYFILE.SetFileLayout(FileLayout.TREE_LEVEL) Then
      &REC1 = CreateRecord(Record.PSTREELEVEL);
      &FILEROWSET = &MYFILE.CreateRowset();
      &SQL = CreateSQL("%Selectall(:1)", &REC1);
      /* write first File ID to file */
      &MYFILE.WriteLine("999 FILE LAYOUT 1");
      While &SQL.Fetch(&REC1)
         &REC1.CopyFieldsTo(&FILEROWSET.GetRow(1).PSTREELEVEL);
         &MYFILE.WriteRowset(&FILEROWSET, True);
      End-While;
   Else
      /* file layout not found, do error processing */
   End-If;

   If &MYFILE.SetFileLayout(FileLayout.TREE_USERLEVEL) Then
      &REC2 = CreateRecord(Record.TREE_LEVEL_TBL);
      &FILEROWSET = &MYFILE.CreateRowset();
      &SQL = CreateSQL("%Selectall(:1)", &REC2);
      /* write second File ID to file */
      &MYFILE.WriteLine("999 FILE LAYOUT 2");
      While &SQL.Fetch(&REC2)
         &REC2.CopyFieldsTo(&FILEROWSET.GetRow(1).TREE_LEVEL_TBL);
         &MYFILE.WriteRowset(&FILEROWSET);
      End-While;
   Else
      /* file layout not found, do error processing */
   End-If;
Else
   /* file not opened, do error processing */
End-If;
&MYFILE.Close();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Chapter 16, “File Class,” WriteLine, page 545; Chapter 16, “File Class,” WriteString, page 550 and Chapter 16, “File Class,” WriteRecord, page 546.

See Chapter 31, “Record Class,” page 1377.

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 17

Grid Classes

This chapter provides an overview of Grid class and the GridColumn subobject and discusses the following topics:

- Shortcut considerations
- The Grid class in PeopleCode
- Grid or Grid Column declaration
- Scope of a Grid or Grid Column class
- Grid Class built-in function
- Grid Class reference
- Grid Column class

Note. Use the grid classes to control the display of a grid control. If you want to manipulate an analytic grid, used with PeopleSoft Analytic Calculation Engine data, you must use the AnalyticGrid classes.

See Also

Chapter 4, "Analytic Grid Classes," page 129

Understanding Grid and GridColumn Classes

A grid looks and behaves like a spreadsheet embedded in a page: it has column headings, row headings, cells, and horizontal and vertical scroll bars. It can be used instead of a single-level scroll. It is analogous to a scroll region on a page. Each row in a grid corresponds to a set of controls in a scroll occurrence. Each cell in a grid corresponds to a field on a page.

The grid class enables you to instantiate only GridColumn objects, which in turn enables you to change grid column attributes without having to write PeopleCode that loops through every row of the grid.

Note. PeopleSoft builds a page grid one row at a time. Because the grid class applies to a complete grid, you can't attach PeopleCode that uses the grid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 17, "Grid Classes," GridColumn Class, page 580

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Using Page Controls," Using Grids

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Activate Event

Shortcut Considerations

An expression of the form

```
&MYGRID.columnname.property
```

is converted to an object expression by using**GetColumn**(*columnname*). The following examples are equivalent.

Using *columnname:*

```
&MYGRIDCOLUMN = &MYGRID.CHECKLIST_ITEMCODE;
```

Using the GetColumn method:

```
&MYGRIDCOLUMN = &MYGRID.GetColumn("CHECKLIST_ITEMCODE");
```

The following examples are equivalent.

Using *columnname*:

```
&MYGRID.CHECKLIST_ITEMCODE.Visible = False;
```

Using the GetColumn method:

```
&MYGRID.GetColumn("CHECKLIST_ITEMCODE").Visible = False;
```

The Grid Class in PeopleCode

The PeopleCode grid object is a reference to a page runtime object for the grid. These particular page runtime objects aren't present until the Component is started.

Note. PeopleSoft builds a page grid one row at a time. Because the Grid class applies to a complete grid, you can't attach PeopleCode that uses the Grid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

If you're using the grid within a secondary page, the runtime object for the grid isn't created until the secondary page is run. The grid object can't be obtained until then, which means that the earliest PeopleCode event you can use to activate a grid that's on a secondary page is the Activate event for the secondary page.

The attributes you set for displaying a page grid remain in effect only while the page is active. When you switch between pages in a component, you have to reapply those changes *every time* the page is displayed.

 Copyright © 1988-2007, Oracle. All rights reserved.

In addition, the Activate event associated with a page fires every time the page is displayed. Any PeopleCode associated with that Activate event runs, which may undo the changes you made when the page was last active. For example, if you hide a grid column in the Activate event, then display it as part of a user action, when the user tabs to another page in the component, then tabs back, the Activate event runs again, hiding the grid column again.

If a user at runtime hides a column of a grid, tabs to another page in the component, then tabs back to the first page, the page is refreshed and the grid column is displayed again.

When you place a grid on a page, the gird is automatically named the same as the name of the primary record of the scroll for the grid. This is the name you use with the GetGrid function. You can change this name on the Record tab of the Grid properties.

Note. There is no visible property for a grid, just a grid column. However, you can still hide an entire grid. Remember, many of the Rowset methods and properties work on a grid. If you want to hide an entire grid, get the rowset for that grid by using the HideAllRows Rowset class method.

Use the grid classes to access an ordinary grid. Use the analytic grid classes to access an analytic grid.

See Also

Chapter 4, “Analytic Grid Classes,” page 129

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer’s Guide, “PeopleCode and the Component Processor,” Activate Event

Chapter 33, “Rowset Class,” HideAllRows, page 1436

Data Type for a Grid or Grid Column Object

Grids are declared using the Grid data type. For example,

```
Local Grid &MYGRID;
```

Grid Columns are declared using the GridColumn data type. For example:

```
Local GridColumn &MYGRIDCOL;
```

Scope of a Grid or Grid Column Object

Both the grid and grid column objects can be instantiated from PeopleCode only.

A grid is a control on a page. You generally use these objects only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message subscription, a Component Interface, and so on.

In addition, PeopleSoft builds a page grid one row at a time. Because the Grid class applies to a complete grid, you can’t attach PeopleCode that uses the Grid class to events that occur before the grid is built; the earliest event you can use is the page Activate Event.

Copyright © 1988-2007, Oracle. All rights reserved.

Grid Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetGrid

Grid Class Method

In this section, we discuss the GetColumn Grid class method.

GetColumn

Syntax

GetColumn(*columnname*)

Description

The GetColumn method instantiates a grid column object from the Grid class, and populates it with a grid column from the grid object executing this method. Specify the grid column name in the page field properties for that field, consisting of any combination of uppercase letters, digits and "#", "$", "@", and "_".

To specify a grid column name:

1. Open the page in PeopleSoft Application Designer, select the grid and access the page field properties.
2. Select the Columns tab on the grid properties.
3. Either double-click on the grid column you want to name, or select the column and click the Properties button.
4. On the General tab, type the grid column name in the Page Field Name field.

Note. Although it's possible to base multiple grid columns on the same record field, the Page Field Name you enter for each grid column must be unique, not just for that grid, but for that page. If you have two grids on a page, every page field name must be unique to the page.

Parameters

Parameter	Description
columnname	Specify a string containing the value of the Page Field Name on the General tab of the grid column's properties. You must have previously entered a value for the Page Field Name for the grid column you want to work with.

Returns

A GridColumn object populated with the grid column specified as the parameter to this method.

Example

```
local Grid &MYGRID;
local GridColumn &MYGRIDCOLUMN;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&MYGRID = GetGrid(PAGE.EMPLOYEE_CHECKLIST, "EMPL_GRID");
&MYGRIDCOLUMN = &MYGRID.GetColumn("CHECKLIST_ITEMCODE");
```

The following function loops through rows on a grid. The function finds each row that is selected. It does this through the Selected property of the Row class of PeopleCode. Data is then moved from the selected row to a new row, on a different grid, in the same page. The way in which this function is written, data is moved from &SCROLL_SHELF to &SCROLL_CART. These are two different rowset objects, of two different grids, on the same page. Note that the two grids in this example are on the same occurs level.

```
/* Moving data between grids on the same occurs level */
/* of the same page */

Local Rowset &SCROLL_CART, &SCROLL_SHELF;

Function move_rows(&SCROLL_CART As Rowset, &SCROLL_SHELF As Rowset);

   &I = 1;
/* loop to find whether row is selected */
   Repeat
   If &SCROLL_SHELF.GetRow(&I).Selected = True Then

      If All(&SCROLL_CART(1).GetRecord(1).QEPC_ITEM.Value) Then

         &SCROLL_CART.InsertRow(&SCROLL_CART.ActiveRowCount);
      End-If;

   /* if it is selected move data to other grid   */

      &SCROLL_SHELF.GetRow(&I).GetRecord(1).CopyFieldsTo(&SCROLL_CART.GetRow⇒
(&SCROLL_CART.ActiveRowCount).GetRecord(1));

/* delete row from current grid so data disappears  */

      &SCROLL_SHELF.DeleteRow(&I);
      &I = &I - 1;
   End-If;

   &I = &I + 1;
   Until &I = &SCROLL_SHELF.ActiveRowCount + 1;

/* end of loop ************************************/

End-Function;

/********* end of function ****************************/

/* Creating the rowset object */

&SCROLL_CART = GetLevel0()(1).GetRowset(SCROLL.QEPC_CART);
&SCROLL_SHELF = GetLevel0()(1).GetRowset(SCROLL.QEPC_SHELF);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* calling the function   */

move_rows(&SCROLL_CART, &SCROLL_SHELF);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetGrid

Grid Class Properties

In this section, we discuss the grid class properties. The properties are described in alphabetical order.

gridcolumn

Description

If a grid column name is used as a property, it accesses the grid column with that name. This means the following code:

```
&Mycolumn = &MyGrid.gridcolumnname;
```

acts the same as

```
&Mycolumn = &MyGrid.GetColumn(columnname);
```

This property is read-only.

Label

Description

This property returns a string specifying the label that appears as the title of the grid.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters. Always put any changes to labels in the Activate event. This way the label is set every time the page is accessed.

This property is read-write.

GridColumn Class

Grid columns are page fields that comprise a grid, which is itself a page field. Similarly, a group of GridColumn objects comprises a Grid object. The GridColumn class enables you to change grid column attributes without having to write PeopleCode that loops through every row of the grid.

This class has no methods or built-in functions, only properties.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. To use a GridColumn object, you must instantiate it from a grid object. This requires you to first instantiate a Grid object using the GetGrid built-in function.

You create an analytic grid column object from the analytic grid class. However, the analytic grid column class has the same properties as the grid column class. Any differences are noted in the documentation.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetGrid.

GridColumn Class Properties

In this section, we discuss each GridColumn class property.

Enabled

Description

This property specifies whether the fields in the column are enabled (that is, can be edited) or if they are "disabled", that is, un-editable. All columns are enabled by default.

Note. For an analytic grid column, the Enabled property only works with cubes, not dimensions.

This property is read-write.

Example

```
If Not &ValidID Then
   &MYGRID = GetGrid(PAGE.EMPLOYEE_CHECKLIST, "EMPL_GRID");
   &MYGRIDCOLUMN = &MYGRID.GetColumn("CHECKLIST_ITEMCODE");
   &MYGRIDCOLUMN.Enabled = False;
End-If;
```

Label

Description

This property returns a string specifying the display label for the GridColumn object executing the property, as distinct from the grid column's Record Field Name and Page Field Name. This property overrides the equivalent settings on the Label tab of the grid column's Page Field Properties.

Note. You can't use this property to set labels longer than 100 characters. If you try to set a label of more than 100 characters, the label is truncated to 100 characters. If you change the column label, then change the field label, you may overwrite the column label with the field label. The grid object is part of a page, *not* the data buffers, while the field is part of the data buffers. To avoid this problem, always put any changes to column labels in the Activate event. This way the label is set every time the page is accessed.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MYGRIDCOLUMN.Label = "Checklist Item";
```

Name

Description

This property returns the name of the grid column, as a string. This value comes from the Page Field Name on the General tab in the Page Field Properties of the GridColumn object executing the property. This property was the value used to instantiate the GridColumn object.

This property is read-only.

Example

```
&PF_NAME = &MYGRIDCOLUMN.Name;
```

Visible

Description

This property returns a Boolean value which specifies whether a grid column should be hidden from view. Set this property to False to hide the grid column, and to True to unhide the grid column. This property defaults to True.

The Visible property also hides grid columns that are displayed as tabs in the PeopleSoft Pure Internet Architecture.

If you specify "Show Column on Hide Rows" in PeopleSoft Application Designer, the column headers and labels of a grid display at runtime, even when the rest of the column is hidden. You can’t override this value using PeopleCode.

Note. For an analytic grid column, this property is only valid if the column is bound to a dimension on a slicer, and bound to a cube that is on the column axis.

This property is read-write.

Note. In previous releases of PeopleTools, the methodology for hiding a grid column was to use the Hide function in a loop to hide each cell in the column, one row at a time. This method of hiding grid columns still works. However, your application may experience deteriorated performance if you continue to use this method.

Example

```
&MYGRIDCOLUMN.Visible = False;
```

The following example checks for the value of a field in every row of the grid. If that value is "N" for every row, the column is hidden.

```
&RS = GetRowset(Scroll.EX_SHEET_LINE);
&HIDE = True;
While (&HIDE)

   For &I = 1 To &RS.ActiveRowCount;
      &OUT_OF_POLICY = &RS(&I).EX_SHEET_LINE.OUT_OF_POLICY.Value;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
      &NO_RECEIPT_FLG = &RS(&I).EX_SHEET_LINE.NO_RECEIPT_FLG.Value;
      If &OUT_OF_POLICY = "Y" Then
         &OUT_OF_POLICY_HIDE = False;
         &HIDE = False;
      End-If;

      If &NO_RECEIPT_FLG = "Y" Then
         &NO_RECEIPT_HIDE = False;
         &HIDE = False;
      End-If;
   End-For;

   If &HIDE = True Then
      &HIDE = False;
   End-If;

End-While;

If Not (&OUT_OF_POLICY_HIDE) Then
   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").OUT_OF_POLICY.Visible = True;

Else

   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").OUT_OF_POLICY.Visible = False;

End-If;

If Not (&NO_RECEIPT_HIDE) Then

   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").NO_RECEIPT_FLG.Visible =⇒
 True;

Else

   GetGrid(Page.EX_SHEET_LINE_APV1, "EX_SHEET_LINE").NO_RECEIPT_FLG.Visible =⇒
 False;

End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, “Using Page Controls,” Using Grids

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 18

Internet Script Classes (iScript)

This chapter describes how an Internet Script (iScript) works in your application, how to create an iScript, and the Internet Script classes that you use in your PeopleCode. It also discusses the following topics:

- URL vs. URI.
- Web libraries.
- iScript security.
- When to use an iScript.
- Details of an iScript URL.
- The Generate functions.
- Error handling.
- Scope of the Internet Script classes.
- Data Types of the Internet Script classes.
- Internet Script classes built-in functions.
- Internet Script reference.

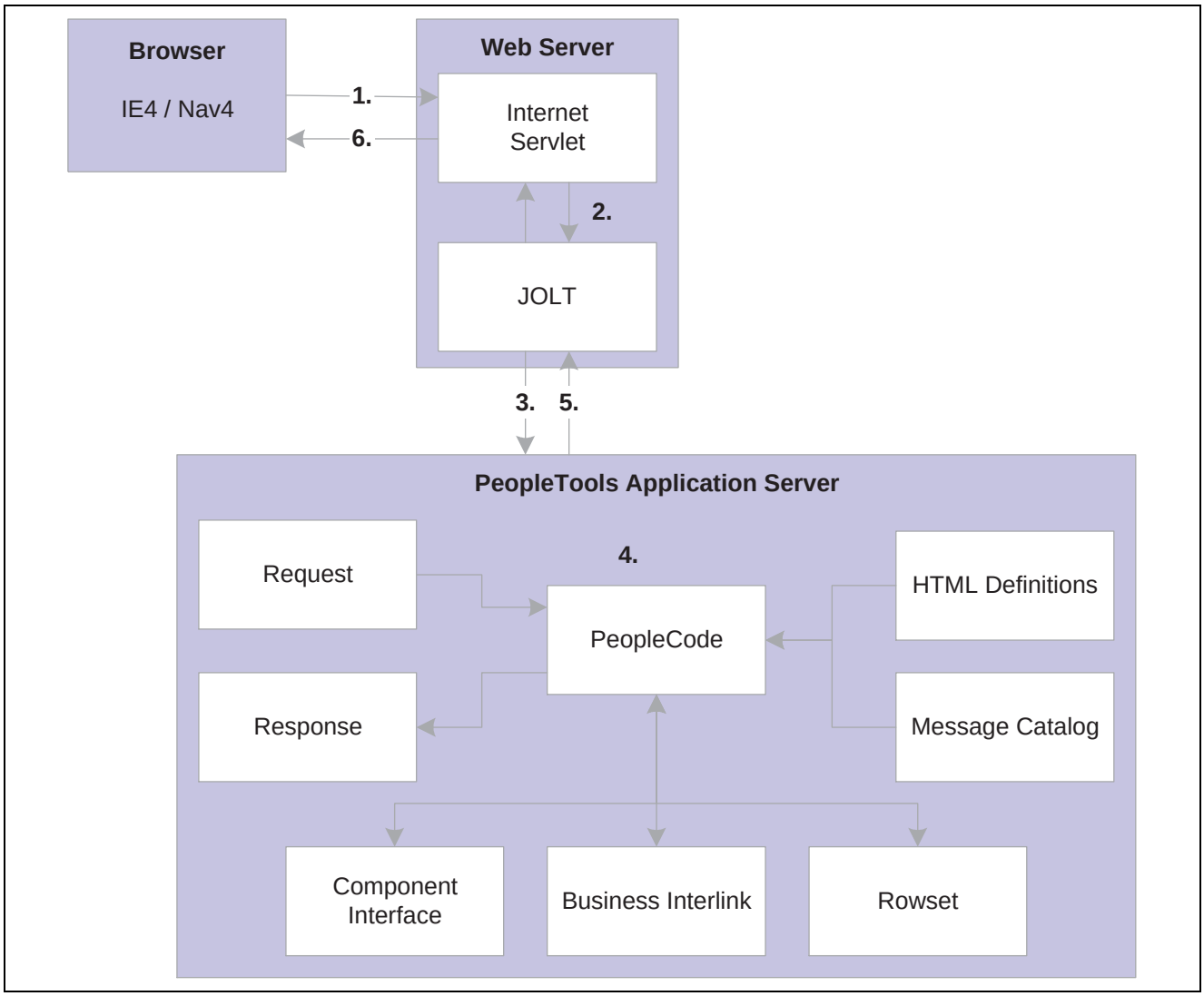
Understanding Internet Script Classes

An Internet Script, also called an iScript, is a specialized PeopleCode function that generates dynamic web content. iScripts interact with web clients (browsers) using a request-response paradigm based on the behavior of the Hypertext Transfer Protocol.

iScripts work with PeopleSoft Pure Internet Architecture. You must have PeopleSoft Pure Internet Architecture set up correctly before you can run an iScript.

The following flow chart shows the interaction of an iScript (PeopleCode) in a PeopleSoft Pure Internet Architecture application.

Copyright © 1988-2007, Oracle. All rights reserved.



Communication flow chart for Internet Scripts

1. An HTTP Request arrives at the web server specifying in its URL the Presentation Relay Servlet (psc). The path information included in the URL specifies which PeopleCode program to run. No component is specified because none is needed. The PeopleCode program (iScript) runs directly. The following is an example of the URL:

   ```
   http://serverx/Servlets/psp/ps84/e_procurement/fdm/s/WEBLIB_BEN_401k.PAGES.Field⇒
   Formula.IScript_Home401k
   ```

2. The psc servlet is invoked by the web server. It serializes all necessary information from the Web Server Request and Response objects, then makes a call through JOLT to the PeopleSoft Application Server service passing the serialized object information.
3. The Application Server unpacks all of the object information and creates PeopleSoft versions of the Request and Response objects. It then calls the function in the PeopleCode program specified in the URL. The PeopleCode Request and Response objects are available to the PeopleCode program through the system variables %Request and %Response.
4. The PeopleCode program is then responsible for generating every aspect of the HTML page that is to be returned to the browser. It has access to the Request object to view items such as query string parameters, cookies, and headers. It also has access to Component Interfaces, Business Interlinks,

Copyright © 1988-2007, Oracle. All rights reserved.

and Rowset objects to interact with applications. HTML definitions are available to include language sensitive blocks of HTML. They can also contain JavaScript. The Message Catalog is also available for language sensitive messages. The Response object is used to "write" the generated HTML back to the browser.

5. When the PeopleCode program is finished running, the headers, cookies, and HTML in the PeopleSoft Response object are serialized and returned through JOLT to the Presentation Relay Servlet.
6. The Presentation Relay Servlet then unpacks the Response information generated by the PeopleCode, and plays it back to the web server response object, which in turn sends the response to the user's browser.

URL vs. URI

In this document, the term URL refers to the entire URL, that points to content. The following is an example of a URL:

```
http://someserver/servlets/psc/ps84/e_procurement/fdm/s/WEBLIB_Portal.Field⇒
Change.IScript_DoSomething?page=view&key1=value1&key2=value2
```

A URI does not include the content information. Think of it as a subset of the URL that points to the location of the resource, but does not include any parameters passed to that resource. From the previous example, the URI portion of the URL is:

```
http://someserver/servlets/psc/ps84
```

Web Libraries

iScripts use the existing PeopleCode Function Library infrastructure. However, instead of naming your record FUNCLIB_xxx, all iScripts *must* be contained in records named WEBLIB_xxx. All of the existing tools and techniques for working with function libraries (such as Upgrade, Find In, Rename, and so on) also apply to Web Libraries.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide*, "Using the PeopleCode Editor," Accessing PeopleCode External Functions.

To create an iScript:

1. Create or extend a domain specific WEBLIB (funclib).

   WEBLIBs are derived/work record definitions that have their name prefixed with WEBLIB_.

2. Add a function to a WEBLIB.

   The name of your function *must* be prefaced with IScript_. For example:

   ```
   IScript_HelloWorld

   IScript_FuncHRPage
   ```

Note. iScript functions take no arguments, and do not return a value.

Copyright © 1988-2007, Oracle. All rights reserved.

The following iScript writes data.

```
Function IScript_HPDefaultCategories()

   &ClearDotImage = %Response.GetImageURL(Image.PT_PORTAL_CLEAR_DOT);
   &CatHTML = GetHTMLText(HTML.PORTAL_HP_CATEGORY, &ClearDotImage, GetCategories⇒
());
   %Response.Write(&CatHTML);

End-Function;
```

The following iScript uses both the request and response objects to echo what the user types into an edit control.

Example iScript

```
Function IScript_HelloWorld()

%Response.WriteLine("<html><head><title>Hello World</title></head><body>");
   %Response.WriteLine("<h1>Hello World</h1>");
   %Response.WriteLine("<form method=POST action = " | %Request.FullURI | "?" |⇒
 %Request.QueryString | "><input type=submit value='Say this!'> <input type=⇒
edit name=WhatYouSaid value=" | %Request.GetParameter("WhatYouSaid") | "><⇒
/form>");

   rem echo back what the user typed into the edit box...;
   %Response.WriteLine("<p><font face='Arial, Helvetica' size='5' color='blue'⇒
 >You said: ");
   %Response.WriteLine(%Request.GetParameter("WhatYouSaid"));

   %Response.WriteLine("</body></html>");
   Return;
End-Function;
```


Copyright © 1988-2007, Oracle. All rights reserved.

iScript Security

iScripts are secured on your system similar to Component Interfaces. After you create a WEBLIB record, and your functions, you must add them just as you would add a page to an application.

To add security to an iScript :

1. Navigate to the Maintain Security page and open the Permission List to which you want to give access.
2. Select Web Libraries.

   You may need to scroll through the tabs at the top of the page to access the Web Libraries tab.
3. (Optional) Add new WebLibrary for Permission List.

   If this is a new Web Library, you must add it to the permission list. Click the plus button to add a new Web Library, then select the library you want to add from the drop-down list box.
4. Select Edit for the Web Library you want to grant access for.
5. Select the access you want to give each function in the WEBLIB record.

   From the page that displays, you can choose to allow or disallow access on a per function basis. The buttons on the side of the page either grant or disallow access for all functions.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Security Administration*

When to Use an iScript

iScripts give you complete control over the HTML sent to the browser. This enables flexibility in creating a user interface. However, there are some responsibilities that you inherit when choosing to use iScripts.

- You are responsible for creating HTML and JavaScript that is cross-browser compatible.
- You are responsible for ensuring that all the text sent to the browser is stored in either the message catalog or in HTML definitions to enable translation of the text.
- iScripts aren't part of the regular Component Processor flow, so you are responsible for accessing the database in a multi-language, multi-market, and multi-currency sensitive way. (Do this by using a Component Interface to access the database.)

For these reasons, PeopleSoft recommends that you first try to build a page in PeopleSoft Application Designer. This approach enables PeopleTools to generate all of the HTML in a cross-browser, multi-language, multi-market, and multi-currency sensitive way.

You don't have to use an iScript to generate an entire page. You can use the Request and Response objects with an HTML area, so you can develop a portion of your HTML using the iScript objects, while allowing the majority of it to be generated by the Component Processor. If you use this method, instead of using the Response object's Write and WriteLine methods to output your HTML, you set the value of the HTML area to the HTML that you want to display. Remember, that just as with an iScript, you must ensure that the HTML you generate is cross-browser compatible and multi-market sensitive.

So when *should* you use iScripts? Here are two scenarios where iScripts would be an appropriate choice:

- The page being developed *cannot* be built using PeopleSoft Application Designer. An example of this is a page that requires more than one HTML form. PeopleSoft Pure Internet Architecture places the entire page

Copyright © 1988-2007, Oracle. All rights reserved.

inside of a single form tag, so no other HTML form tags can be added. In this case the requirements of the page can't be met by pages created in PeopleSoft Application Designer, so use iScripts instead.

Note. You should use Component Interfaces for all database access so you have multi-language and multi-currency sensitivity.

- The page being developed never accesses the database. Using a page and Component Processor for this type of page incurs unnecessary processing overhead. An example of this is a page that talks to another website and redisplays HTML from the remote site.

See *PeopleSoft PeopleBook: Enterprise Portal*

Style Sheets and Styles

PeopleSoft recommends that developers using the iScripts always use styles (also known as classes) defined in the style sheets to specify the attributes (that is, background color, font, size, alignment, borders, and so on) of objects referenced in the iScripts. The Response object provides access to Style Sheets stored in the PeopleSoft database.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Style Sheet Definitions"

Other Considerations

PeopleSoft does not recommend using the following technologies in iScripts:

- Java applets
- Browser plug-ins

Details of an iScript URL

Viewing an iScript requires the assembly of a URL with the following pieces:

Section	Description
http://Server/	The scheme (http / https) and web server name.
servlet_name/	The name of the physical servlet that the web server invokes to handle the request. This is either psp or psc.
SiteName/	The user-defined site name. This is defined during the installation of PIA. This enables you to set up multiple sites on one physical web server.

 Copyright © 1988-2007, Oracle. All rights reserved.

Section	Description
PortalName/	The name of the portal to use for this request. The portal object contains metadata that describes how to present the content (that is, template, pagelets and so on.)
NodeName/	The name of the node that contains the content for this request.
content_type/	The type of the content for this request. For iScripts, this is "s".
content_id	The identification of the content. This, and the type is the unique key to the content being retrieved.
?content_parm	The query string parameters (name value pairs) for the content.

For an iScript, the *content_id* has the following form:

```
Record.Field.Event.Function
```

The following is an example of the call to an iScript:

```
http://mlee2038//psp/PS84/e_procurement/fdm/s/WEBLIB_Portal.PORTAL_HEADER.Field⇒
Formula.IScript_UniHeader_PIA
```

The Generate Functions

There are many different types of content on the portal, such as components, iScripts, homepages, and so on. Sometimes you want to access this content directly, through the URL that points to this content. To do this, PeopleSoft provides several Generate functions that generate the URL for the specified content.

The Generate functions can be categorized as:

- Content type.
- Absolute or relative URL.
- Portal service or content servlet (psp or psc).

The following is a list of the functions, plus a description of what to use them for. More information about a function is found under that function's description in the section PeopleCode Built-in Functions.

Copyright © 1988-2007, Oracle. All rights reserved.

Function Name	Content Type	Absolute or Relative URL	Portal service or content servlet
GenerateActGuideContentUrl	Activity Guide	Absolute	psc
GenerateActGuidePortalUrl	Activity Guide	Absolute	psp
GenerateActGuideRelativeUrl	Activity Guide	Relative	NA
GenerateComponentContentURL	Component	Absolute	psc
GenerateComponentContentRelURL	Component	Relative	psc
GenerateComponentPortalURL	Component	Absolute	psp
GenerateComponentPortalRelURL	Component	Relative	NA
GenerateComponentRelativeURL	Component	Relative	NA
GenerateExternalPortalURL	External	Absolute	psp
GenerateExternalRelativeURL	External	Relative	NA
GenerateHomepagePortalURL	Homepage	Absolute	psp
GenerateHomepageRelativeURL	Homepage	Relative	NA
GenerateQueryContentURL	WQuery	Absolute	psc
GenerateQueryPortalURL	Query	Absolute	psp
GenerateQueryRelativeURL	Query	Relative	NA
GenerateScriptContentURL	IScript	Absolute	psc
GenerateScriptContentRelURL	IScript	Relative	psc
GenerateScriptPortalURL	Iscript	Absolute	psp

Copyright © 1988-2007, Oracle. All rights reserved.

Function Name	Content Type	Absolute or Relative URL	Portal service or content servlet
GenerateScriptPortalRelURL	IScript	Relative	psp
GenerateScriptRelativeURL	IScript	Relative	NA
GenerateWorklistPortalURL	Worklist	Absolute	psp
GenerateWorklistRelativeURL	Worklist	Relative	NA

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions"

Error Handling

All errors are handled through the session object. You should check the PSMessages collection to see if there are any errors in your code.

Note. `Exit(1)` does *not* rollback iScript transactions. To rollback in an iScript, you can use the SqlExec built-in function with the parameter of ROLLBACK (`SQLEXEC("ROLLBACK")`) or the MessageBox built-in function with a message error severity of error. You can also use the built-in function Error, but only if you are not sending HTML or XML in the error text itself.

See Also

Chapter 36, "Session Class," Error Handling, page 1512

Scope of the Internet Script Classes

The Request, Response, and Cookie classes can be accessed only from PeopleCode. They can be used only as part of a PeopleSoft Pure Internet Architecture application—either as part of a page created in PeopleSoft Application Designer or an iScript. An iScript is always contained in a WEBLIB record. You wouldn't use any of these objects in an Application Engine program, a Component Interface, and so on. However, an iScript could call and start a Component Interface, a Business Interlink, and so on.

To access the Request or Response object, use the %Request and %Response system variable. These variables are objects, so you use them with dot notation.

```
%Response.SetContentType("text/HTML");

If %Request.GetParameter("encode") <> "" Then
```

Copyright © 1988-2007, Oracle. All rights reserved.

Data Types of the iScript Classes

Cookie objects are instantiated from Response object. You can declare the cookie object as data type cookie before any functions. You can also declare variables of type Response and Request, to assign the %Response and %Request system variables to local variables.

```
Local Cookie &Cookie;
Local Response &Resp;
Local Request &Req;

Function IScript_SetCookie()
    rem set a cookie called "MyCookie" to store my user id in it's value.
Make the cookie expire when the user's browser session expires;

   &Resp = %Response;
   &Req = %Request;
   &Cookie = &Resp.CreateCookie("MyCookie");
   &Cookie.Value = %UserId;
   &Cookie.MaxAge = -1;

End-Function;
```

Internet Script Classes

The following is a description of the PeopleCode classes, methods, and properties that you use to create an iScript.

Request Class

The Request object encapsulates all information from the request issued from the browser. This includes the URI, Query String, QueryString parameters, Cookies, and Headers. The properties for the Request object are read-only.

Parameters

Request parameters are name-value pairs sent by the client to the web server as part of an HTTP request. They include the pairs in the query string part of the URL, and data posted in a form, if the request was a post request. Multiple parameter values can exist for any given parameter name. The following methods are available to access parameters:

- GetParameter
- GetParameterNames
- GetParameterValues

The GetParameterValues method returns an array of String objects containing all the parameter values associated with a parameter name. The value returned from the GetParameter method equals the first value in the array of String objects returned by GetParameterValues.

 Copyright © 1988-2007, Oracle. All rights reserved.

All form data from both the query string and the post body are aggregated into the request parameter set. The order of this aggregation is that query string data appears before post body parameter data.

Response

The response object encapsulates all the information to be sent back to the browser. This includes the body of the response, content type, response headers, and cookies.

The response object also includes helper methods that retrieve HTML, Image, StyleSheet, and other objects from the database and move them to the web server, and returning strings that can be used to reference these objects in the response content.

The body of the response is created using the response object's Write and WriteLine methods. If the content type of the response is not explicitly set in the PeopleCode program, it defaults to "text/html".

Cookies

Cookies are data sent from the client to the server on every request that the client makes. Cookies are stored by the browser, either on disk or in memory, and returned to the server that originally set the cookie.

The Internet Script API makes available the name and value of each cookie that's sent with the request, through the Request object's GetCookieNames and GetCookieValue methods. The API also enables you to set new cookies (or alter existing cookies) through the Response object's CreateCookie method and the Cookie object's properties.

Internet Script Classes Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateActGuideContentUrl

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateActGuidePortalUrl

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateActGuideRelativeUrl

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentContentRelURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentContentURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentPortalRelURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentPortalURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateComponentRelativeURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateExternalPortalURL

Copyright © 1988-2007, Oracle. All rights reserved.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateExternalRelativeURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateHomepagePortalURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateHomepageRelativeURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateQueryContentURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateQueryPortalURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateQueryRelativeURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptContentRelURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptContentURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptPortalRelURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptPortalURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateScriptRelativeURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateWorklistPortalURL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateWorklistRelativeURL

Request Class Methods

In this section, we discuss the Request class methods. The methods are discussed in alphabetical order.

GetContentBody

Syntax

```
GetContentBody()
```

Description

This method retrieves the text content of an XML request. This is part of the incoming Business Interlink functionality, which enables PeopleCode to receive an XML request and return an XML response.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A string.

Example

The following example gets the XML text content of a request, then uses that as input to create a BiDoc. Then the BiDoc methods GetDoc and GetValue are used to access the value of the skills tag.

```
Local BIDocs &rootInDoc, &postreqDoc;
Local string &blob;
Local number &ret;

&blob = %Request.GetContentBody();
/* process the incoming xml(request)- Create a BiDoc */
&rootInDoc = GetBiDoc(&blob);
&postreqDoc = &rootInDoc.GetDoc("postreq");
&ret = &postreqDoc.GetValue("skills", &skills);
```

See additional Business Interlink documentation available on PeopleSoft Customer Connection.

GetCookieNames

Syntax

GetCookieNames()

Description

This method returns an array containing names of all the cookies present in this request. If there are no cookies in the request, an empty array is returned. When cookies are present, this method returns an array of strings.

GetCookieValue

Syntax

GetCookieValue (*name*)

Description

This method returns a string containing value of the cookie identified *name*. The *name* parameter takes a string value. The NPmatch between *name* and the request cookie is case-insensitive. If there is no cookie in the request matching the name, an empty string is returned.

GetHeader

Syntax

GetHeader(*name*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

This method returns the value of the header requested by the string *name*. The match between *name* and the request header is case-insensitive. If the header requested does not exist, an empty string is returned.

Example

The following example gets the "Referer" header to see where the request came from:

```
&Referer = %Request.GetHeader("Referer");
```

GetHeaderNames

Syntax

GetHeaderNames()

Description

This method returns an array of Strings representing the header names for this request.

GetHelpURL

Syntax

GetHelpURL(*HelpContext*)

Description

This method returns the help context path (as a string) to the help directory. It's used to construct a URL to a specific help page.

Example

```
&HelpURLString = %Request.GetHelpURL("hc0110");
```

GetParameter

Syntax

GetParameter(*name*)

Description

This method returns the value of a specified query string parameter or posted form data parameter. The match between *name* and the request parameter is case-insensitive. If there are multiple parameter values for a single name, the value returned is the first value in the array returned by the GetParameterValues method. If the parameter has (or could have) multiple values, you should use the GetParameterValues method in your Internet scripts.

See Also

Chapter 18, "Internet Script Classes (iScript)," GetParameterValues, page 599

 Copyright © 1988-2007, Oracle. All rights reserved.

GetParameterNames

Syntax

```
GetParameterNames()
```

Description

This method returns all the parameter names for this request as an array of Strings, or an empty array if there are no input parameters.

GetParameterValues

Syntax

```
GetParameterValues(name)
```

Description

This method returns the values of the specified parameter (*name*) as an array of Strings, or an empty array if the named parameter does not exist. The *name* parameter takes a string value.

Request Class Properties

In this section, we discuss the Request class properties. The properties are discussed in alphabetical order.

AuthTokenDomain

Description

This property returns the web server domain as a string across which the single signon authentication token is valid. Use this property as the domain of any cookie which you want to apply across the same domain as the single signon token.

The value of this property is in the format:

```
".domain.com"
```

Note that it begins with a dot "." character. You can use this format as the value of a cookie's domain property.

However, sometimes you cannot use a value that is prefixed with a dot character. For example, the javascript document.docmain property should not begin with a dot. In this case, you must strip it off manually in your code.

Note. The value of this property is the domain across which the authentication token is valid, set in the AuthTokenDomain configuration property in the configuration properties file. The value of the system variable %AuthenticationToken is the authentication token itself.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "System Variables," %AuthenticationToken

Copyright © 1988-2007, Oracle. All rights reserved.

AuthType

Description

This property corresponds to the CGI variable AUTH_TYPE. This variable does not necessarily contain a value, and may require special web server configuration to obtain non-empty values.

This property is read-only.

See Also

Chapter 18, “Internet Script Classes (iScript),” RemoteUser, page 604 and Chapter 18, “Internet Script Classes (iScript),” ServletPath, page 605.

ByPassSignOn

Description

This property returns a Boolean value, indicating what ByPassSignOn has been set to in the configuration properties file. If this value has been set to True, this property returns True, False otherwise.

This property is read-only.

BrowserPlatform

Description

This property returns a string containing the value the web server passes as the browser platform.

This property is read-only.

BrowserType

Description

This property returns a string describing the browser that sent the request.

This property is read-only.

BrowserVersion

Description

This property returns a string describing the version of the browser that sent the request.

This property is read-only.

ContentURI

Description

This property returns a string containing the portion of the current URI before the portal name, and referencing the content servlet (psc).

For the URL:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
http://localhost/psc/ps84/PORTAL/NODE/e_procurement/fdm/s/WEBLIB. . .
```

This property returns the following:

```
http://localhost/psc/ps84/
```

This property is read-only.

ExpireMeta

Description

This property returns the refresh meta-tag string that contains the cmd=expire parameter (the ExpireMeta string.)

The following is an example of the ExpireMeta string:

```
<meta HTTP-EQUIV='Refresh' Target='_top' CONTENT='10; URL=/servlets/psp/ps84/?cmd=⇒
expire'>
```

All Internet script pages should use this property to generate the meta tag to cause the page to expire. It is the same expiration tag used by pages originally created using PeopleSoft Application Designer, unless the Internet script generates a menu link in a separate frame that is not supposed to expire.

This tag should be included in the <head> section of the HTML generated by the iScript.

This property is read-only.

Example

```
%Response.Write("<html><head>");
%Response.Write(%Request.ExpireMeta);
%Response.Write("</head><body>");
```

FullURI

Description

This property returns the complete URI up to, but not including, the query string. This method returns a string value.

To return only the request URI (that is, without the scheme, server name or port) use the RequestURI method.

This property is read-only.

Example

From the following code

```
&FullURI = &Request.FullURI;
```

&FullURI might contain the following:

```
http://serverx/servlets/psp/ps84/Portal/Node/s/WEBLIB_TEST.SCRIPTS.Field⇒
Formula.IScript_Test
```

See Also

Chapter 18, "Internet Script Classes (iScript)," RequestURI, page 603

Copyright © 1988-2007, Oracle. All rights reserved.

HTTPMethod

Description

This property returns the HTTP method as a string, (for example, GET, POST, PUT) by which this request was made.

This property is read-only.

LogoutURL

Description

This property returns the complete URL (as a string) to logout of the PeopleSoft session. Use this property to generate a link that causes a page to logout. You should normally not need to include this link on your page.

This property is read-only.

PathInfo

Description

This property returns any path information following the servlet path, but prior to the query string. This property returns null if there is no path information following the servlet path.

This property is read-only.

Example

For the URL:

```
http:/localhost/psp/ps84/PORTAL/NODE/e_procurement/fdm/s/WEBLIB. . .
```

This property returns the following:

```
psp84/PORTAL/NODE/eprocurements/fdm/s/WEBLIB_. . .
```

Protocol

Description

This property returns the protocol being used for this request as a string in the following form:

```
protocol/major_version.minor_version
```

An HTTP 1.0 request, as defined by the HTTP 1.0 specification, should return the string HTTP/1.0. Use the Scheme property instead of the Protocol property when generating hrefs (links).

This property is read-only.

QueryString

Description

This property returns the query string present in the request URL, if any. A query string is defined as any information following a ? character in the URL.

 Copyright © 1988-2007, Oracle. All rights reserved.

If there is no query string, this method returns null.

Use the request Parameters methods (GetParameterNames, GetParameterValues, GetParameter) to get the values of individual parameters on the query string.

This property is read-only.

RelativeURL

Description

This property returns whether a relative URL should be generated for PeopleSoft Pure Internet Architecture pages. This property is set in the configuration properties file. This property takes a Boolean value: True if a relative URL is generated for PeopleSoft Pure Internet Architecture pages, False otherwise.

This property is read-only.

RemoteAddr

Description

This property returns the IP address of the agent that sent the request.

This property is read-only.

RemoteHost

Description

This property returns the fully qualified host name of the agent that sent the request.

This property is read-only.

RequestURI

Description

This property returns the URI, without the protocol.

This property is read-only.

Example

This example uses the following URL:

```
http://serverx/servlets/psc/ps84/PORTAL/NODE/fdm/s/WEBLIB_Portal. . .
```

From the following code

```
&MyRequestURI = &Request.RequestURI;
```

&MyRequestURI contains the following:

```
/servlets/psc/ps84/PORTAL/NODE/fdm/s/WEBLIB_PORTAL . . .
```

Use the following to build an absolute URL:

```
&myURL = &Request.Scheme | "://" | &Request.ServerName | ":" | &Request.Port | ⇒
&Request.RequestURI | "?" | &Request.QueryString;
```

Copyright © 1988-2007, Oracle. All rights reserved.

RemoteUser

Description

This property corresponds to the CGI variable REMOTE_USER. It is not specific to your PeopleSoft implementation.

This variable does not necessarily contain a value, and may require special web server configuration to obtain non-empty values.

This value is generally populated when J2EE Authentication is enabled in the web-server configuration. The specific implementation of your web-server dictates how exactly this variable is populated. See your web-server specific documentation for more information.

This property is read-only.

See Also

Chapter 18, "Internet Script Classes (iScript)," AuthType, page 600 and Chapter 18, "Internet Script Classes (iScript)," ServletPath, page 605.

Scheme

Description

This property returns the scheme, also known as the protocol, used by the request. Common schemes include http, https, ftp, and telnet. This property returns a string value.

This property is read-only.

Example

For the URL

```
http://serverx/servlets/psp/ps84/eprocurement/fdm/s/WEBLIB. . . .
```

the Scheme property returns:

```
http
```

ServerName

Description

This property returns the host name of the server on which the servlet is running. This property returns a string value.

This property is read-only.

Example

For the URL

```
http://serverx/servlets/psp/ps84/eprocurement/fdm/s/WEBLIB. . . .
```

the ServerName property returns:

```
serverx
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ServerPort

Description

This property returns an integer representing the port on which the browser's request was received (that is, the port on which the server is listening.)

This property is read-only.

Example

For the URL

```
http://servername:80/servlets/iclientservlet/peoplesoft8/?ICType=Panel. . .
```

the ServerPort property returns:

```
80
```

ServletPath

Description

This property corresponds to the CGI variable SCRIPT_NAME. This variable does not necessarily contain a value, and may require special web server configuration to obtain non-empty values.

This property is read-only.

See Also

Chapter 18, "Internet Script Classes (iScript)," AuthType, page 600 and Chapter 18, "Internet Script Classes (iScript)," RemoteUser, page 604.

Timeout

Description

This property returns an integer representing the timeout value, in seconds, set in the configuration.properties file.

This property is read-only.

Response Class

The response object encapsulates all information to be returned from the iScript to the browser.

Response Class Methods

In this section, we discuss the Response class methods. The methods are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

Clear

Syntax

```
Clear()
```

Description

This method removes all cookies and headers as well as deletes any HTML that has been written to the Response object. After using this method, the Response object appears as it did when the script was first called.

CreateCookie

Syntax

```
CreateCookie(name)
```

Description

This method adds a cookie to the response with the name specified by the string *name*. It returns a reference to the cookie object that is used to update the values of this cookie. If the cookie by the specified name already exists, a reference to the existing cookie is returned. This method can be called multiple times to set more than one cookie. Cookie names may not contain the "$" character.

Note. The web server only allows 20 cookies per server, and randomly throws away some cookies when you exceed that number. PeopleTools uses 8 cookies. In addition, the number of cookies used by PeopleTools may grow depending on the number of sites that are within the same domain. If you move from site to site, the number of cookies grows (session and homepage refresh cookies.) Some PeopleSoft applications use all the remaining cookies. You may or may not be able to create your own cookies to modify an existing PeopleSoft application. If you do create your own cookies, keep the use conservative, or your application may end up losing signon or authentication cookies.

GetCookie

Syntax

```
GetCookie(name)
```

Description

This method returns the cookie object specified by the string *name*. If the cookie is not already present in this response, a null value is returned.

GetCookieNames

Syntax

```
GetCookieNames()
```

Description

This method returns an array of strings that contains the names of all the cookies present in this response. If there are no cookies in the response, an empty array is returned.

Copyright © 1988-2007, Oracle. All rights reserved.

GetHeader

Syntax

GetHeader(*name*)

Description

This method returns the value of the response header requested by the string *name*. The match between *name* and the response header is case-insensitive. If the header requested does not exist, an empty string is returned.

GetHeaderNames

Syntax

GetHeaderNames()

Description

This method returns an array of Strings representing the header names for this response.

GetImageURL

Syntax

GetImageURL(**IMAGE.***ImageName* | *rowset.row.record*)

Description

This method returns a string which represents the URL of the requested image, for images in the database.

Because the image field is a long data type, you can have only one on the record. Therefore you must specify the record only. The field on the record is assumed to be the image.

Parameters

Parameter	Description
imagename	Specify the name of an existing image. This name must be prefaced with the reserved word **IMAGE.**
rowset.row.record	Specify the path to the image, that is, the rowset, row, and record.

Example

The following example gets the URL to use for the image, and the second uses the URL reference.

```
&strImage = %Response.GetImageURL(Image.ARROW);

%Response.WriteLine("<img src=" | &strImage | ">");
```

The following gets an image from the record.

```
%Response.GetImageURL(&rs.GetRow(1).GetRecord(Record.EMPL_PHOTO));
```

Copyright © 1988-2007, Oracle. All rights reserved.

GetJavaScriptURL

Syntax

GetJavaScriptURL(**HTML**.*HTMLname*)

Description

This method returns a string which represents the URL of the requested HTML definition, for JavaScript HTML definitions stored in the database.

Example

This example assumes the existence in the database of an HTML definition called "HelloWorld_JS", that contains some JavaScript.

```
Function IScript_TestJavaScript()

   %Response.WriteLine("<script src= " | %Response.GetJavaScriptURL(HTML.Hello⇒
World_JS) | "></script>");

End-Function;
```

In this example, the HTML definition is called FOO. The following PeopleCode example:

```
%Response.GetJavaScriptURL(HTML.FOO)
```

Resolves to something similar to:

```
http://myserver/cache/js/FOO_ENG_1.js
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating HTML Definitions"

GetStyleSheetURL

Syntax

GetStyleSheetURL(STYLESHEET.*stylename*)

Description

This method returns a URL string pointing to a style sheet created in PeopleSoft Application Designer.

Example

In the following example, the first line gets the URL to use for the style sheet, and the second is the standard HTML to include the style sheet in the HTML document. The stylesheet link must be included in the HTML *before* any of the styles in the stylesheet are used. It should be included in the <head> section of the HTML document.

```
&strStyleSheet = %Response.GetStyleSheetURL(StyleSheet.BENEFITS_STYLE);

%Response.WriteLine("<link rel=stylesheet href=" | &strStyleSheet | " type=""text⇒
/css"">");
```

Now to use a style, assign the class attribute of your choice to your HTML tags, as follows:

```
%Response.WriteLine("<p class=PSTEXT>I am some classy text!!</p>");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

RedirectURL

Syntax

RedirectURL(*location*)

Description

This method sends a temporary redirect response to the client using the location specified by the string location. The given location must be an absolute URL. No further output should be made by the iScript after calling this method.

Note. If your URL string contains special characters (such as foreign characters) make sure you encode it first using the EncodeURL function, such as, `%Response.RedirectURL(EncodeUrl(&Url))`.

In addition, while you can redirect relative URLs in certain circumstances, you *cannot* encode them.

Example

```
%Response.RedirectURL(&URL);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," EncodeURL

SetContentType

Syntax

SetContentType(*Type*)

Description

This method sets the content type for this response. The parameter *type* takes a string value. This type may later be implicitly modified by the addition of properties such as the MIME charset property if the service finds it necessary and the appropriate property has not been set.

The content type defaults to "text/html" if it is not set.

SetHeader

Syntax

SetHeader(*name, value*)

Description

This method sets a response header with the specified name and value. Both parameters take string values. If the field has already been set with a value, this new value overwrites the previous one.

Example

The PortalRegisteredURL response header identifies to the portal the registered URL of the current content, so that it can find the correct content reference to look at for the template.

You can override the value of the PortalRegisteredURL response header in a PIA Script or PIA Page by adding the header to the response yourself, like this:

Copyright © 1988-2007, Oracle. All rights reserved.

```
%Response.SetHeader("PortalRegisteredURL", &myURL);
```

You can do this to register the content with additional parameters.

UseSimpleURL

Syntax

UseSimpleURL({True | False})

Description

The UseSimpleURL method sets the response header that informs the portal that the current content uses relative simple URLs, and does not have to be proxied.

By default, all PeopleSoft Pure Internet Architecture content sets the header to true. You must to call this method to specify that your content does not use the simple URL format, and therefore does need to be proxied.

Parameters

Parameter	Description
True | False	Specify a Boolean value, either True or False, to indicate whether simple URLs are used.

Returns

None.

Write

Syntax

Write(*String*)

Description

This method prints *String* to the HTTP output stream.

You can use an HTML string from PeopleSoft Application Designer HTML catalog with the Write method if you also use the GetHTMLText function, as follows:

```
%Response.Write(GetHTMLTEXT(HTML.MY_HTML));
```

You can also use an XML string. The following example takes a BiDocs structure that contains an XML response and puts that into a text string. After this is done, the %Response.Write function can send this as an XML response.

```
Local BIDocs &rootDoc;
Local string &xmlString;

&xmlString = %Response.GetContentBody();
&rootDoc = GetBIDoc(&xmlString);

/* do processing */

&xmlString = &rootDoc.GenXMLString();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
%Response.Write(&xmlString);
```

WriteLine

Syntax

WriteLine(*String*)

Description

This method adds a carriage control and line feed to the end of the string *String,* then prints the string to the HTTP output stream.

You can use an HTML string from PeopleSoft Application Designer HTML catalog with the WriteLine method if you also use the GetHTMLText function, as follows:

```
%Response.WriteLine(GetHTMLTEXT(HTML.MY_HTML));
```

Response Class Properties

In this section, we discuss the Response class properties.

Charset

Description

This property returns the character set used by the response object as a string. This value is set in the user personalizations.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, "Managing PeopleSoft Personalizations"

DefaultStyleSheetName

Description

This property returns the name of the default style sheet defined on the PeopleTools Options page. If not defined, PSSTYLEDEF is returned.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," PeopleTools Options

Copyright © 1988-2007, Oracle. All rights reserved.

Cookie Class

The cookie class encapsulates all information to be sent to the browser for a single cookie. The cookie class is used by the response object to set new cookies. The request object uses a simple name-value pair mechanism for retrieving previously stored cookie values sent with the HTTP request.

Cookie Class Properties

In this section, we discuss the Cookie class properties. The properties are discussed in alphabetical order.

Domain

Description

This property returns the domain of this cookie, or null if not defined. This property is a string value.

This property is read-write.

Example

```
&cookie = &Response.AddCookie("My cookie", "My value");

&cookie.Domain = ".MyDomain.com";
```

MaxAge

Description

This property represents the maximum specified age of the cookie, as a signed number in seconds. The default value is -1. Setting the MaxAge property to a negative value ensures the cookie does not persist on the client when the client session ends. If the MaxAge property is set to zero, the cookie is deleted immediately from the client.

Value	Description
Non Negative Integer	Lifetime of the Cookie in seconds
0	Delete the cookie from the client immediately
-1	Default MaxAge property value (remove the cookie after the client exits)
Negative Integer	Remove the cookie after the client exits

This property is read-write.

Name

Description

This property returns the name of the cookie as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Path

Description

This property returns the prefix of all the URL paths for which this cookie is valid, or an empty string if not defined.

This property is read-write.

Secure

Description

This property specifies whether the cookie is to be secured. This property takes a Boolean value. The default value is False. Secure cookies are sent only if the client has established a secure link (such as HTTPS) with the server. This property is read-write.

Value

Description

This property returns the value of the cookie (as a string), or an empty string if it isn't defined.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 19

Java Class

This chapter provides an overview of Java class and discusses the following topics:

- Supported versions of Java.
- Naming standards for classes and packages.
- System setup for Java classes.
- From PeopleCode to Java.
- From Java to PeopleCode.
- PeopleCode and Java data types mapping.
- Considerations when using the PeopleCode Java functions.
- Error handling and the PeopleCode Java functions.
- The Java debugging environment.
- Java object declaration.
- Scope of a Java object.
- PeopleCode Java built-in functions.

Understanding Java Class

Using the PeopleCode Java functions, you can access Java classes, or create instances of Java objects. Calling a Java program from PeopleCode can greatly extend your application. Also, the PeopleCode integration with Java enables you to pass parameters such as record, record fields, or rowsets into Java programs.

The PeopleCode Java functions are:

- CopyToJavaArray
- CopyFromJavaArray
- CreateJavaArray
- CreateJavaObject
- GetJavaClass

See Also

http://java.sun.com/index.html

Copyright © 1988-2007, Oracle. All rights reserved.

Supported Versions of Java

PeopleSoft supports Java version 1.4.1 for all its platforms. The Java Runtime Engine (JRE) comes bundled with PeopleSoft for all but the z/OS platform. On that platform, you may have to install the JRE.

See your platform installation documentation for more information.

Naming Standards for Classes and Packages

PeopleSoft recommends following Sun standards for naming classes and packages.

Java classes delivered with PeopleSoft are generally in the following directory for Windows machines:

<PS_HOME>/appserv/classes

Java classes delivered with PeopleSoft are generally in the following directory for Unix machines:

<PS_HOME>\classes

At PeopleSoft, we use the following for the package hierarchy:

```
com.peoplesoft.<prodline>.<prodcode>
```

Where *prodline* is the company standardized product line code (like hrms) and *prodcode* is the product code unique within the product line (like hr).

In addition, if there are any classes that are common across products, they are placed into the common package for that product line, that is:

```
com.peoplesoft.<prodline>.common
```

See Also

http://java.sun.com/docs/codeconv/index.html

System Setup for Java Classes

If you only access the classes that come defined with PeopleSoft you don't need to do any additional setup.

If you include your JAR or class files in the Java directory you don't need to do any additional setup. That is, you don't need to do anything if your files are in the following directory:

```
<PS_HOME>\appserv\classes
```

If you create your own Java classes that you want to access with PeopleCode, and you do not store them in the Java directory, you *must* create a PS_CLASSPATH environment variable that points to where your classes are stored. The exact path depends on how and where you package your classes.

- If you package your Java classes in a JAR file, and you do *not* place your JAR file in the Java class directory, you must specify the JAR file name (with the JAR extension) in the PS_CLASSPATH environment variable.

 Copyright © 1988-2007, Oracle. All rights reserved.

Usually when there is a package hierarchy, the JAR file is at the package root, and contained within it a directory-like structure of the class files. For example, if the package hierarchy is like this:

```
c:\myjava\com\mycompany\myprod\xxx.java
```

Then you would have to set your PS_CLASSPATH environment variable to the following:

```
c:\myjava\com.mycompany.myprod.jar
```

Note. PeopleSoft recommends using the utility that comes with the Java SDK for creating JAR files.

- If you don't store the Java classes you create in a JAR file, and you do *not* place your class files in the Java directory, the PS_CLASSPATH environment variable must point to the top level of your package hierarchy.

  For example, suppose your Java classes were in the following package:

  ```
  c:\myjava\com\mycompany\myproduct\
  ```

  You would set the PS_CLASSPATH environment variable to:

  ```
  c:\myjava
  ```

Like most environment variables, you can specify more than one entry. On Windows, the PS_CLASSPATH entries are separated by semicolons. On Unix, they're separated by colons. The following is for Windows:

```
c:\myjava;d:\myjava\com\mycompany\myproduct\myjar.jar; . . .
```

The following would be for Unix:

```
/etc/myjava:/home/me/myjava/com/mycompany/myproduct/myjar.jar: . . .
```

This PS_CLASSPATH setting must be set for the process that runs the Java. For example, if Java is being called from PeopleCode that is running on the Application Server, it is the Application Server process that needs this setting. Similarly, if the Java is being run from PeopleCode in the Application Engine, that run of the Application Engine must have this setting.

Also when developing your own classes you must be aware that most Java virtual machines cache the class definitions. This means that even if you change the class files, a running Java virtual machine (inside, perhaps, an Application Server) that has loaded and is running the old version of the class files. It won't pick up the new version of the class files. You must restart the Application Server to make it reload the new classes.

See your system documentation for more information about setting up an environment variable.

From PeopleCode to Java

In this section, we provide an overview of state management concerns and present the following examples:

- State Management Concerns.
- CreateJavaObject example.
- CreateJavaArray example.
- GetJavaClass example.

Copyright © 1988-2007, Oracle. All rights reserved.

State Management Concerns

The application server is *stateless,* which means that it doesn't keep any information (state) for its clients between calls to it. For one reason, calls to the application server can use different actual servers for different calls. When you are using Java in the application server, be careful to not leave state in the Java virtual machine that would cause your application to fail if a different application server (which would use a different invocation of the Java virtual machine) was used for subsequent calls. One method to leave state in the virtual machine is to use static (class) variables.

Similar considerations to these apply using Java in Application Engine programs, though here the difficulty arises when you try to checkpoint and then restart the program. The restart starts with a Java virtual machine invocation that doesn't have any of the state you might have stored into the Java virtual machine before the checkpoint.

JavaObject variables cannot have Global or Component scope because of this lack of ability to save the state of these objects.

An example of this is issuing messages. When you're running with PeopleSoft Pure Internet Architecture and issue a message, the message is produced by an end-user action, so the Application Server gathers up its state to return it to the browser. This state saving attempts to save the current PeopleCode execution state, causing it to issue an error because of the JavaObject.

The solution is to not have any non-Null JavaObject objects when the message is issued.

The following is a simple Java program:

```
public class PC_Java_Test{

public String pcTest(){

   String message;

   message = "PeopleCode is successfully executing Java.";

   return message;
   }
}
```

Here is the PeopleCode that calls this Java program. Note that the JavaObject is set to NULL before the message is issued.

```
&java_test = CreateJavaObject("PC_Java_Test");
&java_message = &java_test.pcTest();
&java_test = Null;
WinMessage(&java_message);
```

In SavePreChange, Workflow, or SavePostChange PeopleCode the situation is more complicated. Usually messages with a zero style parameter (no buttons other than OK and perhaps Explain, therefore no result possible except OK) are queued up by the Application Server. They are output by the browser when the service completes, so the serialization won't happen until after the PeopleCode has finished, so you won't have to set your JavaObject to NULL. With other kinds of messages, you must do this.

 Copyright © 1988-2007, Oracle. All rights reserved.

CreateJavaObject Example

The following is an example program creating a Java object from a sample program that generates a random password.

```
/* Example to return Random Passwords from a Java class */
Local JavaObject &oGpw;

/* Create an instance of the object */
&oGpw = CreateJavaObject("com.PeopleSoft.Random.Gpw_Demo");

&Q = "1";

/* Call the method within the class */
&NEW_VALUE = &oGpw.getNewPassword(&Q, PSRNDMPSWD.LENGTH);

/* This is just returning one value for now */
PSRNDMPSWD.PSWD = &NEW_VALUE;
```

CreateJavaArray Example

Suppose we had a PeopleCode array of strings (&Parms) that we wanted to pass to a Java method xyz of class Abc. This example assumes that you don't know when you write the code just how many parameters you will have.

```
Local JavaObject &Abc, &RefArray;
Local array of String &Parms;

&Parms = CreateArray();

/* Populate array how ever you want to populate it */

&Abc = GetJavaObject("com.peoplesoft.def.Abc");

/* Create the java array object. */

&JavaParms = CreateJavaArray("java.lang.String[]", &Parms.Len);

/* Populate the java array from the PeopleCode array. */

&RefArray = GetJavaClass("java.lang.reflect.Array");

For &I = 1 to &Parms.Len
   &RefArray.set(&JavaParms, &I - 1, &Parms[&I]);
End-For;

/* Call the method. */
&Abc.xyz(&JavaParms);
```

Copyright © 1988-2007, Oracle. All rights reserved.

GetJavaClass Example

The following example gets a system class.

```
&Sys = GetJavaClass("java.lang.System");
&Sys.setProperty("java.security.policy", "C:\java\policy");

WinMessage("The security property is: " | &Sys.getProperty("java.security.policy"));

&Props = &Sys.getProperties();
&Props.put("java.security.policy", "C:\java\policy");
&Sys.setProperties(&Props);

WinMessage("The security property is: " | &Sys.getProperty("java.security.policy"));
```

From Java to PeopleCode

The Java classes delivered with PeopleTools enable you to call PeopleCode from your Java program. Calling into PeopleCode works only from Java code that you have initially called from PeopleCode.

You must call PeopleCode facilities only from the same thread that was used for the call into Java. PeopleTools is not multithreaded.

You cannot call any PeopleCode facility that would cause the server to return to the browser for an end-user action, because the state of the Java computation cannot be saved and restored when the action is complete.

See Also

Chapter 19, “Java Class,” Considerations When Using the PeopleCode Java Functions, page 625

SysVar Java Class

Use the SysVar Java Class to refer to System Variables, such as %Language or %DBType.

For example, %Session, becomes SysVar.Session()

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “System Variables”

SysCon Java Class

Use the SysCon Java Class to refer to system constants, such as %SQLStatus_OK or %FilePath_Absolute.

For example, %CharType_Matched becomes SysCon.CharType_Matched.

Func Java Class

Use the Func Java Class to refer to built-in functions, such as CreateRowset or GetFile.

For example, SetLanguage(LANG_CD) becomes Func.SetLanguage(LANG_CD)

 Copyright © 1988-2007, Oracle. All rights reserved.

Name Java Class

The Name Java Class enables you to use the PeopleSoft reserved item references. This enables you to reference pages, components, records, fieldnames, and so on.

For example, in PeopleCode you can refer to a record field using the following:

```
recname.fieldname
```

With the Name class, you can use a similar construct:

```
new PeopleSoft.PeopleCode.Name("RECNAME", "FIELDNAME");
```

Note that these must be in the exact case as the item. As all PeopleTools items are named in uppercase, that means you must use uppercase.

As another example, in PeopleCode you can refer to a page using the following:

```
PAGE.pagename
```

In Java, it would be:

```
new PeopleSoft.PeopleCode.Name("PAGE", "PAGENAME");
```

Accessing PeopleCode Objects

The existing PeopleCode classes (like Array, Rowset, and so on) have properties and methods you can access.

- PeopleCode classes have the same names, so Record becomes Record, SQL becomes SQL, and so on.
- Methods are accessed by the method name.
- The name of a property is pre-pended with either **get** or **set**, depending on whether you're reading or writing to the property.

  For example, to get the IsChanged property would be getIsChanged. To set the value for a field would be &MyField.setValue.

Here is an example of a Java program that uses PeopleCode objects to access the database:

```
/*
 * Class Test
 *
 * This code is used to test the Java/PeopleCode interface.
 *
 */

import PeopleSoft.PeopleCode.*;

public class Test {

/*
 * Test
 *
 * Add up and return the length of all the
 * item labels on the UTILITIES menu,
 * found two different ways.
 *
 */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
public static int Test() {
/* Get a Rowset to hold all the menu item records. */
Rowset rs = Func.CreateRowset(new Name("RECORD", "PSMENUITEM"), new Object[]{});
String menuName = "UTILITIES";
int nRecs = rs.Fill(new Object[]{"WHERE FILL.MENUNAME = :1", menuName});

int i;
int nFillChars = 0;
for (i = 1; i <= rs.getActiveRowCount(); i++) {
String itemLabel = (String)rs.GetRow(i)
.GetRecord(new Name("RECORD", "PSMENUITEM"))
.GetField(new Name("FIELD", "ITEMLABEL"))
.getValue();
nFillChars += itemLabel.length();
}

/* Do this a different way - use the SQL object to read each menu
   item record. */

int nSQLChars = 0;
Record menuRec = Func.CreateRecord(new Name("RECORD", "PSMENUITEM"));
SQL menuSQL = Func.CreateSQL("%SelectAll(:1) WHERE MENUNAME = :2",
new Object[]{menuRec, menuName});

while (menuSQL.Fetch(new Object[]{menuRec})) {
String itemLabel = (String)menuRec
.GetField(new Name("FIELD", "ITEMLABEL"))
.getValue();
nSQLChars += itemLabel.length();
}

return nFillChars + 100000 * nSQLChars;
}
}
```

This can be run from PeopleCode like this:

```
Local JavaObject &Test;
Local number &chars;

&Test = GetJavaClass("Test");
&chars = &Test.Test();

&Test = Null;
WinMessage("The character counts found are: " | &chars, 0);
```


Copyright © 1988-2007, Oracle. All rights reserved.

Using Application Classes From Java to PeopleCode

You call a Java program from an Application Class the same way you do using any other PeopleCode program, that is, by using one of the existing Java class built-in functions.

Calling an Application Class from a Java program has the following considerations:

- Application Classes must be accessed using the object built-in functions, such as CreateObject, ObjectDoMethod, ObjectGetProperty, and so on.
- You cannot declare a variable of type HR.Package.SomeClass in your Java program. The variable must be of type Object.
- There is no pre-pending the word ‘get’ or ‘set’ for properties. All classes, methods, and properties are passed as strings.

The following is an example of how to call an Application Class from a Java program.

This is the Java program:

```
package com.peoplesoft.pcode;
import PeopleSoft.PeopleCode.*;

public class foo {

  public foo() {
  }
  public String getString() {

    Object foo = Func.CreateObject("GTP:Foo", new Object[]{});
    return (String)Func.ObjectDoMethod((Peer)foo, "GetString", new Object[]{});
  }
}
```

The following is the Application Class Foo, in the Application Package Foo:

```
class Foo
   method GetString() Returns string;
end-class;

method GetString
   /+ Returns String +/
   Return "Hello";
end-method;
```

The following is the PeopleCode program that starts it all:

```
Local JavaObject &foo = CreateJavaObject("com.peoplesoft.pcode.foo");
GTP_PARSER.GTP_STR_RESULT = &foo.getString();
```

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleCode and Java Data Types Mapping

The following table describes the matching of types for resolution of overloaded Java methods and basic conversions. The first Java Type/Class is the one that is produced in the absence of any other type of information.

PeopleCode Type	Java Type/Class
Float	double, float
Number	double, float, byte, char, short, int, long
Integer	int, byte, char, short, long
Boolean	Boolean
String	java.lang.String
Date	java.sql.Date
Time	java.sql.Time
Date Time	java.util.Date
any kind of object	any kind of object

The following table represents the conversions done to produce the Java class java.lang.Object. In addition to these, the conversions (listed in the previous table) from String onwards are done to produce a java.lang.Object.

PeopleCode Type	Java Type/Class
Float, Number	java.lang.Double
Integer	java.lang.Integer
Boolean	java.lang.Boolean

The following table represents other conversions that are done as required by the signature of a Java method or constructor.

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleCode Type	Java Class
Integer, Number, Float	java.lang.Integer, java.lang.Byte, java.lang.Character, java.lang.Short, java.lang.Long, java.lang.Float, PeopleSoft.PeopleCode.intHolder, PeopleSoft.PeopleCode.doubleHolder
String	PeopleSoft.PeopleCode.StringHolder
peoplecode builtin class Xxx	PeopleSoft.PeopleCode.Xxx
JavaObject	corresponding Java object

Considerations When Using the PeopleCode Java Functions

Some PeopleCode built-in functions can't be called from a Java program. Many of these restrictions arise because you can't serialize Java objects. Inside Java, you can't serialize and save the state of the Java Virtual Machine (JVM.)

This means that you cannot call the following built-in functions:

- Think-time functions, such as DoCancel, Prompt, RevalidatePassword, and so on.
- Math functions, such as Abs, Cos, and Sin. Use the Java math functions instead.
- Java class functions, such as CreateJavaArray, CreateJavaObject, or GetJavaClass.
- Deprecated functions, such as SetNextPanel, PanelGroupChanged, and so on. Use the new function instead, like SetNextPage, ComponentChanged, and so on.
- PeopleCode language elements, such as Exit, Return, If, and so on.
- Cursor position, such as SetCursorPos.
- Functions that rely on specific character encoding, such as char, code, exact, codeb, and so on.
- Menu appearance functions, such as CheckMenuItem, HideMenuItem, and so on.
- Transfer functions such as Transfer or TransferPage.
- DoSaveNow isn't allowed. However, DoSave is.

When you're creating your Java program, keep the following points in mind:

- If you're starting from an online application, avoid calling anything that will wait a long time.
- If you use third-party Java that require third-party platforms, you are limiting your program to just those platforms.

When setting a null date in Java, use the following:

```
myField.SetValue("");
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Think-Time Functions

Copying Arrays of Data Between PeopleCode and Java

When PeopleCode is called from your Java program it executes what are called native methods. These methods looks like regular Java methods in their definition but are implemented in the PeopleTools layer. In order to go from Java into PeopleTools you have to use an interface called the Java Native Interface (JNI). There is a cost associated with each transition across the JNI. Using the CopyToJavaArray and the CopyFromJavaArray functions may improve your performance, as they act as a type of bulk data copy that minimizes the transition overhead.

For example you could copy an array by copying each element using the array's Get method. However, that would require traversing the JNI twice for each element; once going into Tools and once coming back from Tools. Not only is there transition overhead but there is also conversion between object types. For example a PeopleCode string has to be converted to a Java String and vice-versa. While these two builtins do not eliminate the latter conversions, they minimize the number of transitions across the JNI.

These functions can be used when you are selecting data into a PeopleCode array and you want to copy that data into a Java array.

These functions also allow you to supply an optional parameter that specifies the list of items you want copied.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyFromJavaArray

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyToJavaArray

Considerations Working with the Java Garbage Collector

Memory is managed in the JRE by the Java garbage collector. Because PeopleTools has no control over when the Java garbage collector runs (apart from a user in their Java program calling it using `system.gc()`), PeopleTools interacts dynamically with the Java runtime garbage collector. In particular, PeopleCode objects created from a Java program (such as records, fields, and so on,) are in effect peer objects of real PeopleCode objects in PeopleTools. The release of these objects must be linked. Using the weak reference mechanism in Java, PeopleTools dynamically interracts with the Java garbage collector each time the execution thread passes through the Java Native Interface(JNI.) This allows long-running Java applications in a PeopleTools context to function without having to wait for an "end of service" event for object cleanup.

There might be occasions where a Java application creates many application classes and wants to force the PeopleTools garbage collector to run. That can be achieved by calling the CollectGarbage function. This is not normally necessary.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CollectGarbage

Error Handling and the PeopleCode Java Functions

Java functions throw exceptions to indicate something unusual has happened. However, all exceptions from Java called by PeopleCode are turned into fatal errors. You can catch these exception by enclosing the call to Java in a Try-Catch PeopleCode block. If you do not try to catch these exception, the PeopleCode program is terminated, and the user transaction must be canceled.

PeopleSoft recommends that you write a Java wrapper to handle errors.

Use either the All or None built-in functions to check values that are returned if you think you may call a Java method that is defined to return a string, but returns a Null object reference instead. Java Null object references are automatically converted into PeopleCode Null object references.

Accessing the Application Log File

For doing additional error handling in your application, you can access the application log file using the PeopleCode WriteToLog built-in function. For example:

```
Func.WriteToLog(SysCon.ApplicationLogFence_Level1, myLogString);
```

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," WriteToLog.

The Java Debugging Environment

To use Sun's JPDA V1.0 debugging architecture, you must do the following. These instructions are general: how you actually set up the debugger depends on your system.

To use the Java Debugging Environment (JDB):

1. Download and install a copy of JPDA V1.0.
2. Set the path for your system.

   Suppose you install JPDA V1.0 in C:\jpda. Set your PATH environment variable to include C:\jpda\bin.
3. Set the path for the application server.

   For the application server, set the Domain Settings/Add to PATH to include C:\jpda\bin.
4. Set the JavaVM Options.

   Set the JavaVM Options to be something like the following (see the JPDA documentation for a more complete example):

   ```
   -Xdebug -Djava.compiler=NONE -Xnoagent -Xrunjdwp:transport=dt_socket,suspend=⇒
   n,address=8765,server=y
   ```

Copyright © 1988-2007, Oracle. All rights reserved.

5. Run the debugger.

   After starting the tools session and causing it to start the JVM, you can use the JDB command line debugger that comes with JPDA, using a command like the following:

   ```
   jdb -connect com.sun.jdi.SocketAttach:port=8765
   ```

   You can also use the (no cost) Forte for Java Community Edition IDE from Sun or any of the Java IDEs noted on the JPDA pages.

See Also

http://java.sun.com/products/jpda

http://www.sun.com/forte/ffj/ce/index.html

Data Type of a Java Object

You should declare a Java object as type JavaObject. For example:

```
Local JavaObject &MyJavaClass;
```

Note. Java objects can be declared as type Local only.

Scope of a Java Object

A Java object can be instantiated from PeopleCode only. This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

PeopleCode Java Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyFromJavaArray

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CopyToJavaArray

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateJavaArray

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateJavaObject

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetJavaClass

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 20

Mail Classes

This chapter provides an overview of PeopleSoft MultiChannel Framework mail classes and discusses how to:

- Import mail classes.
- Create a mail object.
- Use the mail classes.
- Retrieve email from a mail server with the MCFGetMail class.
- MCFMail reference.

Understanding PeopleSoft MultiChannel Framework Mail Classes

This section discusses:

- PeopleSoft MultiChannel Framework mail classes.
- Scope of the mail classes.
- Data types of the mail classes.

PeopleSoft MultiChannel Framework Mail Classes

The PT_MCF_MAIL application package contains the following classes:

- MCFBodyPart
- MCFEmail
- MCFGetMail
- MCFHeaders
- MCFInboundEmail
- MCFMailStore
- MCFMultiPart
- MCFMailUtil
- MCFOutboundEmail
- MCFPart
- SMTPSession

Copyright © 1988-2007, Oracle. All rights reserved.

Use the mail classes to develop applications to create and send emails, fetch and delete emails on the mail server, manage emails in your PeopleSoft database, and access attachments . By using the mail classes, you are not required to know the details of the GETMAILTARGET connector or the database schema.

Scope of the Mail Classes

The mail classes can be instantiated only from PeopleCode.

The mail classes can be called from a component, an internet script, or a PeopleSoft Application Engine program.

Mail classes can be of local, global, or component scope.

Data Types of the Mail Classes

Every mail class is its own data type; that is, get mail objects are declared as data type MCFGetMail, store mail objects are declared as type MCFMailStore, and so on.

The following are the data types of the mail classes:

- MCFBodyPart
- MCFEmail
- MCFGetMail
- MCFHeaders
- MCFInboundEmail
- MCFMailStore
- MCFMailUtil
- MCFMultiPart
- MCFOutboundEmail
- MCFPart
- SMTPSession

Importing Mail Classes

The mail classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or a particular application class.

The application package PT_MCF_MAIL contains the following classes:

- MCFBodyPart
- MCFEmail
- MCFGetMail
- MCFHeaders
- MCFInboundEmail

 Copyright © 1988-2007, Oracle. All rights reserved.

- MCFMailStore
- MCFMailUtil
- MCFMultiPart
- MCFOutboundEmail
- MCFPart
- SMTPSession

To import all the mail classes, use the following import statement:

```
import PT_MCF_MAIL:*;
```

Using an asterisk after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available. The PT_MCF_MAIL application package contains no subpackages.

See Also

Chapter 6, “Application Classes,” page 153

Creating a Mail Object

After you’ve imported the mail classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of the MCFGetMail class, as the variable &gm, with local scope:

```
Local MCFGetMail &gm = create MCFGetMail();
```

Using the Mail Classes

This section discusses how to use the mail classes, and includes the following:

- General lifecycle of an outbound email
- Delivery status notifications and return receipts
- Priority
- MCFBodyPart and MCFEmail considerations

General Lifecycle of an Outbound Email

The following are the general steps used to create an outbound email:

1. Create an MCFOutboundEmail object
2. Set the usual properties for an email, such as the From, Subject, Recipients, and Text properties.
3. If necessary, set the parts of the email using MCFMultipart and MCFBodyPart classes.
4. Use the Send method to send the email.

Copyright © 1988-2007, Oracle. All rights reserved.

The MCFPart and MCFEmail classes provide the common functionality for the MCFBodyPart and MCFOutboundEmail classes, respectively. In general, you never need to use the MCFPart or MCFEmail classes directly in your PeopleCode programs. However, in rare cases you may use these classes if you want to extend the application package.

The SMTPSession class encapsulates all the session parameters that can be set in an SMTP session. Each MCFOutboundEmail object points to one SMTPSession object. In most cases, the parameters for the SMTP session are set from values found in the application server configuration file. However, you can also set these parameters by creating an SMTPSession object, setting the desired parameters, then creating an MCFOutboundEmail object.

For convenience, all SMTP attributes are also directly accessible from the MCFOutboundEmail class, so you don't need to create an SMTPSession object when you're sending just one email.

When you want to send many emails in a single SMTP session, you don't have to set the SMTP session parameter in each and every email. For this case, you can create a single SMTPSession, using either the default settings or explicitly setting the desired parameters, then use the CreateOutboundEmail method of SMTPSession object to create all the MCFOutboundEmail objects. This way all the MCFOutboundEmail objects use the same SMTP session parameters. After specifying the value of all the parameters for all the MCFOutboundEmail objects, use the SendAll method of SMTPSession object to send all the emails in one SMTP session.

All the application classes mentioned above hold all the data until one of the Send methods is called. The Send methods connect to the mail server using a JavaMail API that sends the mail.

Since all communication to the mail server happens when a Send method is called, errors are only reported when the Send method returns. The return value indicates the success or the failure of the send process. In case of failure, you can query the MCFOutboundEmail application class for the details of the error. The error details are available in properties such as MessageSetNumber, MessageNumber, ErrorDescription, ErrorDetails, and so on.

Delivery Status Notification and Return Receipts

There is an SMTP extension that supports Delivery Status Notifications (DSN) defined in RFC 1891. This may or may not be supported by the user's Mail Transport Agent (MTA). The acknowledgements do not indicate whether the user reads the message, they only acknowledge receipt of the message by their mail server. It is possible that the mail server throws the message away as spam and the end user never actually receives the email. To get such notifications, set the StatusNotifyOptions and StatusNotifyReturn properties, as in the following example:

```
&email.StatusNotifyOptions  = "SUCCESS,FAILURE";
&email.StatusNotifyReturn = "HDRS";
```

You can use the IsReturnReceiptReqd property to specify if a return receipt should be sent when the mail server receives and opens the email. This property only works if the underlying SMTP server supports this feature. The following is an example of using this property:

```
&email.IsReturnReceiptReqd = True;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Priority

There is no SMTP standard for message priority. Most email applications use an X−Priority value of 3 for messages with "normal" priority. More important messages have lower values and less important messages have higher values. In most cases, urgent messages have an X−Priority value of 1. The priority code used with the Mail classes, and set in the headers are as follows:

1. Highest Priority
2. High Priority
3. Normal
4. Low Priority
5. Lowest Priority

The default value is 3, that is, normal priority.

Different email programs render these different values in a variety of ways, usually using some kind of colored symbols or arrows.

Set the priority for an email using the Priority property. The following is an example:

```
&Email.Priority = 2;
```

MCFBodyPart and MCFEmail Considerations

Both MCFBodyPart and MCFEmail objects are used for both inbound and outbound email. The following sections specify which methods and properties are generally used for inbound, and which are used for outbound, for the different objects.

Inbound and Outbound MCFBodyPart Class Methods

All of the methods except the GetUnparsedHeaders method for the MCFBodyPart class are used with outbound email. The following methods are generally used with inbound email as well:

- GetHeader
- GetHeaderNames
- GetUnparsedHeaders

Inbound and Outbound MCFBodyPart Class Properties

The following properties are primarily used with inbound email. Note that some of these properties are also used with outbound email.

- AttachmentURL
- ContentType
- Filename
- IsAttachment
- Text

The following properties are primarily used with outbound email. Note that some of these properties are used with inbound email as well.

- ContentLanguage

Copyright © 1988-2007, Oracle. All rights reserved.

- ContentType
- Description
- Disposition
- FileName
- FilePath
- FilePathType
- Multipart
- Text

Inbound and Outbound MCFEmail Class Properties

The following properties are used with inbound email. Note that most of these properties are also used with outbound email.

- From
- RefIDs
- ReplyIDs
- ReplyTo
- Sender
- Subject

The following properties are used with outbound email. Note that many of these properties are also used with inbound email.

- BCC
- BounceTo
- CC
- From
- Importance
- Priority
- Recipients
- RefIDs
- ReplyIDs
- ReplyTo
- Sender
- Sensitivity
- Subject

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, “Mail Classes,” MCFBodyPart Class Methods, page 644

Chapter 20, “Mail Classes,” MCFBodyPart Class Properties, page 650

Chapter 20, “Mail Classes,” MCFEmail Class Properties, page 653

Retrieving Email From a Mail Server With the MCFGetMail Class

Use the MCFGetMail class to retrieve email from your mail server and load it in a rowset. After you have decided that an email needs to be saved to the PeopleSoft database, use MCFMailStore class.

This section covers the following topics:

- Rowset structure.
- Error messages common to all MCFGetMail methods.

Structure of Rowset Used for Email Information

The following list describes the different levels of the rowset that contains email information, and what each level contains.

Level 0 The first row at level zero is empty except for the number of rows contained within level zero and return status. The first row contains the return status for the entire batch of emails. Apart from the return status of the whole batch, each individual row has its own EMAIL_RET_STATUS. When a fatal error occurs, such as a connector size overflow, an IB overflow, or an error accessing attachments , the return status is copied to the first row. Subtle errors like unsupported encoding for a part of a individual email appear in the return status of that email only. Each row after the first row contains information about an individual email, such as the header information, the time received, and so on.

Level 1 Level one is where the parts of an email reside. The level one child rowsets are linked to the parent rowset in level zero by the MCF_EMAIL_ID, a unique ID assigned by PeopleSoft.

Note. Multipurpose Internet Mail Extensions (MIME) is an Internet standard for representing multipart and multimedia data in email. It enables email to be exchanged between different email systems. The parts may be nested. PeopleSoft embeds the Sun JavaMail API to implement support for the MIME standard. However, all parts of an email are represented in PeopleSoft as level one rowsets, regardless of the hierarchy that existed in the original email.

The following table describes the fields within the rowset at level zero.

Copyright © 1988-2007, Oracle. All rights reserved.

Field	Description
MCF_ATTACH_LIST	Contains a list of all the file names that are attachments for a particular email. Multiple items are separated by a semicolon (;).
MCF_IS_ATT_URL	This is a boolean value indicating whether the email body is located in the MCF_EMAIL_TEXT or in the attachment directory. If this value is 1 (true), the email body is located in the attachment directory and can be accessed using the MCF_ATT_URL value. If this value is 0 (false), then the email body is located in MCF_EMAIL_TEXT.
MCF_ATT_URL	This URL enables you to view an attachment. This URL is viewable only if the corresponding email has been authorized for the current user. This value is a relative URL. The URL becomes fully qualified only after the rowset has been saved to the PeopleSoft database and then later retrieved using the MCFMailStore class. The following is a sample URL: http:/ /sundance.peoplesoft.com/PSAttachServlet/ps/mcfdev1 /39297_31030554591_1.gif?contentType=image/gif *mcfdev1* is the mail user name. *39297__31030554591_1.gif* is the file name in the attachment directory. *contentType* is used by the browser to open a viewer associated with this type. The filename of the downloaded attachment is constructed by using the email's unique ID and the part number. ***Warning!*** Downloading the same emails twice from the mail server overwrites the associated files in the directory. If the same email is stored twice in the PeopleSoft database, the system creates two separate rows. If you delete either one of these rows, the system deletes the associated files in the directory. To avoid this situation, delete emails from the mail server after you have retrieved them. See Chapter 20, "Mail Classes," AuthorizeEmailAttach, page 676.

 Copyright © 1988-2007, Oracle. All rights reserved.

Field	Description
MCF_IBNODE_NAME	Identifies the name of the PeopleSoft Integration Broker node that received the email, such as MCF_GetMail. The system uses this value to construct the fully qualified URL required for viewing attachments.
MCF_ATTACH_SIZES	Indicates the size of the attachments associated with an email. Multiple values are separated by a semicolon (;). The attachment sizes appear in the same order as the file names in MCF_ATTACH_LIST.
MCF_EMAIL_FROM	Indicates the email account that sent the email, such as jsmith@company.com.
MCF_DTTM_RECV	Indicates the time that the mail server received the email. ***Warning!*** If you are using a Post Office Protocol 3 (POP3) mail server, this field may not be populated.
MCF_OFFSET_RECV	Indicates the time zone of the mail server that received this email. **Note.** This field contains the time zone of the integration gateway used to fetch this email.
MCF_DTTM_SENT	Indicates the time that the email was sent.
MCF_OFFSET_SENT	Indicates the time zone of the mail server that sent this email. **Note.** Currently, this field contains the time zone of the integration gateway used to fetch this email.
MCF_DTTM_SAVED	Records the date and time that the PeopleSoft system saved an email to the PeopleSoft database. If you use a POP3 mail server, PeopleSoft recommends using this value as the base, or starting point, for any time-sensitive transactions.

Copyright © 1988-2007, Oracle. All rights reserved.

Field	Description
MCF_EMAIL_SENDER	This is the reply to address. This supports a mail feature in which users can send an email from one account, but when the receiver chooses to reply, the system addresses the email to a different account. For instance, sales representatives may send an email from their personal email account, as in tom_sawyer@company.com, but when customers reply, the email is addressed to sales@company.com.
MCF_EMAIL_TEXT	Represents the text that was delivered in the body of the email. Before the email is downloaded, this field contains the body of the email. After the email has been downloaded, this field is blank. Check the MCF_IS_ATT_URL value to determine whether your email text resides in this field or the attachment directory. PeopleSoft relies on the originating email client embedding encoding information to determine character set. For example, Content-type: text/plain; charset=Big5
MCF_NOTIFY_CC	Indicates the parties who were copied (carbon copied) on the email. Multiple addresses are separated by a semicolon (;).
MCF_NOTIFY_TO	Indicates the parties who received the email. Multiple addresses are separated by a semicolon (;).
MCF_UID_LIST	This contains a unique identifier created by POP3 or an IMAP4 mail server. It is guaranteed to be unique within a particular mail box. **Note.** The DeleteEmail method also uses this value when deleting a list of emails. When deleting emails, this field can contain multiple values separated by a space.

 Copyright © 1988-2007, Oracle. All rights reserved.

Field	Description
MCF_WL_SUBJECT	Contains the subject of the email. A maximum of 254 characters is allowed. PeopleSoft relies on the originating email client embedding encoding information to determine character set. For example: `Email Subject: Subject: =⇒ ?Big5?B?uvuqwLBPqsy67qZYs/i+yS+kp⇒ LDqqXik6Ldztdg=?= =?Big5?B?uvSlSKTOp⇒ Eilwbr0MzCk6bVvqu2q+KTloUGkaqRP?==⇒ ?Big5?B?scCxUqa/v0GlwQ==?=` where ?Big5? is the prefixed encoding information.
MCF_RET_STATUS	Represents where the system stores the return status of a particular mail class method.
MCF_EMAIL_STATUS	This field is not currently used.
MCF_NUMROWS	Indicates the number of rows (level one child rowsets) that contain parts of a particular email. For example, if an email has two child rowsets associated with it at level one, this value would be 2. **Note.** For row one of the level zero rowset, this field reflects the number of emails in this rowset. If you fetch 10 emails, the rowset contains 11 rows, but the MCF_NUMROWS value in row 1 equals 10. Row 1 of the level zero rowset never contains an email.
MCF_EMAIL_MSG_ID	This is the globally unique identifier for this email. This ID is generated by the mail server that sent this email.
MCF_REPY_TO	This is the reply to field.
MCF_REF_IDS	Use this field for the reference header fields of the message.
MCF_REPLY_IDS	This value contains the email address that the email can reply to.
MCF_MSG_SIZE	Represents the total size of the entire email. This value includes the sizes of attachments.

Copyright © 1988-2007, Oracle. All rights reserved.

Field	Description
MCF_EMAIL_LANG_CD	This value reflects the language code of the PeopleSoft Integration Broker node that received this email. **Note.** PeopleSoft recommends setting up email accounts that are dedicated to one language such as, support_french@company.com and support_english@company.com. This enables you to anticipate the language of the email you read into your system. In turn, configure separate PeopleSoft Integration Broker nodes to handle each of the languages you support. There are no language recognition procedures within the PeopleSoft system.
MCF_USER	Represents the user account (mail box) on the mail server that received the email.
MCF_SERVER	Represents the mail server that received the email.
MCF_ERROR_COUNT	Used to return the number of messages that caused errors during processing on the target connector.
MCF_QUARANTINE_COUNT	Used to return the number of messages that caused errors during processing on the target connector and were successfully moved to the quarantine folder (This value is always 0 for POP3).
MCF_CONTENT_TYPE	This is the content type of the email itself. For example, a content type could be text/plain, text/html, or multipart/alternative.
MCF_EMAIL_HEADERS	This field is used to return the email message headers. It contains the raw headers in RFC 822 format. The MCFHeaders class can be used to parse the contents of this field.

The following table describes the fields within the rowset at level one. This rowset contains the parts of the email.

 Copyright © 1988-2007, Oracle. All rights reserved.

Field	Description
MCF_ATT_URL	This is the URL of an associated attachment in the attachment directory. This URL is a relative value when this rowset is first received from the mail server and stored in the database. It only becomes a fully qualified URL once you retrieve the rowset using the MCFMailStore class. Keeping the URL a relative value enables you to make changes within the system without breaking the URL.
MCF_IS_ATT_URL	This is a boolean value indicating whether the email part is located in the MCF_EMAIL_TEXT or on the attachment directory. If this value is 1 (true), the email part can be accessed using the MCF_ATT_URL value. If this value is 0 (false), then the email part is located in MCF_EMAIL_TEXT.
MCF_FILENAME	Represents the file name for an attachment, such as revenue.xls or resume.doc. The file name can be used as a read-only label on the page used by end users to view attachments. For example, the file name may appear next to the link that opens the attachment. **Note.** Not all parts have file names.
MCF_FILE_DATA	This field is not currently used.
MCF_EMAIL_HEADERS	This field is used to return the email part headers. It contains the raw headers in RFC 822 format. The MCFHeaders class can be used to parse the contents of this field.
MCF_EMAIL_TEXT	Contains the text content of the email part.

Note. Although the same rowset structure is used for retrieving email from the mail server and for retrieving email from the PeopleSoft database, not all of the fields apply to each situation. For example, when you first retrieve email from the mail server, the rowset does not contain the fully qualified URL to any attachments.

Note. After an email is saved to the database, always use the methods in the MailStore application package class to access the email. Do not read directly from the mail tables.

Copyright © 1988-2007, Oracle. All rights reserved.

Error Messages Returned by MCFGetMail Class Methods

The GetMail PeopleSoft Integration Broker connector returns message codes denoting status. These return codes apply to all the methods within the MCFGetMail class. The following are returned in MCF_RET_STATUS field, and by the Status property.

0 Success.

1 Connector size overflow.

The size of the email requested exceeds the value in MCF_EmSz_Conn.

The system returns the header information for this email with the return status set to 1.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft MultiChannel Framework*, "Configuring the Email Channel," Configuring GETMAILTARGET Properties.

2 PeopleSoft Integration Broker threshold size overflow.

The size of the email requested exceeds the value in MCF_EmSz_IB.

This happens when the size of the XML message constructed to transfer emails from the mail server to the database exceeds the threshold value.

If there are any more emails to be processed in the current batch, the system does not fetch them from the mail server. For example, suppose your program fetches 20 emails at a time from the mail server, and this error occurred on the tenth email. In this case, the system processes the tenth email, but emails 11–20 are still be on the mail server.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft MultiChannel Framework*, "Configuring the Email Channel," Configuring GETMAILTARGET Properties.

3 Cannot delete all the messages from the mail server.

4 Connecting to the mail server failed.

5 The attachments to be deleted were not found.

6 At least one attachment cannot be deleted.

7 Unsupported encoding.

8 Cannot write to the attachment directory.

9 Messages were downloaded to directory and deleted from mail server.

This status code occurs when an exception occurs in PeopleSoft Integration Broker while storing the emails to the database. The exception might occur due to invalid characters in the email. If the email is left on the mail server, the same exception will occur repeatedly, preventing you from fetching further emails. Because the header might contain these invalid characters also, only MCF_RET_STATUS, MCF_MSG_SIZE, MCF_UID_LIST, date time fields, MCF_IS_ATT_URL, MCF_ATT_URL are set. There is no way to determine which email caused the exception, so the system writes all the emails to the directory. All the emails in the batch are deleted from the mail server.

10 Email content error. One example of this is unsupported encoding.

 Copyright © 1988-2007, Oracle. All rights reserved.

11	Returned if attempting to create a mail server folder while using the POP3 protocol.
12	Folder creation on mail server failed.
13	GetMail connector internal error.

See Chapter 20, "Mail Classes," Status, page 675.

Mail Classes Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," DeQueue

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," EnQueue

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Forward

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," InitChat

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," NotifyQ

Mail Classes Constructors

The following section provides examples of the mail class constructors.

MCFBodyPart

Example

```
Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();
```

MCFGetMail

Example

```
Local MCFGetMail &gm = create MCFGetMail();
```

MCFOutboundEmail

Example

```
Local PT_MCF_MAIL:MCFOutboundEmail &eMail =
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
create PT_MCF_MAIL:MCFOutboundEmail();
 `
```

MCFMailStore

Example

```
Local MCFGetMail &sm = create MCFMailStore();
```

SMTPSession

Example

```
Local PT_MCF_MAIL:SMTPSession &commonSession =
create PT_MCF_MAIL:SMTPSession();
```

MCFBodyPart Class

Use the MCFBodyPart class to send email with attachments, HTML, or other non-standard text information.

MCFBodyPart Class Methods

In the following, we discuss the MCFBodyPart class methods. The methods are described in alphabetical order.

AddHeader

Syntax

AddHeader(*HeaderName*, *HeaderValue*)

Description

Use the AddHeader method to add a header to the body part. This method allows for email server customizations. Some commonly used headers are already exposed as properties, such as ContentType and ContentLanguage. Advanced applications can adapt this technique to meet their own requirements. These headers can be either standard SMTP headers or custom headers starting with "X-".

Parameters

Parameter	Description
HeaderName	Specify the name of the header that you want to add. This parameter takes a string value.
HeaderValue	Specify the actual text of the header, as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

```
  Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail⇒
();
   Local string &TestName = "Custom Header";

   &email.From = &def.From;
   &email.Recipients = &def.Recipients;
   &email.SMTPServer = &def.SMTPServer;
   &email.Subject = &TestName;
   &email.Text = &TestName;

   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP2");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP3");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP4");
   &res = &email.Send();
```

See Also

Chapter 20, "Mail Classes," GetHeader, page 645 , Chapter 20, "Mail Classes," GetHeaderCount, page 646 , Chapter 20, "Mail Classes," GetHeaderName, page 646 , Chapter 20, "Mail Classes," GetHeaderNames, page 647 , Chapter 20, "Mail Classes," GetHeaderValues, page 647 and Chapter 20, "Mail Classes," GetUnparsedHeaders, page 648.

GetHeader

Syntax

GetHeader(*HeaderName*)

Description

Use the GetHeader method to return the value of the specified header.

This method is primarily used with inbound email messages.

Parameters

Parameter	Description
HeaderName	Specify the name of the header that you want to access the values for, as a string. The matching of header names is case insensitive.

Returns

An array of string containing the header values. If the header is not present, returns an array of length zero.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 8, “Array Class,” page 215
Chapter 20, “Mail Classes,” AddHeader, page 644 , Chapter 20, “Mail Classes,” GetHeaderCount, page 646 , Chapter 20, “Mail Classes,” GetHeaderName, page 646 , Chapter 20, “Mail Classes,” GetHeaderNames, page 647 , Chapter 20, “Mail Classes,” GetHeaderValues, page 647 and Chapter 20, “Mail Classes,” GetUnparsedHeaders, page 648.

GetHeaderCount

Syntax

```
GetHeaderCount()
```

Description

Use the GetHeaderCount method to return the number of headers.

Parameters

None.

Returns

An integer, representing the number of headers present.

See Also

Chapter 20, “Mail Classes,” GetHeader, page 645 , Chapter 20, “Mail Classes,” AddHeader, page 644 , Chapter 20, “Mail Classes,” GetHeaderName, page 646 , Chapter 20, “Mail Classes,” GetHeaderNames, page 647 , Chapter 20, “Mail Classes,” GetHeaderValues, page 647 and Chapter 20, “Mail Classes,” GetUnparsedHeaders, page 648.

GetHeaderName

Syntax

```
GetHeaderName(Index)
```

Description

Use the GetHeaderName to return the name of the header that is located at *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the header you want to access.

Returns

A string.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 20, “Mail Classes,” GetHeader, page 645 , Chapter 20, “Mail Classes,” GetHeaderCount, page 646 , Chapter 20, “Mail Classes,” AddHeader, page 644 , Chapter 20, “Mail Classes,” GetHeaderNames, page 647 , Chapter 20, “Mail Classes,” GetHeaderValues, page 647 and Chapter 20, “Mail Classes,” GetUnparsedHeaders, page 648.

GetHeaderNames

Syntax
```
GetHeaderNames()
```

Description

Use the GetHeaderNames method to return an array containing the names of all the headers.

This method is primarily used with inbound email messages.

Parameters

None.

Returns

An array of string.

See Also
Chapter 20, “Mail Classes,” AddHeader, page 644 , Chapter 20, “Mail Classes,” GetHeader, page 645 , Chapter 20, “Mail Classes,” GetHeaderName, page 646 , Chapter 20, “Mail Classes,” GetHeaderCount, page 646 and Chapter 20, “Mail Classes,” GetHeaderValues, page 647.

GetHeaderValues

Syntax
```
GetHeaderValues(Index)
```

Description

Use the GetHeaderValues method to return the value for the header located at the position specified by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the header whose value you want to access.

Returns

An array of string.

See Also
Chapter 20, “Mail Classes,” AddHeader, page 644 , Chapter 20, “Mail Classes,” GetHeader, page 645 , Chapter 20, “Mail Classes,” GetHeaderCount, page 646 , Chapter 20, “Mail Classes,” GetHeaderName, page 646 , Chapter 20, “Mail Classes,” GetHeaderNames, page 647 and Chapter 20, “Mail Classes,” GetUnparsedHeaders, page 648.

Copyright © 1988-2007, Oracle. All rights reserved.

GetUnparsedHeaders

Syntax

```
GetUnparsedHeaders()
```

Description

Use the GetUnparsedHeaders method to return all of the headers in the body part without any formatting occurring.

Parameters

None.

Returns

A string containing the headers.

See Also

Chapter 20, “Mail Classes,” AddHeader, page 644 , Chapter 20, “Mail Classes,” GetHeader, page 645 , Chapter 20, “Mail Classes,” GetHeaderCount, page 646 , Chapter 20, “Mail Classes,” GetHeaderName, page 646 , Chapter 20, “Mail Classes,” GetHeaderNames, page 647 and Chapter 20, “Mail Classes,” GetHeaderValues, page 647.

SetAttachmentContent

Syntax

```
SetAttachmentContent({FilePath | FileURL}, FilePathType, FileName, FileDescr,
OverrideContentType, OverrideCharset)
```

Description

Use the SetAttachmentContent method to specify the body (content) of this body part from a file.

Use the *FilePathType* parameter to specify if the file path is relative or absolute. If it’s an absolute path, the file path could be a file on the local machine, a network folder, or a fully-qualified URL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
FilePath | *FileURL*	You can either specify the file path to the file, or a URL to the file. Depending on the *FilePathType* parameter, you may specify either a relative or absolute file path to the file. This parameter should include the file name and extension of the file. If you specify a URL address where the file is located, it must be an absolute URL. This URL should include the actual name of the file. If you specify a URL address, you must specify the *FilePathType* parameter as %FilePath_Absolute.
FilePathType	Specify the path type for the file, whether it is an absolute or relative path. The values are: • %FilePath_Absolute — the file path is an absolute path. • %FilePath_Relative — the file path is a relative path.
FileName	Specify the name of the file that you want to attach, as a string. You must include the file extension as well.
Description	Specify a description of the file.
OverrideContentType	The system detects the content type of the attachment using the file location and name. Specifying *OverrideContentType* overrides the system detected content type. The character set can also be included in the ContentType. For example, “text/plain; charset=US-ASCII”
OverrideCharset	Specify a character set to be used to override the existing character set.

Returns

None.

Example

The following uses an absolute path to set the content:

```
&image.SetAttachmentContent("///file:C:/User/Documentum/XML%20Applications/proddoc⇒
/peoplebook_upc/peoplebook_upc.dtd",
 %FilePath_Absolute, "sample.jpg", "This is a sample image!", "", "");
```

The following uses a relative path to set the content:

```
&attach1.SetAttachmentContent("Ocean Wave.jpg", %FilePath_Relative,
"Ocean Wave.jpg", "Ocean Wave", "", "");
```

The following uses a URL to set the content:

```
&attach1.SetAttachmentContent("http://www.yahoo.com/members_agreement",
 %FilePath_Absolute, "hotmail.htm", "Hotmail Home page", "", "");
```

See Also

Chapter 20, “Mail Classes,” AddAttachment, page 688

Copyright © 1988-2007, Oracle. All rights reserved.

MCFBodyPart Class Properties

In this section we discuss the MCFBodyPart class properties. The properties are described in alphabetical order.

AttachmentURL

Description

Use this property to specify the URL for the attachment for this body part, as a string.

The URL must be an absolute URL.

This property is read-write.

Charset

Description

Use this property to specify the character set for the text or the attachment.

You can also specify this property using the SetAttachmentContent method, or the ContentType property.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” ContentType, page 650

Chapter 20, “Mail Classes,” SetAttachmentContent, page 648

ContentType

Description

Use this property to specify the content type of this body part.

You can specify the content type with the SetAttachmentContent method.

You can also use this property to specify the character set. For example, “text/plain; charset=US-ASCII”.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” SetAttachmentContent, page 648

Chapter 20, “Mail Classes,” Charset, page 650

Description

Description

Use this property to specify a description of the attachment. You should only use this property if the content is an attachment.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Disposition

Description

Use this property to specify how the body part is presented in the received mail.

Some email clients ignore the setting of this property and present body parts either inline or as file attachments.

Valid values for this property are:

Value	Description
Attachment	The body part is shown as a file attachment.
Inline	The body part is shown in the body itself.

This property is read-write.

Filename

Description

If you are adding an attachment to the email, use this property to specify the name of the file.

PeopleSoft recommends that you keep the file extension specified with this property the same as the original extension found in the file path, otherwise client applications may not be able to display it properly.

This property is read-write.

FilePath

Description

Specify the path for the file that contains the contents of this email.

Whether this is a relative or absolute path depends on the FilePathType property.

You can also specify a URL to the file using this property, if the FilePathType property is specified as %FilePath_Absolute.

If you specify a value for this property, the ‘Text’ content is ignored.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” FilePathType, page 652

Copyright © 1988-2007, Oracle. All rights reserved.

FilePathType

Description

Use this property to specify whether the path specified with the FilePath property is a relative or absolute path. The values for this property are:

Value	Description
%FilePath_Relative	The file path specified with the FilePath property is a relative path.
%FilePath_Absolute	The file path specified with the FilePath property is either an absolute path to a file, or a URL to a file.

If you specify a relative path, the file must be available in the FILES folder of application server folder.

If you specify an absolute path, the file could be on the local machine, in any network folder, or a URL.

This property is read-write.

IsAttachment

Description

This property indicates if the body part is an attachment or not. This property returns a boolean value: true if it is an attachment, false otherwise.

This property is read-only.

MultiPart

Description

A bodypart can contain simple text, one attachment, or a MultiPart object.

If you have assigned a MultiPart object using this property, the text and attachment related properties are ignored.

This property is read-write.

Example

```
Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.SubType = "alternative; differences=Content-type";

&mp.AddBodyPart(&text);
&mp.AddBodyPart(&html);

&email.MultiPart = &mp;
```


Copyright © 1988-2007, Oracle. All rights reserved.

Text

Description

Use this property to specify the text for the email.

Use the SetAttachmentContent method to specify image or other types of attachments. For a more complex bodypart, build a multipart and set the Multipart property of MCFBodyPart.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” SetAttachmentContent, page 648

MCFEmail Class

This class inherits many properties from the MCFPart class, and is the base class for both the MCFOutboundEmail and MCFInboundEmail classes. Generally, you only use the MCFOutboundEmail and MCFInboundEmail classes. You should only use the MCFEmail class if you are extending the PT_MCF_MAIL application package.

MCFEmail Class Properties

In this section, we discuss the MCFEmail class properties. The properties are discussed in alphabetical order.

BCC

Description

Use this property to specify comma-separated list of addresses of all the blind carbon copy recipients of this email. (A blind carbon copy is a copy of the email that contains all of the text of the email, but does not show any of the other email addresses to which this email was sent.)

This property is read-write.

See Also

Chapter 20, “Mail Classes,” CC, page 654

BounceTo

Description

Use this property to specify the email address the system should direct all bounced mail to.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

CC

Description

Use this property to specify a comma-separated list of addresses that copies of this email are to be sent to (carbon copy.)

This property is read-write.

See Also

Chapter 20, “Mail Classes,” BCC, page 653

From

Description

Use this property to specify the email address of the person sending the email.

You can specify more than one address as from in a comma-separated list.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” Recipients, page 655 , Chapter 20, “Mail Classes,” Sender, page 656 and Chapter 20, “Mail Classes,” ReplyTo, page 655.

Importance

Description

Use this property to specify the value of the importance header field.

The importance can be set to the following:

- low
- normal
- high

If the Priority property is not set, the system automatically sets it to the corresponding values like 5, 3 and 1.

This property is read-write.

Priority

Description

Use this property to specify the priority of this email.

The values are:

1. Highest Priority
2. High Priority
3. Normal

 Copyright © 1988-2007, Oracle. All rights reserved.

4. Low Priority
5. Lowest Priority

The default value is 3, that is, normal priority.

This value set with this property is used to set to a header X-Priority field, as this is the most common but non-standard field to show priority. In addition, the corresponding values are set in two header fields: X-MSMail-Priority and Priority. The headers X-Priority or X-MSMail-Priority are set to the corresponding value only if the user does not specify a value.

This property is read-write.

Recipients

Description

Use this property to specify the email addresses of all the main recipients to whom the email is being sent. All the addresses are specified in a comma-separated string.

This property is read-write.

RefIDs

Description

Use this property to specify a value for the REFERENCES header field of the message.

This property is read-write.

ReplyIDs

Description

Use this property to specify a value for the IN-REPLY-TO header field of the message.

This property is read-write.

ReplyTo

Description

Use this property to specify the email address where the reply should be sent. You do not need to specify a value for this property if the value is the same as the From property.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” From, page 654

Copyright © 1988-2007, Oracle. All rights reserved.

Sender

Description

Use this property to specify the address of the author of the message. You do not need to specify a value for this property if the value for the From property is the same.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” From, page 654

Sensitivity

Description

Use this property to specify the value of the sensitivity header field.

The values are:

- personal
- private
- company-confidential

This property is read-write.

Subject

Description

Use this property to specify the subject of the email. A subject can only contain 254 characters.

This property is read-write.

Text

Description

Use this property to specify the text of the email.

This property is read-write.

MCFGetMail Class Methods

This section discusses MCFGetMail class methods.

CreateQuarantineFolder

Syntax

```
CreateQuarantineFolder()
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the CreateQuarantineFolder method to create a quarantine folder on the server. Specify the name of the folder using the QuarantineFolder property

Parameters

None.

Returns

A boolean value; true if the folder is created, or if it already exists on the server, false otherwise.

See Also

Chapter 20, "Mail Classes," QuarantineFolder, page 667 and Chapter 20, "Mail Classes," QuarantineCount, page 667.

GetCount

Syntax

```
GetCount()
```

Description

Use the GetCount method to determine the total number of emails currently stored in a particular email account.

This method does not distinguish between read and unread emails.

This method uses the connection properties set by SetMCFEmail.

If you want to determine the number of emails for an account, and to set your own connection properties, use the GetEmailCount method instead.

Parameters

None.

Returns

Returns the total number of emails stored on the mail server for a particular email account. If the method returns –4, the system could not connect to the mail server using the user name and password passed in the parameter list.

See Also

Chapter 20, "Mail Classes," GetEmailCount, page 657 and Chapter 20, "Mail Classes," SetMCFEmail, page 664.

GetEmailCount

Syntax

```
GetEmailCount(user, password, server, node)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use GetEmailCount to determine the number of emails currently stored in a particular email account. GetEmailCount does not distinguish between read and unread email. You can use this value to set limits on loops in your program.

If you want to determine the number of emails for an account using the connection properties set with SetMCFEmail, use the GetCount method instead.

Parameters

Parameter	Description
user	The user name on the mail server, such as “support” or “john_doe”.
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The PeopleSoft Integration Broker node on which the request runs.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, PeopleSoft Integration Broker returns an error.

Returns

Returns the total number of emails stored on the mail server for a particular email account. If the method returns –4, the system could not connect to the mail server using the user name and password passed in the parameter list.

Example

```
Local number &email_count;

&email_count = &gm.GetEmailCount(&username, &passwd, &mailserver,
&node);
```

See Also

Chapter 20, “Mail Classes,” GetCount, page 657 and Chapter 20, “Mail Classes,” SetMCFEmail, page 664.

ReadAllEmailHeadersWithAttach

Syntax

ReadAllEmailHeadersWithAttach(*user*, *password*, *server*, *node*)

Description

Use ReadAllEmailHeadersWithAttach to read the headers of all of the emails and a list of associated attachments that are currently stored on the mail server for a particular email account. The details about the emails are returned in the level zero rowset. The actual emails and any associated attachments *are not* returned.

Use this method as a screening method to determine whether to store an email in your PeopleSoft system. For example, if you determine that an email is junk mail, you could delete it from your mail server. After you determine the allowable email through header information, you can retrieve the physical email and attachments of the allowable emails.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
user	The user name on the mail server, such as “support” or “john_doe”.
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The PeopleSoft Integration Broker node on which the request runs.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, PeopleSoft Integration Broker returns an error.

Returns

This method returns rowsets containing email headers.

Example

```
import PT_MCF_MAIL:*;

Local MCFGetMail &gm;

Local Rowset &emails_rs;

&emails_rs = &gm.ReadAllEmailHeadersWithAttach(&username, &passwd, &mailserver
&node);
```

ReadEmails

Syntax

ReadEmails(*Count*)

Description

Use the ReadEmails method to read the specified number of emails for this mailbox.

This method uses the connection properties set by SetMCFEmail.

The overall status of the success of this method is available in the Status property.

Parameters

Parameter	Description
Count	Specify the number of emails you want to read.

Returns

An array of MCFInboundEmail.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, "Mail Classes," MCFInboundEmail Class Methods, page 668

Chapter 20, "Mail Classes," Status, page 675

ReadEmailsWithAttach

Syntax

ReadEmailsWithAttach(*User*, *password*, *server*, *node*, *count*)

Description

The ReadEmailsWithAttach method reads the number of emails specified by *count* for a particular user account. If attachments are associated with an email, this method reads the attachments as well.

Parameters

Parameter	Description
User	Specify the user account that contains the emails you want to access, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The PeopleSoft Integration Broker node on which the request runs.
count	Specifies the number of emails to read. If count = 0, the method reads *all* the emails of the user account. If count > 0, the method reads the number specified. For example, if count = 10, the method reads 10 emails.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, PeopleSoft Integration Broker returns an error.

Returns

This method returns an email within a rowset. level one of the rowset contains the email parts. level zero row 1 only contains values for MCF_NUMROWS and MCF_RET_STATUS.

Example

```
import PT_MCF_MAIL:*;

Local MCFGetMail &gm;

Local Rowset &emails_rs;

&emails_rs = &gm.ReadEmailsWithAttach(&username, &passwd, &mailserver
&node, &email_count);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, "Mail Classes," ReadEmailWithAttach, page 661

ReadEmailsWithUID

Syntax

ReadEmailsWithUID(*UID*)

Description

Use the ReadEmailsWithUID method to read and return an email based on a unique identifier *UID*. This method uses the connection properties set by SetMCFEmail. Emails that have an attachment are also read.

Parameters

Parameter	Description
UID_List	The UID is the unique identifier of the email. This parameter is only unique within a particular user account. POP3 and IMAP 4 mail servers automatically create a unique identifier for an email when the server receives the email. The list can contain multiple values, each separated by a space.

Returns

A reference to an array of MCFInboundEmail.

See Also

Chapter 20, "Mail Classes," MCFInboundEmail Class Methods, page 668

ReadEmailWithAttach

Syntax

ReadEmailWithAttach(*User*, *password*, *server*, *node*, *UID*)

Description

Use the ReadEmailWithAttach method to read and return an email based on a unique identifier (*UID*) for the specific email to be read. If attachments are associated with this email, ReadEmailWithAttach fetches those as well.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
user	The user name on the mail server, such as “support” or “john_doe”.
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The PeopleSoft Integration Broker node on which the request runs.
UID	The unique identifier of the email. This parameter is only unique within a particular user account. POP3 and IMAP 4 mail servers automatically create a unique identifier for an email when the server receives the email. You may specify more than one UID, each separated by a space.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, PeopleSoft Integration Broker returns an error.

Returns

This method returns an email within a rowset. level one of the rowset contains the email parts. level zero row 1 only contains values for MCF_NUMROWS and MCF_RET_STATUS.

Example

```
import PT_MCF_MAIL:*;

Local MCFGetMail &gm;

Local Rowset &emails_rs;

&emails_rs = &gm.ReadEmailWithAttach(&username, &passwd, &mailserver, &node, &UID);
```

See Also

Chapter 20, “Mail Classes,” ReadEmailsWithAttach, page 660

ReadHeaders

Syntax

ReadHeaders()

Description

Use the ReadHeaders method to read the headers of all emails currently on the server. This method also returns the attachment information for all emails (names and sizes.)

This method uses the connection properties set by SetMCFEmail.

The overall success of the method is available using the Status property.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

An array of MCFInboundEmail objects.

See Also

Chapter 20, "Mail Classes," MCFInboundEmail Class Methods, page 668

Chapter 20, "Mail Classes," SetMCFEmail, page 664

RemoveEmail

Syntax

RemoveEmail(*user*, *password*, *server*, *node*, *UID list*)

Description

Based on values in the unique identifier list, RemoveEmail deletes emails from the user account on the mail server. It is important to delete emails from the mail server once they are read into PeopleSoft. If you do not delete emails from the mail server after they are read into PeopleSoft, the rowset contains the same collection of email the next time you retrieve email from the mail server.

Parameters

Parameter	Description
user	The user name on the mail server, such as "support" or "john_doe".
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The PeopleSoft Integration Broker node on which the request runs.
UID list	The UID list contains the unique identifier values for the emails that need to be deleted from the mail server. The list can contain multiple values separated by a space.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, the Integration Broker returns an error.

Returns

A boolean value, indicating the status of the deletion process. If the email has been successfully, this method returns True. Otherwise the method returns False.

Example

```
Local MCFGetMail &GM;
Local boolean &del_status;
Local string &uid_list = "";
Local string &msg_str;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
If (&uid_list <> "") Then
   &del_status = &GM.RemoveEmail(&user, &password, &server, &ibnode, &uid_list);
   &msg_str = "Email(s) deleted from the Mail Server with UID = " | &uid_list;
   MessageBox(0, "", 0, 0, &msg_str);
End-If;
```

See Also

Chapter 20, "Mail Classes," RemoveEmails, page 664

RemoveEmails

Syntax

RemoveEmails(*UID_List*)

Description

Use the RemoveEmails method to delete emails from an account on the mail server, based on the *UID_list*. It is important to delete emails from the mail server once they are read into PeopleSoft. If you do not delete emails from the mail server after they are read into PeopleSoft, the rowset contains the same collection of email the next time you retrieve email from the mail server.

Parameters

Parameter	Description
UID_List	The UID list contains the unique identifier values for the emails that need to be deleted from the mail server. The list can contain multiple values separated by a space.

Returns

An array of string containing UIDs of the messages that were successfully removed, separated by semicolons.

See Also

Chapter 20, "Mail Classes," RemoveEmail, page 663

SetMCFEmail

Syntax

SetMCFEmail(*user*, *password*, *server*, *node*)

Description

Use the SetMCFEmail method to set the connection properties for subsequent method calls.

This method is a shortcut for setting the UserId, Password, MailServer, and IBNode properties.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
user	The user name on the mail server, such as “support” or “john_doe”.
password	The password associated with the specified user name.
server	The name of the mail server handling the specified user account.
node	The PeopleSoft Integration Broker node on which the request runs.

Note. If the value of any of these parameters is null, the method uses default values stored on the specified node. If the node is not specified, the integration broker returns an error.

Returns

None.

See Also

Chapter 20, “Mail Classes,” GetCount, page 657 , Chapter 20, “Mail Classes,” ReadEmails, page 659 , Chapter 20, “Mail Classes,” ReadEmailsWithUID, page 661 and Chapter 20, “Mail Classes,” ReadHeaders, page 662.

MCFGetMail Class Properties

In this section we discuss the MCFGetMail class properties. The properties are described in alphabetical order.

AttachmentRoot

Description

Use this property to specify the root directory where the attachments are stored. If not specified, the value from the node is used.

Valid metastrings that can be included in the value for this property are:

%CURRDATE%	Current date in YYYYMMDD format
%CURRHOUR%	Current hour in 24–hour format (00-23)
%DBNAME%	Current value of %DbName system variable.
%OPRID%	Current value of %UserId system variable.

This property is read-write.

Example

To have the attachment root directory change hourly, you could use the following code example:

```
&MyEmail.AttachmentRoot = "c:\temp\%CURRDATE%\%CURRHOUR%\";
```

Copyright © 1988-2007, Oracle. All rights reserved.

ContentTypes

Description

Use this property to specify content types that should be returned in the message or part text rather than stored in the directory that contains the attachments. Specify content types in a comma separated list. For example: `"text/html,text/xml"`.

Note. `"Text/plain"` is always returned in the text field and does not need to be included.

If not specified, the value from the node is used.

This property is read-write.

ErrorCount

Description

This property returns the number of messages that caused errors during processing on the target connector.

You should use this property after you use the ReadHeaders, ReadEmails, ReadEmailsWithUID, or RemoveEmails methods.

This property is read-only.

See Also

Chapter 20, “Mail Classes,” ReadHeaders, page 662 , Chapter 20, “Mail Classes,” ReadEmails, page 659 , Chapter 20, “Mail Classes,” ReadEmailsWithUID, page 661 and Chapter 20, “Mail Classes,” RemoveEmails, page 664.

IBNode

Description

This property returns the name of the PeopleSoft Integration Broker node on which a request ran, such as MCF_GetMail.

This property is set by the SetMCFEmail method. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID, RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” ReadHeaders, page 662 , Chapter 20, “Mail Classes,” ReadEmails, page 659 , Chapter 20, “Mail Classes,” ReadEmailsWithUID, page 661 , Chapter 20, “Mail Classes,” RemoveEmails, page 664 and Chapter 20, “Mail Classes,” CreateQuarantineFolder, page 656.

MailServer

Description

This property returns the name of the mail server that received the email.

This property is set by the SetMCFEmail method. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID, RemoveEmails, and CreateQuarantineFolder methods.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” ReadHeaders, page 662 , Chapter 20, “Mail Classes,” ReadEmails, page 659 , Chapter 20, “Mail Classes,” ReadEmailsWithUID, page 661 , Chapter 20, “Mail Classes,” RemoveEmails, page 664 and Chapter 20, “Mail Classes,” CreateQuarantineFolder, page 656.

Password

Description

This property returns the password associated with the userID that was used to get the email.

This property is set by the SetMCFEmail method. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID, RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” ReadHeaders, page 662 , Chapter 20, “Mail Classes,” ReadEmails, page 659 , Chapter 20, “Mail Classes,” ReadEmailsWithUID, page 661 , Chapter 20, “Mail Classes,” RemoveEmails, page 664 and Chapter 20, “Mail Classes,” CreateQuarantineFolder, page 656.

QuarantineCount

Description

This property returns the number of email messages that caused errors during processing on the target connector and were successfully moved to the quarantine folder.

This property is valid after calls to ReadHeaders, ReadEmails, ReadEmailsWithUID, and RemoveEmails.

This property is read-only.

See Also

Chapter 20, “Mail Classes,” QuarantineFolder, page 667
Chapter 20, “Mail Classes,” ReadHeaders, page 662 , Chapter 20, “Mail Classes,” ReadEmails, page 659 , Chapter 20, “Mail Classes,” ReadEmailsWithUID, page 661 and Chapter 20, “Mail Classes,” RemoveEmails, page 664.

QuarantineFolder

Description

Use this property to specify the name of the quarantine folder. If not specified, the value from the node will be used.

The name must adhere to whatever naming restrictions are imposed by the mail server in use. There is no built-in validation of the folder name.

The following types of errors can cause an email to be moved to the quarantine folder:

- Connector size overflow.
- Unsupported encoding.

Copyright © 1988-2007, Oracle. All rights reserved.

- Unknown Java exception.
- Javamail content error.
- No attachment directory.
- Mail parse exception.

This property is read-write.

See Also

Chapter 20, "Mail Classes," QuarantineCount, page 667

Status

Description

This property returns the status of the last mail server operation.

See Chapter 20, "Mail Classes," Error Messages Returned by MCFGetMail Class Methods, page 642.

This property is valid after calls to ReadHeaders, ReadEmails, ReadEmailsWithUID, and RemoveEmails.

This property is read-only.

UserID

Description

This property returns the user ID (mail box) on the mail server that received the email.

This property is set by SetMCFEmail. It is used by ReadHeaders, ReadEmails, ReadEmailsWithUID,RemoveEmails, and CreateQuarantineFolder methods.

This property is read-write.

MCFInboundEmail Class Methods

In this section, we discuss the MCFInboundEmail class methods. The methods are described in alphabetical order.

DumpToFile

Syntax

```
DumpToFile(&File)
```

Description

Use the DumpToFile method to write information from the email to an already instantiated file.

This method copies the information from the following email properties, placing each on a separate line:

- ContentType

 Copyright © 1988-2007, Oracle. All rights reserved.

- IsAttachment
- AttachmentURL
- Text

Whether the information is appended to the bottom of the file or if it overwrites the file depends on how you instantiated the file object.

Parameters

Parameter	Description
&File	Specify an already instantiated file object that you want to write the email data to.

Returns

None.

Example

```
Local File &BI_FILE;
&BI_FILE = GetFile("mcfdata.out", "w", "a");

Local PT_MCF_MAIL:MCFGetMail &gm;
Local number &ret_status, &nemails, &i;
&gm = create PT_MCF_MAIL:MCFGetMail();

Local array of PT_MCF_MAIL:MCFInboundEmail &emails;
&gm.SetMCFEmail(&user, &password, &server, &ibnode);

&emails = &gm.ReadEmails(0);
If (&gm.StatusCheck(&BI_FILE) = False) Then
   Return;
End-If;

&nemails = &emails.Len;
&ret_status = &gm.Status;
If &nemails > 0 Then
   For &i = 1 To &nemails
      &BI_FILE.WriteLine(MsgGetText(162, 1615, "Message Not Found", &i));
      &emails [&i].DumpToFile(&BI_FILE);
   End-For;
End-If;
```

See Also

Chapter 16, “File Class,” page 523

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetFile

Copyright © 1988-2007, Oracle. All rights reserved.

GetAttachments

Syntax

```
GetAttachments()
```

Description

Use the GetAttachments method to return the attachments with this email.

Parameters

None.

Returns

An array of MCFBodyPart objects. The methods returns an array of length 0 if there are no attachments

See Also

Chapter 8, “Array Class,” page 215

Chapter 20, “Mail Classes,” MCFBodyPart Class Methods, page 644

GetFrom

Syntax

```
GetFrom()
```

Description

Use the GetFrom method to split the semicolon delimited list of addresses in the From property into an array of addresses.

Parameters

None.

Returns

An array of string. Each item in the array contains an email address.

See Also

Chapter 20, “Mail Classes,” From, page 654

GetParts

Syntax

```
GetParts()
```

Description

Use the GetParts method to return all of this email’s parts.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

An array of MCFBodyPart objects.

See Also

Chapter 20, "Mail Classes," MCFBodyPart Class Methods, page 644

GetSender

Syntax

```
GetSender()
```

Description

Use the GetSender method to split the semicolon delimited list of addresses in the Sender property into an array of addresses.

Parameters

None.

Returns

An array of string.

See Also

Chapter 20, "Mail Classes," Sender, page 656

Chapter 20, "Mail Classes," GetFrom, page 670

ReadFromDatabase

Syntax

```
ReadFromDatabase(Email_ID)
```

Description

Use the ReadFromDatabase method to load a saved email from the database into this instance of email.

Parameters

Parameter	Description
Email_ID	Specify the ID of the email, as a number.

Returns

A boolean: true if the email is restored successfully, false otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, “Mail Classes,” SaveToDatabase, page 672

SaveToDatabase

Syntax

```
SaveToDatabase()
```

Description

Use the SaveToDatabase method to save this instance of email to the database.

Parameters

None.

Returns

A number, representing the email_ID of the saved email.

See Also

Chapter 20, “Mail Classes,” ReadFromDatabase, page 671

MCFInboundEmail Class Properties

In this section, we discuss the MCFInboundEmail class properties. The properties are described in alphabetical order.

AttachList

Description

This property returns the list of all the file names that are attachments for a particular email.

Multiple items are separated by a semicolon (;).

This property is read-write.

See Also

Chapter 20, “Mail Classes,” AttachSizes, page 672

AttachSizes

Description

This property returns the sizes of the attachments associated with an email.

This property takes a string value, *not* a numeric value. Multiple values are separated by a semicolon (;).

 Copyright © 1988-2007, Oracle. All rights reserved.

The attachment sizes appear in the same order as the file names in AttachList.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” AttachList, page 672

DttmReceived

Description

Use this property to return the date and time an email was received.

Warning! If you are using a Post Office Protocol 3 (POP3) mail server, this property may not be populated.

This property takes a datetime value.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” DttmSaved, page 673 and Chapter 20, “Mail Classes,” DttmSent, page 673.

DttmSaved

Description

This property returns the date and time that the PeopleSoft system saved an email to the database.

If you use a POP3 mail server, PeopleSoft recommends using this value as the base, or starting point, for any time-sensitive transactions.

This property takes a datetime value.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” DttmReceived, page 673 and Chapter 20, “Mail Classes,” DttmSent, page 673.

DttmSent

Description

This property returns the date time an email was sent.

This property takes a datetime value.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” DttmReceived, page 673 and Chapter 20, “Mail Classes,” DttmSent, page 673.

Copyright © 1988-2007, Oracle. All rights reserved.

IBNode

Description

This property returns the name of the PeopleSoft Integration Broker node that received the email, such as MCF_GetMail.

The system uses this value to construct the fully qualified URL required for viewing attachments.

This property is read-write.

Language

Description

This value reflects the language code of the PeopleSoft Integration Broker node that received this email.

Note. PeopleSoft recommends setting up email accounts that are dedicated to one language such as, support_french@company.com and support_english@company.com. This enables you to anticipate the language of the email you read into your system. In turn, configure separate PeopleSoft Integration Broker nodes to handle each of the languages you support. There are no language recognition procedures within the PeopleSoft system.

This property is read-write.

MessageID

Description

Use this property to return the globally unique identifier for this email.

This ID is generated by the mail server that sent this email.

NotifyCC

Description

This property returns a list of the parties who were copied (carbon copied) on the email. Multiple addresses are separated by a semicolon (;).

This property is read-write.

NotifyTo

Description

This property returns a list of the parties who received the email. Multiple addresses are separated by a semicolon (;).

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

OffsetReceived

Description

This property returns time zone of the mail server that received this email, as a number.

Note. This property contains the time zone of the integration gateway used to fetch this email.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” OffsetSent, page 675

OffsetSent

Description

This property returns the time zone of the mail server that sent this email, as a string.

Note. Currently, this property contains the time zone of the integration gateway used to fetch this email.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” OffsetReceived, page 675

Server

Description

This property specifies the name of the mail server that received the email.

This property is read-write.

Size

Description

This property returns the total size of the entire email. This value includes the sizes of attachments.

This property is read-write.

Status

Description

This property returns the status of an individual email.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

UID

Description

This property returns the unique identifier of the email.

This property is only unique within a particular user account. POP3 and IMAP 4 mail servers automatically create a unique identifier for an email when the server receives the email.

This property is read-write.

User

Description

This property returns the user account (mail box) on the mail server that received the email.

This property is read-write.

MCFMailStore Class

Use the MCFMailStore class to write emails to the PeopleSoft database and to retrieve emails from the PeopleSoft database.

Note. When writing emails to the PeopleSoft database, it is assumed that you have previously used the MCFGetMail class to retrieve email into memory. You use the MCFMailStore class immediately after to save the rowsets to the database.

MCFMailStore Import Statements

Here is the import statement:

```
import PT_MCF_MAIL:MCFMailStore;
```

MCFMailStore Methods

This section discusses MCFMailStore class methods.

AuthorizeEmailAttach

Syntax

AuthorizeEmailAttach(*email ID, authentication name, authentication type, operation*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use AuthorizeEmailAttach to authorize a user or a PeopleSoft role to view attachments associated with an email. By default, no user or role is authorized to view an email; you must explicitly grant authorization with this method. This method authorizes email and the associated attachments one at a time.

Note. If you specify ALL as the user, *any* user can view the attachments for an email.

Parameters

Parameter	Description
email ID	The unique email ID for the email. This is the PeopleSoft email ID, not the ID issued by the mail server.
authentication name	The name of the PeopleSoft user profile or role that needs to be authorized to view the attachments.
authentication type	Specify the type security definition entered as the authentication name. • Specify *2* to indicate a user profile. • Specify*3* to indicate a role.
operation	Specify the type of authorization operation. The following list contains the supported operations. You either add or delete user or role authorization. • 1 = ADD • 2 = DELETE

Returns

Returns a boolean value: True for success, False otherwise.

Example

```
&ms = create MCFMailStore();
&status = &ms.AuthorizeEmailAttach(&emailid, "MCFUser", 2, 1);
```

DeleteEmail

Syntax

DeleteEmail(*email ID*, *forced delete*)

Description

Deletes the specified email from the PeopleSoft database and corresponding attachments from the directory where they are stored.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
email ID	The ID used to uniquely identify each email within the PeopleSoft database.
forced delete	This parameter is used to set the forced delete flag. It is a boolean value. A forced delete ensures that the deletion process continues even in cases where the process encounters issues or errors related to an email. For example, with forced delete enabled, the deletion process continues even though the attachments of the email cannot be found or do not exist. True means "force delete" and False means "do not force delete if there is an error in deleting this email."

Returns

Returns a boolean value. True for success, False otherwise.

Example

```
Local MCFMailStore &ms;

&ms = create MCFMailStore();
&forcedel = True;
&status = &ms.DeleteEmail(&emailid, &forcedel);
```

RetrieveEmail

Syntax

RetrieveEmail(*email ID*)

Description

Use RetrieveEmail to read email from the PeopleSoft database.

Parameters

Parameter	Description
email ID	The ID that uniquely identifies an email within the PeopleSoft database.

Returns

Returns a rowset.

Example

```
Local MCFMailStore &ms;
Local Rowset &EMAIL_ROWSET;

&ms = create MCFMailStore();
&EMAIL_ROWSET = &ms.RetrieveEmail(&emailid);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

StoreEmail

Syntax

StoreEmail(*&rowset, row*)

Description

The StoreEmail method inserts an email retrieved from the mail server into the PeopleSoft database. PeopleSoft stores the contents or pieces of the email in rowsets and child rowsets.

Parameters

Parameter	Description
&rowset	The rowset containing the contents of an email. The rowset should be the rowset used by the MCFGetMail class to retrieve emails from the mail server. The rowset should be an already instantiated rowset. **Note.** The first row of the rowset is empty. The first email in the rowset has a row value of 2.
row	The row number in which the email data is stored. **Note.** The first row is always empty. Row 2 always contains the first email of the rowset.

Returns

Returns a PeopleSoft email ID to act as the unique key for a particular email within the PeopleSoft database.

Returns 0 if an error occurred.

Example

```
Local MCFGetMail &GM;
Local Rowset &rs;
Local number &nemails;
Local string &email_id;

&GM = create MCFGetMail();
&rs = &GM.ReadEmailsWithAttach(&user, &password, &server, &ibnode,
&count);

/* Get the number of emails in the recordset from MCF_NUMROWS field */

&nemails = &rs.GetRow(1).GetRecord(Record.MCFEM_RES_MAIN).GetField
(Field.MCF_NUMROWS).Value;

   If (&nemails > 0) Then

/* First row in the rowset is for header info only. Start writing from
 second row */

      For &i = 1 To &nemails
         &email_id = &ms.StoreEmail(&rs, &i + 1);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      If (&email_id <> 0) Then
         /* Store was successful - email ID returned = &email_id */
      Else
         /* Error in storing email */
      End-If;
   End-For;
End-If;
```

MCFMailUtil Class

Use the MCFMailUtil class validate email addresses, as well as determining if the server is available.

MCFMailUtil Class Methods

In this section we discuss the MCFMailUtil class methods. The methods are described in alphabetical order.

DecodeText

Syntax

DecodeText(*TextToDecode, &DecodedText*)

Description

Use the DecodeText method to decode text. If you only want to decode a word, use the DecodeWord property.

The text is decoded based on the values used with the EncodeText method.

The decoded text is placed in the *&DecodedText* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return all the substitution strings used to format the error message.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TextToDecode	Specify the text that is to be decoded, as a string.
&DecodedText	Specify a string variable, used to hold the decoded text.

Returns

A boolean value: true if successful, false otherwise.

See Also

Chapter 20, "Mail Classes," DecodeWord, page 681 and Chapter 20, "Mail Classes," EncodeText, page 682. Chapter 20, "Mail Classes," ErrorDescription, page 697 , Chapter 20, "Mail Classes," ErrorDetails, page 698 , Chapter 20, "Mail Classes," MessageSetNumber, page 701 and Chapter 20, "Mail Classes," MessageNumber, page 700.

DecodeWord

Syntax

```
DecodeWord(WordToDecode, &DecodedWord)
```

Description

Use the DecodeWord method to decode a word. If you want to decode a string, use the DecodeText method instead.

The word is decoded based on the values used with the EncodeWord method.

The decoded word is placed in the *&DecodedWord* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return all the substitution strings used to format the error message.

Parameters

Parameter	Description
WordToDecode	Specify the word that is to be decoded, as a string.
&DecodedWord	Specify a string variable, used to hold the decoded word.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A boolean value: true if successful, false otherwise.

See Also

Chapter 20, “Mail Classes,” DecodeText, page 680 and Chapter 20, “Mail Classes,” EncodeWord, page 683. Chapter 20, “Mail Classes,” ErrorDescription, page 697 , Chapter 20, “Mail Classes,” ErrorDetails, page 698 , Chapter 20, “Mail Classes,” MessageSetNumber, page 701 and Chapter 20, “Mail Classes,” MessageNumber, page 700.

EncodeText

Syntax

```
EncodeText(TextToEncode, charset, EncodingStyle, &EncodedText)
```

Description

Use the EncodeText method to encode text. If you only want to encode a single word, use the EncodeWord method instead.

The encoded text is placed in the *&EncodedText* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return all the substitution strings used to format the error message.

Using Encoding Styles

You can specify either “B” or “Q” for the *EncodingStyle* parameter, indicating either Base64 or “Quoted-Printable” content-transfer-encoding, respectively.

PeopleSoft recommends using the “Q” encoding when most of the characters to be encoded are in the ASCII character set; otherwise, you should use the “B” encoding. However, a mail reader that claims to recognize encoded text must be able to accept either encoding for any character set that it supports.

Both Base64 and “Quoted-Printable” content-transfer-encoding are defined by RCF 1521.

See http://www.freesoft.org/CIE/RFC/1522/4.htm

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TextToEncode	Specify the text that is to be encoded, as a string.
charset	Specify the character set to be used for encoding the text, as a string. See *Enterprise PeopleTools 8.49 PeopleBook: Global Technology*, "Selecting and Configuring Character Sets and Language Input and Output," Character Sets in the PeopleSoft Pure Internet Architecture.
EncodingStyle	Specify the encoding style. Valid values are: • B — for Base64 encoding. • Q — for "Quoted-Printable" content- transfer-encoding.
&EncodedText	Specify a string variable used to hold the encoded text.

Returns

A boolean value: true if successful, false otherwise.

See Also

Chapter 20, "Mail Classes," DecodeText, page 680 and Chapter 20, "Mail Classes," EncodeWord, page 683. Chapter 20, "Mail Classes," ErrorDescription, page 697 , Chapter 20, "Mail Classes," ErrorDetails, page 698 , Chapter 20, "Mail Classes," MessageSetNumber, page 701 and Chapter 20, "Mail Classes," MessageNumber, page 700.

EncodeWord

Syntax

```
EncodeWord(WordToEncode, charset, EncodingStyle, &EncodedWord)
```

Description

Use the EncodeWord method to encode a word. If you want to encode more than a single word, use the EncodeText method.

The encoded word is placed in the *&EncodedWord* parameter.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return all the substitution strings used to format the error message.

Copyright © 1988-2007, Oracle. All rights reserved.

Using Encoding Styles

You can specify either “B” or “Q” for the *EncodingStyle* parameter, indicating either Base64 or “Quoted-Printable” content-transfer-encoding, respectively.

PeopleSoft recommends using the “Q” encoding when most of the characters to be encoded are in the ASCII character set; otherwise, you should use the “B” encoding. However, a mail reader that claims to recognize encoded text must be able to accept either encoding for any character set that it supports.

Both Base64 and “Quoted-Printable” content-transfer-encoding are defined by RCF 1521.

See http://www.freesoft.org/CIE/RFC/1522/4.htm

Parameters

Parameter	Description
WordToEncode	Specify the word that is to be encoded, as a string.
charset	Specify the character set to be used for encoding the word, as a string. See *Enterprise PeopleTools 8.49 PeopleBook: Global Technology*, “Selecting and Configuring Character Sets and Language Input and Output,” Character Sets in the PeopleSoft Pure Internet Architecture.
EncodingStyle	Specify the encoding style. Valid values are: • B — for Base64 encoding. • Q — for "Quoted-Printable" content- transfer-encoding.
&EncodedWord	Specify a string variable, used to hold the encoded word.

Returns

A boolean value: true if successful, false otherwise.

See Also

Chapter 20, “Mail Classes,” DecodeWord, page 681 and Chapter 20, “Mail Classes,” EncodeText, page 682. Chapter 20, “Mail Classes,” ErrorDescription, page 697 , Chapter 20, “Mail Classes,” ErrorDetails, page 698 , Chapter 20, “Mail Classes,” MessageNumber, page 700 and Chapter 20, “Mail Classes,” MessageSetNumber, page 701.

IsEmailServerAvailable

Syntax

IsEmailServerAvailable(*server, port, user, password*)

Description

Use the IsEmailServerAvailable method to check if the email server specified with the SMTP settings in the application server configuration file is available.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

• ErrorDescription

 Copyright © 1988-2007, Oracle. All rights reserved.

- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

In addition you can use the GetErrorMsgParam method to return all the substitution strings used to format the error message.

Parameters

Parameter	Description
server	Specify the name of the mail server handling the specified user account, as a string.
port	Specify the port of the mail server handling the specified user account, as an integer.
user	Specify the user name on the mail server, such as “support” or “john_doe”, as a string.
password	Specify the password associated with the specified user name, as a string.

Returns

A boolean value: true if the server is available, false otherwise.

See Also

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, “Setting Application Server Domain Parameters”
Chapter 20, “Mail Classes,” ErrorDescription, page 697 , Chapter 20, “Mail Classes,” ErrorDetails, page 698 , Chapter 20, “Mail Classes,” MessageSetNumber, page 701 and Chapter 20, “Mail Classes,” MessageNumber, page 700.

ValidateAddress

Syntax

```
ValidateAddress(EmailAddress)
```

Description

Use the ValidateAddress method to validate an email address. It checks the syntax of the email address. It does not verify the domain name or the validity of the user name.

This method returns true if successful. If the return value is false, the following MCFMailUtil properties are set accordingly:

- ErrorDescription
- ErrorDetails
- ErrorMsgParamsCount
- MessageNumber
- MessageSetNumber

Copyright © 1988-2007, Oracle. All rights reserved.

In addition you can use the GetErrorMsgParam method to return all the substitution strings used to format the error message.

Parameters

Parameter	Description
EmailAddress	Specify the email address you would like to verify, as a string.

Returns

A boolean value: true if the specified email address is valid, false otherwise.

See Also

Chapter 20, "Mail Classes," ErrorDescription, page 697 , Chapter 20, "Mail Classes," ErrorDetails, page 698 , Chapter 20, "Mail Classes," MessageSetNumber, page 701 and Chapter 20, "Mail Classes," MessageNumber, page 700.

MCFMultiPart Class

Use the MCFMultiPart class to create complex emails with attachments, HTML content, and so on.

MCFMultiPart Class Methods

In this section, we discuss the MCFMultiPart class methods. The methods are described in alphabetical order.

AddBodyPart

Syntax

AddBodyPart(*&bodyPart*)

Description

Use the AddBodyPart method to add the specified body part to this multipart email.

Parameters

Parameter	Description
&bodyPart	Specify an already instantiated MCFBodyPart object.

Returns

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

GetBodyPart

Syntax

```
GetBodyPart(Index)
```

Description

Use the GetBodyPart method to return a reference to the body part specified by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the body part you want to access.

Returns

A reference to an MCFBodyPart object.

See Also

Chapter 20, "Mail Classes," MCFBodyPart Class Methods, page 644

GetContentType

Syntax

```
GetContentType()
```

Description

Use the GetContentType method to determine the content type.

Parameters

None.

Returns

A string.

GetCount

Syntax

```
GetCount()
```

Description

Use the GetCount method to return the number of parts in the MCFMultiPart object.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A number.

MCFMultiPart Class Property

In this section we discuss the MCFMultiPart class property.

SubType

Description

Use this property to specify the subtype of the MCFMultiPart object. The values are:

- alternative
- related
- mixed

The default value is mixed.

This property is read-write.

MCFOutboundEmail Class

This class inherits many properties from the MCFEmail and MCFPart classes. Generally, you only use the MCFOutboundEmail and MCFInboundEmail classes.

MCFOutboundEmail Class Methods

In this section, we discuss the MCFOutboundEmail class methods. The methods are discussed in alphabetical order.

AddAttachment

Syntax

AddAttachment({*FilePath* | *FileURL*}, *FilePathType*, *FileName*, *FileDescr*, *OverrideContentType*, *OverrideCharset*)

Description

Use the AddAttachment method to add an attachment to this email.

Use the *FilePathType* parameter to specify if the file path is relative or absolute. If it's an absolute path, the file path could be a file on the local machine, a network folder, or a fully-qualified URL.

 Copyright © 1988-2007, Oracle. All rights reserved.

You can also add an attachment by creating a multipart object, adding the attachments as bodyparts to the multipart object, and then setting the MultiPart property of the MCFOutbounEmail object. The AddAttachment method is provided for convenience.

Parameters

Parameter	Description
FilePath | *FileURL*	You can either specify the file path to the file, or a URL to the file. Depending on the *FilePathType* parameter, you may specify either a relative or absolute file path to the file. This parameter should include the file name and extension of the file. If you specify a URL address where the file is located, it must be an absolute URL. This URL should include the actual name of the file. If you specify a URL address, you must specify the *FilePathType* parameter as %FilePath_Absolute.
FilePathType	Specify the path type for the file, whether it is an absolute or relative path. The values are: • %FilePath_Absolute — the file path is an absolute path. • %FilePath_Relative — the file path is a relative path.
FileName	Specify the name of the file that you want to attach, as a string. You must include the file extension as well.
Description	Specify a description of the file.
OverrideContentType	The system detects the content type of the attachment using the file location and name. Specifying *OverrideContentType* overrides the system detected content type. The character set can also be included in the ContentType. For example, “text/plain; charset=US-ASCII”
OverrideCharset	Specify a character set to be used to override the existing character set.

Returns

None.

AddHeader

Syntax

AddHeader(*HeaderName, HeaderValue*)

Description

Use the AddHeader method to add a header to the email. This method allows for email server customizations. Some commonly used headers are already exposed as properties, such as From, To, Subject, ContentType and ContentLanguage. Advanced applications can adapt this technique to meet their own requirements. These headers can be either standard SMTP headers or custom headers starting with "X-".

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
HeaderName	Specify the name of the header that you want to add. This parameter takes a string value.
HeaderValue	Specify the actual text of the header, as a string.

Returns

None.

Example

```
  Local PT_MCF_MAIL:MCFOutboundEmail &email = create PT_MCF_MAIL:MCFOutboundEmail⇒
();
   Local string &TestName = "Custom Header";

   &email.From = &def.From;
   &email.Recipients = &def.Recipients;
   &email.SMTPServer = &def.SMTPServer;
   &email.Subject = &TestName;
   &email.Text = &TestName;

   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP2");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP3");
   &email.AddHeader("X-Mailer", "CRM Sales Application 8.9 SP4");

   &res = &email.Send();
```

See Also

Chapter 20, “Mail Classes,” GetHeader, page 690 , Chapter 20, “Mail Classes,” GetHeaderCount, page 691 , Chapter 20, “Mail Classes,” GetHeaderName, page 691 , Chapter 20, “Mail Classes,” GetHeaderNames, page 692 and Chapter 20, “Mail Classes,” GetHeaderValues, page 692.

GetHeader

Syntax

GetHeader(*HeaderName*)

Description

Use the GetHeader method to return the value of the specified header.

Parameters

Parameter	Description
HeaderName	Specify the name of the header that you want to access the values for, as a string. The matching of header names is case insensitive.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

An array of string containing the header values. If the header is not present, returns an array of length zero.

See Also

Chapter 8, “Array Class,” page 215
Chapter 20, “Mail Classes,” AddHeader, page 689 , Chapter 20, “Mail Classes,” GetHeaderCount, page 691 , Chapter 20, “Mail Classes,” GetHeaderName, page 691 , Chapter 20, “Mail Classes,” GetHeaderNames, page 692 and Chapter 20, “Mail Classes,” GetHeaderValues, page 692.

GetHeaderCount

Syntax

```
GetHeaderCount()
```

Description

Use the GetHeaderCount method to return the number of headers.

Parameters

None.

Returns

An integer, representing the number of headers present.

See Also

Chapter 20, “Mail Classes,” AddHeader, page 689 , Chapter 20, “Mail Classes,” GetHeader, page 690 , Chapter 20, “Mail Classes,” GetHeaderName, page 691 , Chapter 20, “Mail Classes,” GetHeaderNames, page 692 and Chapter 20, “Mail Classes,” GetHeaderValues, page 692.

GetHeaderName

Syntax

```
GetHeaderName(Index)
```

Description

Use the GetHeaderName to return the name of the header that is located at *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the header you want to access.

Returns

A string.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 20, “Mail Classes,” AddHeader, page 689 , Chapter 20, “Mail Classes,” GetHeaderCount, page 691 , Chapter 20, “Mail Classes,” GetHeaderNames, page 692 and Chapter 20, “Mail Classes,” GetHeaderValues, page 692.

GetHeaderNames

Syntax
```
GetHeaderNames()
```

Description

Use the GetHeaderNames method to return an array containing the names of all the headers.

Parameters

None.

Returns

An array of string.

See Also
Chapter 20, “Mail Classes,” AddHeader, page 689 , Chapter 20, “Mail Classes,” GetHeader, page 690 , Chapter 20, “Mail Classes,” GetHeaderName, page 691 , Chapter 20, “Mail Classes,” GetHeaderCount, page 691 and Chapter 20, “Mail Classes,” GetHeaderValues, page 692.

GetHeaderValues

Syntax
```
GetHeaderValues(Index)
```

Description

Use the GetHeaderValues method to return the value for the header located at the position specified by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the header whose value you want to access.

Returns

An array of string.

See Also
Chapter 20, “Mail Classes,” AddHeader, page 689 , Chapter 20, “Mail Classes,” GetHeader, page 690 , Chapter 20, “Mail Classes,” GetHeaderName, page 691 , Chapter 20, “Mail Classes,” GetHeaderCount, page 691 and Chapter 20, “Mail Classes,” GetHeaderNames, page 692.

 Copyright © 1988-2007, Oracle. All rights reserved.

Send

Syntax

```
Send()
```

Description

Use the Send method to send the email.

When you call the Send method, the following properties might be set based on the type of error:

- InvalidAddresses
- MessageNumber (from the PeopleTools Message Catalog)
- MessageSetNumber (from the PeopleTools Message Catalog)
- ValidSentAddresses
- ValidUnsentAddresses

In addition, a list of substitution strings for the error message can be accessed bythe ErrorMsgParamsCount property and the GetErrorMsgParam method.

Parameters

None.

Returns

A number indicating the status of the send. Values are:

Numeric Value	Constant Value	Description
0	%ObEmail_ FailedBeforeSending	Email failed before being sent.
1	%ObEmail_Delivered	Email was delivered.
2	%ObEmail_NotDelivered	Email was not delivered.
3	%ObEmail_PartiallyDelivered	Email has only been partially delivered. Only some of the addresses in the list of addresses were delivered to.
-1	%ObEmail_ SentButResultUnknown	Email was sent but whether it was successful or not is not known.

See Also

Chapter 20, “Mail Classes,” InvalidAddresses, page 699 , Chapter 20, “Mail Classes,” MessageNumber, page 700 , Chapter 20, “Mail Classes,” ValidSentAddresses, page 706 and Chapter 20, “Mail Classes,” ValidUnsentAddresses, page 706.

SetSMTPParam

Syntax

```
SetSMTPParam(ParamName, ParamValue)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetSMTPParam method to set parameters for the SMTP session to be used for sending the email. If you are only sending out one email, use this method. If you are sending many emails, use the SMTPSession class and set the parameters once for all the emails you are sending out.

Note. You should use this method before you use the Send method.

Parameters

Parameter	Description
ParamName	Specify the name of the parameter you want to overwrite.
ParamValue	Specify the value for the named parameter that you want used instead of the existing value.

Returns

None.

Example

```
&email.setSMTPParam("mail.mime.decodetext.strict", "false");
&email.setSMTPParam("mail.mime.address.strict", "false");
&email.setSMTPParam("mail.debug, "true");
```

See Also

Chapter 20, “Mail Classes,” SMTPSession Class, page 706

MCFOutboundEmail Class Properties

In this section we discuss the MCFOutboundEmail class properties. The properties are described in alphabetical order.

BackupSMTPPort

Description

Use this property to specify the port number of the backup SMTP server. This is an optional property when creating an email. If you don’t specify a value, the default value is 25.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

BackupSMTPServer

Description

Use this property to specify the server name of the backup SMTP server. The system tries to connect to the backup server if the primary SMTPserver is not available.

This property is read-write.

BackupSMTPUserName

Description

Use this property to specify the user name used to sign onto the backup SMTP server. This user name is used for authentication when sending mail using the backup mail server.

This property is read-write.

BackupSMTPUserPassword

Description

Use this property to specify the user password used for signing onto the backup SMTP server. The BackupSMTPUserName and password are used for the authentication when sending mail using the backup mail server.

This property is read-write.

BCC

Description

Use this property to specify comma-separated list of addresses of all the blind carbon copy recipients of this email. (A blind carbon copy is a copy of the email that contains all of the text of the email, but does not show any of the other email addresses to which this email was sent.)

This property is read-write.

See Also

Chapter 20, “Mail Classes,” CC, page 696

BounceTo

Description

Use this property to specify the email address the system should direct all bounced mail to.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

CC

Description

Use this property to specify a comma-separated list of addresses that copies of this email are to be sent to (carbon copy.)

This property is read-write.

See Also

Chapter 20, “Mail Classes,” BCC, page 695

Charset

Description

Use this property to specify the character set for the text or the attachment.

You can also specify this property using the SetAttachmentContent method, or the ContentType property.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” SetAttachmentContent, page 648

Chapter 20, “Mail Classes,” ContentType, page 650

ContentLanguage

Description

Use this property to set the language of this email. More than one language can be set in a comma-separated list. Use the standard abbreviated language names, such as en, fr, de, da, and so on.

You can also include the country codes when you specify the language, such as en-us.

This property is read-write.

See Also

http://www.w3.org/WAI/ER/IG/ert/iso639.htm
http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

ContentType

Description

Use this property to specify the content type of this body part.

You can also specify the content type with the SetAttachmentConent method.

You can also use this property to specify the character set. For example, “text/plain; charset=US-ASCII”.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, “Mail Classes,” SetAttachmentContent, page 648

Chapter 20, “Mail Classes,” Charset, page 650

DefaultCharSet

Description

Use this property to specify the character set to be applied to all the parts of the email.

This property is read-write.

Description

Description

Use this property to specify a description of the file attachment associated with this email. If there is no file attachment, this property is ignored.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” AddAttachment, page 688

Disposition

Description

Use this property to specify how the body part is presented in the received mail.

Some email clients ignore the setting of this property and present body parts either inline or as file attachments.

Values for this property are:

Value	Description
Attachment	The body part is shown as a file attachment.
Inline	The body part is shown in the body itself.

This property is read-write.

ErrorDescription

Description

If an error occurs, use the ErrorDescription property to get the description of the error.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, “Mail Classes,” ErrorDetails, page 698

ErrorDetails

Description

If an error occurs, use the ErrorDetails property to get the details of the error.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” ErrorDescription, page 697

Filename

Description

If you are adding an attachment to the email, you can specify the name of the file using this property.

PeopleSoft recommends that you keep the file extension specified with this property the same as the original extension found in the file path, otherwise the client applications may not be able to display it properly.

This property is read-write.

FilePath

Description

Specify the path for the file that contains the contents of this email.

Whether this is a relative or absolute path depends on the FilePathType property.

You can also specify a URL to the file using this property, if the FilePathType property is specified as %FilePath_Absolute.

If you specify a value for this property, the ‘Text’ content is ignored (when used with the ContentType property.)

This property is read-write.

See Also

Chapter 20, “Mail Classes,” FilePathType, page 698

FilePathType

Description

Use this property to specify whether the path specified with the FilePath property is a relative or absolute path. The values for this property are:

 Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
%FilePath_Relative	The file path specified with the FilePath property is a relative path.
%FilePath_Absolute	The file path specified with the FilePath property is either an absolute path to a file, or a URL to a file.

If you specify a relative path, the file must be available in the FILES folder of application server folder.

If you specify an absolute path, the file could be on the local machine, in any network folder, or a URL.

If you specify a value for this property, the Text property is ignored.

This property is read-write.

From

Description

Use this property to specify the email address of the person sending the email.

You can specify more than one address as from in a comma-separated list.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” Recipients, page 702 , Chapter 20, “Mail Classes,” Sender, page 703 and Chapter 20, “Mail Classes,” ReplyTo, page 702.

Importance

Description

Use this property to specify the value of the importance header field.

The importance can be set to the following:

- low
- normal
- high

If the Priority property is not set, the system automatically sets it to the corresponding values like 5, 3 and 1.

This property is read-write.

InvalidAddresses

Description

Use this property to get a list of the invalid email addresses where the mail could not be sent. The email addresses are in a string, separated by commas.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 20, “Mail Classes,” ValidSentAddresses, page 706 and Chapter 20, “Mail Classes,” ValidSentAddresses, page 706.

IsAuthenticationReqd

Description

Use this property to specify if authentication is required or not. If the SMTP server does not support authentication or authentication is not enabled, this property is ignored.

This property takes a boolean value: true if authentication is required, false otherwise.

This property is read-write.

IsOkToSendPartial

Description

Use this property to specify if the email can be sent to only some of the specified recipients or if it must be sent to all. This property takes a boolean value: true if a partial list of addresses is acceptable, false otherwise. If the email must go out to all those listed, or to none at all, specify false.

This property is read-write.

Considerations Using IsOkToSendPartial

If the syntax of the email address is wrong, no mail is sent, regardless of the value of this property. The error is reported.

If the syntax of the email addresses is correct but the email server is unable to send mail to some addresses, the value of this property is used to either send or not send to the partial list of email addresses.

If the syntax of all the email addresses are correct and the email server is able to dispatch mail to all the addresses, no error is reported immediately. Later, if some addresses are found invalid by the recipient email server or gateway, the mail is sent back. In this case the source email server has no way to find the invalidity of the email address in advance.

IsReturnReceiptReqd

Description

Use this property to specify if the receiver must send acknowledgement that the email was received when it’s received. This property takes a boolean value: true if the return receipt is required, false otherwise.

This property is read-write.

MessageNumber

Description

This property returns the message number of the error message associated with this email. The message number is associated with messages in the PeopleTools Message Catalog.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

MessageSetNumber

Description

This property returns the message set number of the error message associated with the email.

This property is read-only.

MultiPart

Description

The email can contain simple text, one attachment, or a MultiPart object.

If you have assigned a MultiPart object using this property, the text and attachment related properties are ignored.

This property is read-write.

Example

```
Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.SubType = "alternative; differences=Content-type";

&mp.AddBodyPart(&text);
&mp.AddBodyPart(&html);

&email.MultiPart = &mp;
```

Priority

Description

Use this property to specify the priority of this email.

The values are:

1. Highest Priority
2. High Priority
3. Normal
4. Low Priority
5. Lowest Priority

The default value is 3, that is, normal priority.

This value set with this property is used to set to a header X-Priority field, as this is the most common but non-standard field to show priority. In addition, the corresponding values are set in two header fields: X-MSMail-Priority and Priority. The headers X-Priority or X-MSMail-Priority are set to the corresponding value only if the user does not specify a value.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Recipients

Description

Use this property to specify the email addresses of all the main recipients to whom the email is being sent. All the addresses are specified in a comma-separated string.

This property is read-write.

RefIDs

Description

Use this property to specify a value for the REFERENCES header field of the message.

This property is read-write.

ReplyIDs

Description

Use this property to specify a value for the IN-REPLY-TO header field of the message.

This property is read-write.

ReplyTo

Description

Use this property to specify the email address where the reply should be sent. You do not need to specify a value for this property if the value is the same as the From property.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” From, page 699

ResultOfSend

Description

This property returns the result of the Send method. The values are:

Numeric Value	Constant Value	Description
0	%ObEmail_ FailedBeforeSending	Email failed before being sent.
1	%ObEmail_Delivered	Email was delivered.
2	%ObEmail_NotDelivered	Email was not delivered.
3	%ObEmail_PartiallyDelivered	Email has only been partially delivered. Only some of the addresses in the list of addresses were delivered to.
-1	%ObEmail_ SentButResultUnknown	Email was sent but whether it was successful or not is not known.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

See Also

Chapter 20, “Mail Classes,” Send, page 693

Sender

Description

Use this property to specify the address of the author of the message. You do not need to specify a value for this property if the value for the From property is the same.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” From, page 699

Sensitivity

Description

Use this property to specify the value of the sensitivity header field.

The values are:

- personal
- private
- company-confidential

This property is read-write.

SMTPPort

Description

Use this property to specify the port number of SMTP server. This is an optional property. If you don’t specify a value for this property, the default value is 25.

This property is read-write.

SMTPServer

Description

Use this property to specify the SMTP server to be used when sending this email.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” SMTPPort, page 703 , Chapter 20, “Mail Classes,” SMTPUserName, page 704 , Chapter 20, “Mail Classes,” SMTPUserPassword, page 704 and Chapter 20, “Mail Classes,” SMTPSession Class, page 706.

Copyright © 1988-2007, Oracle. All rights reserved.

SMTPUserName

Description

Use this property to specify the user name to be used for logging into the SMTP server.

You should only use this property if the SMTP server allows authentication. If the SMTP server does not allow authentication, setting this property has no effect.

This property is read-write.

See Also

Chapter 20, "Mail Classes," SMTPPort, page 703 , Chapter 20, "Mail Classes," SMTPServer, page 703 and Chapter 20, "Mail Classes," SMTPUserPassword, page 704.

SMTPUserPassword

Description

Use this property to specify the password for the SMTP user. This property is used with the SMTPUserName property.

This property is read-write.

See Also

Chapter 20, "Mail Classes," SMTPPort, page 703 , Chapter 20, "Mail Classes," SMTPServer, page 703 and Chapter 20, "Mail Classes," SMTPUserName, page 704.

StatusNotifyOptions

Description

Use this property to specify when the system should notify the sender. The values are separated by commas in a string. The values are:

Value	Description
SUCCESS	Send notification if the email is delivered successfully.
FAILURE	Send notification if the email is not delivered successfully.
DELAY	Send notification if delivery of the email is delayed.

This property is read-write.

See Also

Chapter 20, "Mail Classes," StatusNotifyReturn, page 704 and Chapter 20, "Mail Classes," TimeToWaitForResult, page 705.

StatusNotifyReturn

Description

Use this property to specify what should be returned as the status notification. You can return either the header or the full message. This option is meaningful only if StatusNotifyOptions property is set.

 Copyright © 1988-2007, Oracle. All rights reserved.

The values are:

Value	Description
HDRS	Return only the headers of the email.
FULL	Return the full email message.

This property is read-write.

Subject

Description

Use this property to specify the subject of the email. A subject can only contain 254 characters.

This property is read-write.

Text

Description

Use this property to specify the text for the email.

This property is read-write.

TimeToWaitForResult

Description

Use this property to specify the number of milliseconds to wait for the result of send email process. If the result comes back within this time, the returned value is a positive number. Otherwise, %ObEmail_SentButResultUnknown (or -1) is returned.

This property is read-write.

See Also

Chapter 20, "Mail Classes," StatusNotifyOptions, page 704 and Chapter 20, "Mail Classes," StatusNotifyReturn, page 704.

UsedDefaultConfig

Description

Use this property to determine whether the default configuration as specified in application server configuration file was used or not.

This property returns a boolean value: true if the default configuration was used, false if the value specified with the MCFOutboundEmail or the SMTPSession object was used.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

UsedPrimaryServer

Description

Use this property to determine if the default SMTP server was used, or the backup server.

This property returns a boolean value: true if the default SMTP server was used, false if the backup server was used.

This property is read-only.

ValidSentAddresses

Description

This property returns the list of addresses that were valid and that the email was sent to. The email addresses are in a string, separated by commas.

This property is read-only.

See Also

Chapter 20, "Mail Classes," InvalidAddresses, page 699 and Chapter 20, "Mail Classes," ValidUnsentAddresses, page 706.

ValidUnsentAddresses

Description

Use this property to get a list of the email addresses that are valid, yet the system was unable to send to, due to system problems. The email addresses are separated by commas.

This property is read-only.

See Also

Chapter 20, "Mail Classes," InvalidAddresses, page 699 and Chapter 20, "Mail Classes," ValidSentAddresses, page 706.

SMTPSession Class

Use the SMTPSession class to when you want to send more than one email using the same SMTP session. If you are only sending a single email, use the MCFOutboundEmail class instead.

SMTPSession Class Methods

In this section we describe the SMTPSession Class methods. The methods are described in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Send

Syntax

Send(*&Email*)

Description

Use the Send method to send an MCFOutboundEmail object as email.

By default, the system sends the email to the SMTP server using the properties set in the application server configuration file. You can specify different setup parameters by supplying values for the SMTPSession object properties, such as Server, BackupServer, and so on.

Parameters

Parameter	Description
&Email	Specify an already instantiated MCFOutboundEMail object to be sent.

Returns

A constant indicating the status of the method. The values are:

Value	Description
%ObEmail_Delivered	The email was delivered.
%ObEmail_NotDelivered	The email wasn't delivered.
%ObEmail_PartiallyDelivered	The email was only partially delivered, that is, it was only delivered to some of the recipients.
%ObEmail_ FailedBeforeSending	The email wasn't delivered.
%ObEmail_ SentButResultUnknown	The email was sent but the result is unknown. The TimeToWaitForResult value was not sufficient to get the result of the send process.

See Also

Chapter 20, "Mail Classes," SendAll, page 707

SendAll

Syntax

SendAll(*&Emails*)

Description

Use the SendAll method to send a number of emails, all using the same SMTP session and session properties.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&Emails	Specify an array of MCFOutboundEmail objects.

Returns

An array of number. Each item in the array indicates the status of an individual email object. Values are:

Value	Description
%ObEmail_Delivered	The email was delivered.
%ObEmail_NotDelivered	The email wasn't delivered.
%ObEmail_PartiallyDelivered	The email was only partially delivered, that is, it was only delivered to some of the recipients.
%ObEmail_ FailedBeforeSending	The email wasn't delivered.
%ObEmail_ SentButResultUnknown	The email was sent but the result is unknown. The TimeToWaitForResult value was not sufficient to get the result of the send process.

See Also

Chapter 20, "Mail Classes," Send, page 707

SetSMTPParam

Syntax

```
SetSMTPParam(ParamName, ParamValue)
```

Description

Use the SetSMTPParam method to set additional parameters for the SMTP session.

You must use this method before you use any Send method.

Parameters

Parameter	Description
ParamName	Specify the name of the parameter you want to overwrite.
ParamValue	Specify the value for the named parameter that you want used instead of the existing value.

Returns

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

SMTPSession Class Properties

In this section we describe the SMTPSession class properties. The properties are described in alphabetical order.

BackupPort

Description

Use this property to specify the port number of backup SMTP server. This is an optional property. If you don't specify a value for this property, the default value is 25.

This property is read-write.

BackupServer

Description

Use this property to specify the name of the backup SMTP server. The system tries connecting to the backup server if the primary SMTP server is not available.

This property is read-write.

BackupUserName

Description

Use this property to specify the user name used to sign onto the backup SMTP server.

This user name is used for authentication when sending mail using the backup mail server.

This property is read-write.

BackupUserPassword

Description

Use this property to specify the user password used to sign onto the backup SMTP server.

This user name and password are used for authentication when sending mail using the backup mail server.

This property is read-write.

IsAuthenticationReqd

Description

Use this property to specify if authentication is required or not. If the SMTP server does not support authentication or authentication is not enabled, this property is ignored.

This property takes a boolean value: true if authentication is required, false otherwise.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Port

Description

Use this property to specify the SMTP port number to be used for sending this email. This property takes a numeric value. This property is optional. If you don't specify a value for this property, the default value of 25 is used.

This property is read-write.

See Also

Chapter 20, "Mail Classes," SMTPServer, page 703 , Chapter 20, "Mail Classes," SMTPUserName, page 704 , Chapter 20, "Mail Classes," SMTPUserPassword, page 704 and Chapter 20, "Mail Classes," SMTPSession Class, page 706.

Server

Description

Use this property to specify the name of the SMTP server to be used when sending this email.

This property is read-write.

See Also

Chapter 20, "Mail Classes," SMTPPort, page 703 , Chapter 20, "Mail Classes," SMTPUserName, page 704 , Chapter 20, "Mail Classes," SMTPUserPassword, page 704 and Chapter 20, "Mail Classes," SMTPSession Class, page 706.

UsedDefaultConfig

Description

Use this property to determine whether the default configuration as specified in application server configuration file was used or not.

This property returns a boolean value: true if the default configuration was used, false if the value specified with the MCFOutboundEmail or the SMTPSession object was used.

This property is read-only.

UsedPrimaryServer

Description

Use this property to determine if the default SMTP server was used, or the backup server.

This property returns a boolean value: true if the default SMTP server was used, false if the backup server was used.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

UserName

Description

Use this property to specify the user name to be used for logging into the SMTP server.

You should only use this property if the SMTP server allows authentication. If the SMTP server does not allow authentication, setting this property has no effect.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” SMTPPort, page 703 , Chapter 20, “Mail Classes,” SMTPServer, page 703 and Chapter 20, “Mail Classes,” SMTPUserPassword, page 704.

UserPassword

Description

Use this property to specify the password for the SMTP user. This property is used with the SMTPUserName property.

This property is read-write.

See Also

Chapter 20, “Mail Classes,” SMTPPort, page 703 , Chapter 20, “Mail Classes,” SMTPServer, page 703 and Chapter 20, “Mail Classes,” SMTPUserName, page 704.

Mail Classes Examples

The following are some examples of how to create the most common types of email.

Creating Text Email

The following code example creates and sends a very simple email, then tests the results.

```
import PT_MCF_MAIL:*;

/*-- Create an outbound email object --*/

Local PT_MCF_MAIL:MCFOutboundEmail &eMail =
create PT_MCF_MAIL:MCFOutboundEmail();

/*-- Initialize the usual fields of an email --*/

&eMail.From = &FromAddress;
&eMail.Recipients = &ToList;
&eMail.Subject = &Subject;
&eMail.Text = &MailBody;

/*-- The send method uses the default SMTP parameters as set in the app server⇒
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
 configuration file.

This send method makes a connection to the SMTP server, sends the mail and then⇒
 disconnects.

The result are returned as a number corresponding to the possible values.

The list of ValidSent, InvalidSent and Invalid addresses are returned in the email⇒
 object itself
----*/

Local integer &resp = &eMail.Send();
Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating Email and Overriding SMTP Settings

The following code example creates a text email and overrides the SMTP settings as found in the application server configuration file.

```
import PT_MCF_MAIL:*;

/*-- Create an email object by setting individual parameters ---*/
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local PT_MCF_MAIL:MCFOutboundEmail &eMail =
create PT_MCF_MAIL:MCFOutboundEmail();

&eMail.Recipients = &ToList; /* comma separated list of email addresses */
&eMail.CC = &CCList; /* comma separated list of email addresses */
&eMail.BCC = &BCCList; /* comma separated list of email addresses */
&eMail.From = &FromAddress; /* from email address */
&eMail.ReplyTo = &ReplyToAddress; /* in case the reply is to be sent to a⇒
 different email address */
&eMail.Sender = &SenderAddress; /* If different from the "from" address */

&eMail.Subject = &Subject; /* email subject line */
&eMail.Text = &MailBody; /* email body text */


/*-- Override the default SMTP parameters specified in app server configuration⇒
 file ----*/

&eMail.SMTPServer = "psp-smtpg-01";
&eMail.SMTPPort = 10266; /*-- Usually this is 25 by default */

Local integer &resp = &eMail.Send();

/* Now check the &resp for the result */

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      &done = False;
      Break;
   End-Evaluate;
```

Creating HTML Email

The following example creates an HTML email.

```
import PT_MCF_MAIL:*;

/*-- Create an email object by setting individual parameters
---*/


Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;

&email.Text =
"<html><body><H1><b>Hi there!</b></H1><P>We are ready.</body></html>";

   &email.ContentType = "text/html";

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating Multi-Part Email With Both Text and HTML Parts

The following code example creates an email with both HTML and text sections.

The email client on the target host must be able to detect the HTML and text parts in the email and display the part that the client is configured to display.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();
Local string &TestName = "Text and its alternate html body";

&email.From = &FromAddress;
&email.Recipients = &ToList;
&email.Subject = &Subject;

Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();
&text.Text = "Hi There";

Local PT_MCF_MAIL:MCFBodyPart &html = create PT_MCF_MAIL:MCFBodyPart();
&html.Text =
"<html><BODY><H1>EMail test with HTML content</H1><b>Hi There</b>" |
"<A href='http://www.peoplesoft.com'>Check this out!</A>" |
"<P></BODY></html>";
&html.ContentType = "text/html";

Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.SubType = "alternative; differences=Content-type";

&mp.AddBodyPart(&text);
&mp.AddBodyPart(&html);

&email.MultiPart = &mp;

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating HTML Email with Images

The following code example creates an HTML email with imbedded images. The content ID for an image can be a reference to a part in the email. In the following code example, the images become a part of the email and are transmitted as attachments to the email.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;


Local string &htmlText =
"<html>" |
"  <head><title></title></head>" |
"  <body>" |
"    <b>A sample jpg</b><br>" |
"    <IMG SRC=cid:23abc@pc27 width=566 height=424><br>" |
"    <b>End of jpg</b>" |
"  </body>" |
"</html>";
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/* In this sample, the htmlText is assembled using | operator
only for the readability.
Specifying just one string can be more efficient
*/

   Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
   &mp.SubType = "related";

   Local PT_MCF_MAIL:MCFBodyPart &html = create PT_MCF_MAIL:MCFBodyPart();
   &html.Text = &htmlText;
   &html.ContentType = "text/html";

   Local PT_MCF_MAIL:MCFBodyPart &image = create PT_MCF_MAIL:MCFBodyPart();

&image.SetAttachmentContent("///file:C:/User/Documentum/XML%20Applications/proddoc⇒
/peoplebook_upc/peoplebook_upc.dtd",
 %FilePath_Absolute, "sample.jpg", "This is a sample image!", "", "");

   &image.AddHeader("Content-ID", "23abc@pc27");

   &mp.AddBodyPart(&html);
   &mp.AddBodyPart(&image);

   &email.MultiPart = &mp;

   Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating Email with Attachments

The following code example creates an email with an attachment.

PeopleSoft recommends that you always provide the proper extension in the file name, otherwise the receiving email client may not be able to associate it with the appropriate application.

Note. There is no automatic restriction of attachment size using the mail classes. To restrict an email based on attachment size, your application must check the size of an attachment before composing and sending the email.

```
import PT_MCF_MAIL:*;

Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

&email.From = &FromAddress;
&email.Recipients = &ToList;
&email.Subject = &Subject;

Local string &plain_text = "Hi there!";
Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();
&text.Text = &plain_text;

Local PT_MCF_MAIL:MCFBodyPart &attach1 = create PT_MCF_MAIL:MCFBodyPart();
&attach1.SetAttachmentContent("Ocean Wave.jpg", %FilePath_Relative,
"Ocean Wave.jpg", "Ocean Wave", "", "");
/* %FilePath_Relative indicates the file is available at Appserver's FILES⇒
 dierctory */

Local PT_MCF_MAIL:MCFBodyPart &attach2 = create PT_MCF_MAIL:MCFBodyPart();
&attach2.SetAttachmentContent("///file:C:/User/Documentum/XML%20Applications⇒
/proddoc/peoplebook_upc/peoplebook_upc.dtd",
 %FilePath_Absolute, "Sample.jpg", "Sample", "", "");
/* The Sample.jpg is available in the "public" folder of my-server machine*/


Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
&mp.AddBodyPart(&text);
&mp.AddBodyPart(&attach1);
&mp.AddBodyPart(&attach2);

&email.MultiPart = &mp;

Local integer &res = &email.Send();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating Email Attachments Specifying a URL

The following code example creates an email attachment specifying a URL for the file location, instead of an absolute or relative path to the file.

```
Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;

   Local string &plain_text = "Hi there!";
   Local PT_MCF_MAIL:MCFBodyPart &text = create PT_MCF_MAIL:MCFBodyPart();

   &text.Text = &plain_text;

Local PT_MCF_MAIL:MCFBodyPart &attach1 = create PT_MCF_MAIL:MCFBodyPart();

&attach1.SetAttachmentContent("http://www.yahoo.com/members_agreement",
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
 %FilePath_Absolute, "hotmail.htm", "Hotmail Home page", "", "");

Local PT_MCF_MAIL:MCFBodyPart &attach2 = create PT_MCF_MAIL:MCFBodyPart();

&attach2.SetAttachmentContent("ftp://www.w3c/docs/somedoc",
 %FilePath_Absolute, "somedoc.htm", "somedoc from w3c", "", "");

   Local PT_MCF_MAIL:MCFMultipart &mp = create PT_MCF_MAIL:MCFMultipart();
   &mp.AddBodyPart(&text);
   &mp.AddBodyPart(&attach1);
   &mp.AddBodyPart(&attach2);

   &email.MultiPart = &mp;

Local integer &res = &email.Send();

Local boolean &done;

Evaluate &resp
   When %ObEmail_Delivered
      /* every thing ok */
      &done = True;
      Break;

   When %ObEmail_NotDelivered
/*-- Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses */
      &done = False;
      Break;

   When %ObEmail_PartiallyDelivered
/* Check &email.InvalidAddresses, &email.ValidSentAddresses
and &email.ValidUnsentAddresses; */
      &done = True;
      Break;

   When %ObEmail_FailedBeforeSending
/* Get the Message Set Number, message number;
Or just get the formatted messages from &email.ErrorDescription,
&email.ErrorDetails;*/

      &done = False;
      Break;
   End-Evaluate;
```

Creating Multiple Emails

You can create several emails and send them all in a single connection to the SMTP server. However, your system administrator may also need to configure the SMTP server with longer connection time-outs if you are sending multiple emails.

 Copyright © 1988-2007, Oracle. All rights reserved.

```
import PT_MCF_MAIL:*;

/*-- Create an email object by setting individual parameters
---*/
 Local array of PT_MCF_MAIL:MCFOutboundEmail &mails;
 Local PT_MCF_MAIL:MCFOutboundEmail &email;

 Local PT_MCF_MAIL:SMTPSession &commonSession =
create PT_MCF_MAIL:SMTPSession();

   &email = &commonSession.CreateOutboundEmail();
   &email.From = &FromAddress;
   &email.Recipients = &ToList1;
   &email.Subject = &Subject;
   &email.Text = &MailBody1;

   &mails = CreateArray(&email);

   &email = &commonSession.CreateOutboundEmail();
   &email.From = &FromAddress;
   &email.Recipients = &ToList2;
   &email.Subject = &Subject;
   &email.Text = &MailBody2;
   &mails [2] = &email;

   &email = &commonSession.CreateOutboundEmail();
   &email.From = &FromAddress;
   &email.Recipients = &ToList3;
   &email.Subject = &Subject;
   &email.Text = &MailBody3;
   &mails [3] = &email;

   Local array of integer &allRes = &commonSession.SendAll(&mails);
```

Authenticating Email While Sending

The following code example includes a user name and password to be used by the SMTP server.

```
Local PT_MCF_MAIL:MCFOutboundEmail &email =
create PT_MCF_MAIL:MCFOutboundEmail();

   &email.From = &FromAddress;
   &email.Recipients = &ToList;
   &email.Subject = &Subject;
   &email.Text = &MailBody;

   &email.SMTPServer = "MySMTPServer";
   &email.IsAuthenticationReqd = True;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&email.SMTPUserName = "SomeLoginName";
&email.SMTPUserPassword = "SomeLoginPassword";

Local integer &res = &email.Send();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 21

MCFIMInfo Class

This chapter documents the MCFIMInfo class and covers the following topics:

- Using the MCFIMInfo Class
- Data Type for MCFIMInfo Objects
- Scope of MCFIMInfo Objects
- MCFIMInfo Built-in Functions
- MCFIMInfo Reference

Understanding the MCFIMInfo Class

Use the MCFIMInfo class to launch an instant messaging session from a PeopleSoft page and initiate a chat. A developer can use either the PeopleCode class or a push-button with specific characteristics in Application Designer to initiate an instant messaging session.

Note. PeopleSoft recommends using the Application Designer push-button rather than PeopleCode in most applications. The PeopleCode program determines user presence information from the application server, whereas the push-button determines user presence from the browser. The application server must wait until it has *all* the presence information for that page before it can render the page. Though this processing is multi-threaded, it can still be slower than the push-button.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft MultiChannel Framework, "Configuring Instant Messaging"

Using the MCFIMInfo Class

Only use the MCFIMInfo class if your application requires the flexibility the PeopleCode can provide.

The only supported networks are AOL and Yahoo. SameTime is supported using the Application Designer push-button, but not in PeopleCode.

Instant messages are sent by using the native instant message clients. This means that you can only use this for MicroSoft Windows.

The page from which you launch the instant messaging session must be refreshed in order to update presence status.

Copyright © 1988-2007, Oracle. All rights reserved.

In Netscape 4.x the local instant message clients cannot be started, so instant message users cannot be contacted using Netscape.

Data Type for MCFIMInfo Objects

MCFIMInfo objects are declared as type MCFIMInfo. For example:

```
Local MCFIMInfo &MyChat;
```

Scope of MCFIMInfo Objects

An MCFIMInfo object can be only instantiated from PeopleCode.

Use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message subscription, a Component Interface, and so on.

MCFIMInfo Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMCFIMInfo

MCFIMInfo Class Methods

In this section, we discuss the MCFIMInfo class methods. The methods are discussed in alphabetical order.

AddUser

Syntax

AddUser(*User*)

Description

Use the AddUser method to add a user to the instant messaging session.

Parameters

Parameter	Description
User	Specify the user that you want to add to the instant messaging session, as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the user was successfully added, False otherwise.

CheckAll

Syntax

```
CheckAll()
```

Description

Use the CheckAll method to check the status of users on the instant messaging session.

This method makes the actual network requests to retrieve presence information.

Parameters

None.

Returns

A Boolean: True if the check is successful, False otherwise.

GetAdditionalUserInfo

Syntax

```
GetAdditionalUserInfo()
```

Description

This method returns additional user information.

Parameters

None.

Returns

String.

GetErrorImageName

Syntax

```
GetErrorImageName()
```

Description

Use GetErrorImageName to return the name of the image used for errors.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

String.

GetOffLineImageName

Syntax

```
GetOffLineImageName()
```

Description

Use GetOffLineImageName to return the name of the image used to indicate off line status.

Parameters

None.

Returns

String.

GetOnlineImageName

Syntax

```
GetOnlineImageName()
```

Description

Use GetOnlineImageName to return the name of the image used to indicate online status.

Parameters

None.

Returns

String.

GetUnknownImageName

Syntax

```
GetUnknownImageName()
```

Description

Use GetUnknownImageName to return the image of unknown users.

Parameters

None.

Returns

String.

Copyright © 1988-2007, Oracle. All rights reserved.

GetLaunchURL

Syntax

GetLaunchURL(*User*)

Description

Use the GetLaunchURL method to return the URL that launches the native client and contacts the user. This is a local URL of one of the following forms:

```
aim:. . .
```

or

```
ymsgr:. . .
```

Parameters

Parameter	Description
User	Specify the user that you want to contact via the launch URL.

Returns

A string URL.

GetStatus

Syntax

GetStatus(*User*)

Description

Use the GetStatus method to return the current online status of the user specified by *User*. This is the status at the time of the last call to **CheckAll**.

Parameters

Parameter	Description
User	Specify the name of the user for whom you want to check the status of.

Returns

An integer. The values are:

Value	Description
-1	Network Disabled
0	User offline
1	User online
2	User Unknown
3	Error

Copyright © 1988-2007, Oracle. All rights reserved.

RemoveUser

Syntax

RemoveUser(*User*)

Description

Use the RemoveUser method to remove a user from the instant messaging session.

Parameters

Parameter	Description
User	Specify the name of the user you want to remove from the session, as a string.

Returns

A Boolean value: True if the user was successfully removed, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 22

Message Classes

This chapter provides an overview of the Message classes and discusses the following topics:

- Data type of a message object.
- Scope of a message object.
- Message object population.
- Error handling.
- Message classes reference.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Understanding PeopleSoft Integration Broker"

Understanding Message Classes

You can create the following types of messages using PeopleSoft Pure Internet Architecture::

- *Rowset-based* messages, which use record definitions to create a hierarchical structure for the data. These are generally used for data from applications, pages, components, and so on.
- *Nonrowset-based* messages, which do not use record definitions. These messages can have virtually any content or structure.
- *Container* messages, which are messages made up of one or more part messages.

Use the PeopleCode message classes to instantiate message objects based on existing message definitions, as well as to populate the objects with data and manipulate the data. You can also use PeopleCode to publish a message.

Rowset-based messages are built on top of the rowset, row, record, and field classes, so the PeopleCode written to populate or access those types of messages looks similar to PeopleCode written to populate or access the component buffer.

Nonrowset-based messages contain XML data, and can be accessed using the XmlDoc class methods and properties.

Container messages contain parts. Each part is a separate message. A container message can contain either all rowset-based messages, *or* all nonrowset-based messages.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Messages, Service Operations and Handlers

Message definitions only define the shape of the data contained in a message. A message definition doesn't contain any other type of information, such as if the message is being used synchronously or if it's a response message.

After you create a message definition, you can use it in or more service operations. The service operation contains all the information about how the message is to be used, the routing, the queue if appropriate, and so on.

At least one handler is defined with every service operation. Handlers define additional programming to be used with processing the message associated with the service operation. The handlers can be thought of as corresponding to pre PeopleSoft 8.48 message events.

The PeopleCode for the various handlers is contained in the Integration Broker application classes.

Integration Broker Application Classes

The Integration Broker application classes house the processing logic for asynchronous and synchronous messages. By implementing the Integration Broker application classes, you can reuse code more easily and access the other benefits of using application classes.

The following application classes are defined for PeopleSoft Integration Broker. To access these application classes, in PeopleTools Application Designer, open the PS_PT application package, then open the Integration subpackage.

All of the Integration Broker application classes are defined as interfaces. This means that there is no native implementation of them: you must import them to your program and implement them if you want to use them.

Application Class	Methods Contained in Application Class	Comments
INotificationHandler	OnNotify OnError	This interface is the equivalent of the Subscription event in releases prior to PeopleTools 8.48.
IReceiver	OnAckReceive OnError	This interface is the equivalent of the OnAckReceive Message event in releases prior to PeopleTools 8.48.
IRequestHandler	OnRequest OnError	This interface is the equivalent of the OnRequest Message event in releases prior to PeopleTools 8.48.

 Copyright © 1988-2007, Oracle. All rights reserved.

Application Class	Methods Contained in Application Class	Comments
IRouter	OnRouteSend OnRouteReceive OnError	This interface is the equivalent of the OnRouteSend and OnRouteReceive Message events in releases prior to PeopleTools 8.48.
ISend	OnRequestSend OnError	This interface is the equivalent of the OnSend Message event in releases prior to PeopleTools 8.48.

Data Type of Message Objects

Every message class is its own data type, that is, messages are declared as data type Message, IntBroker objects are declared as type IntBroker, and so on.

The following are the data types of the message classes:

- Message
- IntBroker
- IBInfo
- ConnectorInfo

Scope of Message Objects

The message objects can only be instantiated from PeopleCode. These objects can be used anywhere you have PeopleCode, that is, in Component Interface PeopleCode, notification PeopleCode, record field PeopleCode, application engine PeopleCode, and so on.

Message Object Population

After you've declared and instantiated your message object, you want to populate it with data. If your data is coming from the component buffers, populating your message is easy.

A message definition can contain a hierarchy of records. A component buffer contains a hierarchy of records. If you want to copy data from a component buffer rowset to a message, the *structure* of the message and the component must be the same. That is, if you have a record at level two in your message and you want that data, you must have the same level zero and level one records in your message as in your component.

For example, suppose your component had the following structure (that is, that PO_INFO and PO_LINE are at the same level, and PO_DETAIL is the child of PO_INFO):

```
PO_HEADER
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   PO_LINE
   PO_INFO
      PO_DETAIL
```

To include the information in the PO_DETAIL record, you must have *at least* the following record structure in your message:

```
PO_HEADER
   PO_INFO
      PO_DETAIL
```

Any records that are in the page that aren't in the message (and vice-versa) are ignored.

After you get your message object, you can create a rowset from it. This rowset has the same structure as the message. If the message is empty, the rowset has no data. If the message has data, the rowset is automatically populated.

The following example is the simplest way of populating a message with data. This assumes that the structure of the message is the same as the structure of the page.

```
/* this gets all the data in the Component buffer */

&RS = GetLevel0();

/* this instantiates a message object */

&MSG = CreateMessage(OPERATION.MY_MESSAGE);

   /* creates a rowset with the same structure as the message */

&MSG_RS = &MSG.GetRowset();

   /* this copies all the data from the page to the message */

&RS.CopyTo(&MSG_RS);

/* this publishes the message */

%IntBrokerPublish(&MSG_RS);
```

A message rowset is the same as a Component buffer rowset, or any other rowset. It is composed of rows, records, and fields. Suppose you didn't want to get all the data from the Component buffer, but instead wanted to populate just a particular record in your message.

To access a record in a message rowset is the same as accessing a record in a component buffer rowset. You must instantiate the rowset, then specify the row before you can access the record.

The following selects values into a record, then uses the record method CopyFieldTo to copy from the Component buffer record to the message record.

```
Local SQL &LN_SQL;
Local Message &MSG;
Local Rowset &HDR_RS, &LN_RS;
Local Record &LN_REC, &ln_rec_msg;

&MSG = CreateMessage(OPERATION.STOCK_REQUEST);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&HDR_RS = &MSG.GetRowset();
&LN_REC = CreateRecord(Record.DEMAND_INF_INV);

&LN_SQL = CreateSQL("Select * from PS_DEMAND_INF_INV where  BUSINESS_UNIT= :1 and⇒
 ORDER_NO = :2", &BUSINESS_UNIT, &ORDER_NO);

&J = 1;
While &LN_SQL.Fetch(&LN_REC)

  /* copy data into the Level 1 of &MSG object */
  &LN_RS = &HDR_RS(&I).GetRowset(1);

  If &J > 1 Then
    &LN_RS.InsertRow(&J - 1);
  End-If;

  &ln_rec_msg = &LN_RS.GetRow(&J).GetRecord(Record.DEMAND_INF_INV);
  &LN_REC.CopyFieldsTo(&ln_rec_msg);
  &J = &J + 1;
End-While;
```

This section also discusses items to keep in mind when:

- Populating a rowset from a message.
- Publishing and subscribing to partial records.
- Subscribing to character fields.

Considerations When Populating a Rowset From a Message

Suppose your message had two rowsets, one at level zero, a second at level one. In the message, only the level zero rowset contains any data. When you use GetRowset to create a rowset for the entire message, the rowset at level one will contain an empty row, even if there isn't any data in it. (This is standard behavior for *all* rowsets.) However, you can use the IsChanged property on the record object to determine the status of the data.

The following notification PeopleCode example traverse the rowset. (Notice the use of ChildCount, ActiveRowCount, and IsChanged properties).

'. . .' is where application specific code would go.

```
&MSG_ROWSET = &MSG.GetRowset();
For &A0 = 1 To &MSG_ROWSET.ActiveRowCount

/*************************/
/* Process Level 1 Records */
/*-------------------------*/

  If &MSG_ROWSET(&A0).ChildCount > 0 Then
  For &B1 = 1 To &MSG_ROWSET(&A0).ChildCount

    &LEVEL1_ROWSET = &MSG_ROWSET(&A0).GetRowset(&B1);
      For &A1 = 1 To &LEVEL1_ROWSET.ActiveRowCount
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
        If &LEVEL1_ROWSET(&A1).GetRecord(1).IsChanged Then
          . . .

/**************************/
/* Process Level 2 Records */
/*-------------------------*/

          If &LEVEL1_ROWSET(&A1).ChildCount > 0 Then
          For &B2 = 1 To &LEVEL1_ROWSET(&A1).ChildCount
            &LEVEL2_ROWSET = &LEVEL1_ROWSET(&A1).GetRowset(&B2);
              For &A2 = 1 To &LEVEL2_ROWSET.ActiveRowCount
                If &LEVEL2_ROWSET(&A2).GetRecord(1).IsChanged Then
                .  .  .

/**************************/
/* Process Level 3 Records */
/*-------------------------*/

                If &LEVEL2_ROWSET(&A2).ChildCount > 0 Then
                  For &B3 = 1 To &LEVEL1_ROWSET(&A2).ChildCount
                    &LEVEL3_ROWSET = &LEVEL2_ROWSET(&A2).GetRowset(&B3);
                    For &A3 = 1 To &LEVEL3_ROWSET.ActiveRowCount
                      If &LEVEL3_ROWSET(&A3).GetRecord(1).IsChanged Then
                      .  .  .

                      End-If; /* A3 - IsChanged */
                    End-For; /* A3 - Loop */
                  End-For; /* B3 - Loop */
                End-If; /* A2 - ChildCount > 0 */

/*---------------------------------*/
/* End of Process Level 3 Records */
/*********************************/

              End-If; /* A2 - IsChanged */
            End-For; /* A2 - Loop */
          End-For; /* B2 - Loop */
        End-If; /* A1 - ChildCount > 0 */

/*---------------------------------*/
/* End of Process Level 2 Records */
/*********************************/

      End-If; /* A1 - IsChanged */
    End-For; /* A1 - Loop */
   End-For; /* B1 - Loop */
  End-If; /* A0 - ChildCount > 0 */

/*---------------------------------*/
/* End of Process Level 1 Records */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/*********************************/

End-For; /* A0 - Loop */
```

Considerations for Publishing and Subscribing to Partial Records

If you've selected to not publish all the fields in the message, you must be careful when inserting that data into a record.

Deselecting the Include check box in the message definition means that the field is *excluded* from the message definition.

- The field won't be included when generating an XML document (when publishing a message.)
- If the field is present in the XML (that is, it wasn't excluded from the published message) it is ignored by the subscribing system.

When you insert the data from the message into the database, you *must* set the values for the fields that aren't in the message. You can use the SetDefault record class method to do this. You could also use a Component Interface based on the component the message was created from to leverage the component defaults.

See Also

Chapter 31, "Record Class," SetDefault, page 1398

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Creating Component Interface-Based Services"

Considerations When Subscribing to Character Fields

If a message definition has character fields that are defined as uppercase, when the message is subscribed to, character data for those fields is automatically converted to uppercase.

Message Segments

To make processing more efficient, you can divide a large message into pieces using message segments. Generally, you only divide asynchronous messages into segments.

Message nodes can be specified as "segment aware". If a node is not segment aware and you send an asynchronous message that is segmented to it, you received an error message when viewing the error message log in message details on the message monitor. No publication contracts are created. If you send a synchronous message that is segmented to a node that is not segment aware, you receive an error.

There are several methods for creating, updating, and deleting segments. There are also two properties that you need to take into consideration when working with segments.

If you specify the SegmentsByDatabase property as false, you can only have the configured number defined in PSADMIN (Message Segment From DB). If you specify this property as true, you can have as many segments as you need.

Copyright © 1988-2007, Oracle. All rights reserved.

The SegmentsByDatabase property also specifies whether the segments are kept in memory or written to the database when they are received. If you specify true, the segments are automatically written to the database. If you specify false, the segments are held in memory. If you specify true, then cancel out of the program processing the segments, the changes are not committed to the database.

The SegmentUnOrder property is only applicable for asynchronous messages. If you specify the SegmentUnOrder property as true, the receiving node processes the segments in parallel.

Note. You should use DeleteSegment and UpdateSegment only when writing to memory, or when SegmentsByDatabase is set to False. These methods do not work when writing to the database, or when SegmentsByDatabase is set to True.

The following is an example of how to use the segment properties and methods to send a segmented message. Note that there are only two CreateNextSegment calls. By default the first segment is automatically created. The first time you use CreateNextSegment, the message is split into two segments. The next time, you add a third segment. You don't need to call CreateNextSegment to access the third segment, it's automatically generated.

```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE, &RS;
Local boolean &Bo, &Stuff;
Local string &lip;

&nodes = CreateArray("");
&nodes[1] = "QE_YO";
&nodes[2] = "QE_STUFF";

QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(OPERATION.QE_FLIGHTPLAN);

/* the next lines copy the rowset into the first segment */

&MSG.CopyRowset(&FLIGHT_PROFILE);
&MSG.CreateNextSegment();

/* This copies the next portion of the rowset into the next segment */

&MSG.CopyRowset(&FLIGHT_PROFILE);
&MSG.CreateNextSegment();

/* This copies the last portion of the rowset into the third segment */

&MSG.CopyRowset(&FLIGHT_PROFILE);

/* This specifies that the message segments can be processed separately */

&MSG.IBInfo.SegmentsUnOrder = True;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
IntBroker.publish(%MSG);
```

The following is an example of receiving a segmented message. This would be found in a notification PeopleCode program:

```
Local Message &MSG;
Local Rowset &rs, &rs1;
Local Record &FLIGHTDATA, &REC;

Local string &acnumber_value, &msi_sensor_value, &ofp_value, &actype_value,⇒
 &callsign_value, &squadron_value, &comm1_value, &comm2_value, &ecm_value;

Local XmlDoc &xmldoc;
Local string &yo;
Local boolean &bo;

&CRLF = Char(13) | Char(10);

&MSG = %IntBroker.GetMessage();

/* It is very important to set the rowset to null for every iteration */

/* Also, you may want to verify if the segment count is
greater than 10, and if it is, set the SegmentsByDatabase property to True */

For &i = 1 To &MSG.SegmentCount
   &rs = Null;
   &MSG.GetSegment(&i);

   &rs = &MSG.GetRowset();
   &REC = &rs(1).QE_FLIGHTDATA;

   &FLIGHTDATA = CreateRecord(Record.QE_FLIGHTDATA);
   &REC.CopyFieldsTo(&FLIGHTDATA);

   /* Parse out Message Data */
   &acnumber_value = &FLIGHTDATA.QE_ACNUMBER.Value;
   &msi_sensor_value = &FLIGHTDATA.QE_MSI_SENSOR.Value;
   &ofp_value = &FLIGHTDATA.QE_OFP.Value;
   &actype_value = &FLIGHTDATA.QE_ACTYPE.Value;
   &callsign_value = &FLIGHTDATA.QE_CALLSIGN.Value;
   &squadron_value = &FLIGHTDATA.QE_SQUADRON.Value;
   &comm1_value = &FLIGHTDATA.QE_COMM1.Value;
   &comm2_value = &FLIGHTDATA.QE_COMM2.Value;
   &ecm_value = &FLIGHTDATA.QE_ECM.Value;

   &outstring = "Send Async FLight test";

   /* Construct Output String */
   &outstring = &outstring | &acnumber_value | &CRLF | &msi_sensor_value | &CRLF |⇒
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
 &ofp_value | &CRLF | &actype_value | &CRLF | &callsign_value | &CRLF | &squadron_⇒
value | &CRLF | &comm1_value | &CRLF | &comm2_value | &CRLF | &ecm_value;

   /* Log Output String into page record */
   &FLIGHTDATA.GetField(Field.DESCRLONG).Value = &outstring;

   SQLExec("DELETE FROM PS_QE_FLIGHTDATA");
   &FLIGHTDATA.Insert();

End-For;
```

See Also

Chapter 22, “Message Classes,” CreateNextSegment, page 747

Chapter 22, “Message Classes,” SegmentsByDatabase, page 780

Chapter 22, “Message Classes,” SegmentsUnOrder, page 817

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Adding and Configuring Nodes”

Error Handling

In your notification PeopleCode, you may want to validate the information received in the message. If you find that the data isn’t valid, use Exit(1) to end your PeopleCode program. This sets the status of the message to ERROR, logs any error messages to the Application Message Queues, and automatically rolls back any database changes you may have made.

There are many ways to validate the data and capture the error messages:

- Use ExecuteEdits on the message object
- Use ExecuteEdits on the record object
- Use a Component Interface

Write your own validation PeopleCode in the notification program:

The following example validates the Business Unit of a message against an array of valid BU’s:

```
For &I = 1 To &ROWSET.RowCount;

  &POSITION = &BUArray.Find(&ROWSET(&I).GetRecord(1).BUSINESS_UNIT.Value);
  If &POSITION = 0 Then
    &Status = "ERROR: BU not Found or not Active";
    &ROWSET(&I).BCT_ADJS_MSG_VW.BUSINESS_UNIT.IsEditError = True;
    &ROWSET(&I).BCT_ADJS_MSG_VW.BUSINESS_UNIT.MessageSetNumber = 11100;
    &ROWSET(&I).BCT_ADJS_MSG_VW.BUSINESS_UNIT.MessageNumber = 1230;
/* The error message 11100-1230 reads: This Business Unit is */
/* not a valid, open, Inventory Business Unit */

  End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
End-For;
```

In the calling PeopleCode, the program calls Exit(1) based on the value of &Status.

Note. All errors for notification PeopleCode get logged to the application message error table, *not* to the PSMessages collection (on the session object.)

See Also

Chapter 22, "Message Classes," ExecuteEdits, page 748

Chapter 31, "Record Class," ExecuteEdits, page 1386

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

Message Class Built-In Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," AddSystemPauseTimes

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMessage

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," DeleteSystemPauseTimes

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," PingNode

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReValidateNRXmlDoc

Message Class Methods

In this section, we discuss the Message class methods. The methods are discussed in alphabetical order.

Clone

Syntax

```
Clone()
```

Description

The Clone method creates an identical copy of the message, copying the data tree of the message executing the method. The Clone function sets the following properties for the resulting message object, based on the values set for the message definition created in Pure PeopleSoft Internet Architecture:

- Name

Copyright © 1988-2007, Oracle. All rights reserved.

- QueueName
- IsOperationActive

Other properties are set when the message is published or subscribed to (TransactionID, PubNodeName, and so on), or are dynamically generated at other times (Size, IsEditError, and so on.)

Clone creates a unique version of the message. If the original message changes, it is not reflected in the cloned object.

Parameters

None.

Returns

A message object.

Example

Clone could be used in a Hub and Spoke messaging environment. The Hub node uses this method during notification processing to publish a copy of the messages it receives from the Spokes.

```
&Msg = %IntBroker.GetMessage();
&Rowset = &Msg.GetRowset();
&Clone = &Msg.Clone();

%IntBroker.Publish(&Clone);
```

The hub's publication routing rules are then used to determine which spokes receive the message.

See Also

Chapter 22, "Message Classes," CopyRowset, page 741

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMessage

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

CopyPartRowset

Syntax

CopyPartRowset(*PartIndex, &Rowset*)

Description

Use the CopyPartRowset to copy the rowset specified by *&Rowset* to the part of the message specified by *PartIndex*.

Note. This method only works with rowset-based container messages.

The primary record of the level zero rowset for both the specified message part and the specified rowset must be the same.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PartIndex	Specify the number of the part you want to copy the rowset data to.
&Rowset	Specify an already instantiated rowset object that contains the data that you want to copy to the message part.

Returns

None.

See Also

Chapter 22, "Message Classes," GetPartRowset, page 753 and Chapter 22, "Message Classes," GetPartXMLDoc, page 754.

CopyRowset

Syntax

```
CopyRowset(source_rowset [, record_list]);
```

Where *record_list* is a list of record names in the form:

```
RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]. . .
```

Description

The CopyRowset method copies all data from the source rowset to the like-named fields in the message object executing the method. This is an easy way to populate a message with data from a component.

Note. CopyRowset copies all the data, including rows that haven't been modified. If you want to copy only data that has changed in some way, use the CopyRowsetDelta method.

See Chapter 22, "Message Classes," CopyRowsetDelta, page 742.

When the record names in the message and component do not match exactly, use the optional *record_list* to specify the records to be copied from the component into the message.

When you use the CopyRowset method to copy the contents from the source rowset to the message object, you are creating a *unique* copy of the object. If you change the original rowset, the message object is not changed.

Note. You can execute CopyRowset against a message only once. After a message is populated, any other CopyRowsets are ignored. If you have to populate different levels of a message rowset separately, you can use the CreateRowset method to create a rowset that has the same structure as your message, populate the created rowset, then use CopyRowset to populate your message. You can also use the Rowset CopyTo method to populate a message rowset, then populate the PSCAMA record by hand.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
source_rowset	Specifies the name of the rowset to be copied into the message object.
record_list	Specifies specific source and target records to be copied into the message object.

Returns

None.

Example

The following example copies an entire component into a message object.

```
&Msg = %IntBroker.GetMessage();
&Rowset = &Msg.GetRowset()
&Component_Rowset = GetLevel0();
&Msg.CopyRowset(&Component_Rowset);
```

The following example copies a header/line page rowset into a header/line message object, using the *record_list* because the record names don't match:

```
&Msg = %IntBroker.GetMessage();
&Rowset= &Msg.GetRowset();
&Component_Rowset = GetLevel0();
&Msg.CopyRowset(&Component_Rowset, RECORD.PO_HDR_VW, RECORD.PO_HDR, RECORD.PO_LINE_⇒
VW, RECORD.PO_LINE);
```

See Also

Chapter 22, "Message Classes," CopyRowsetDelta, page 742

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

Chapter 33, "Rowset Class," page 1421

Chapter 33, "Rowset Class," Fill, page 1428

CopyRowsetDelta

Syntax

CopyRowsetDelta(*source_rowset* [, *record_list*]);

Where *record_list* is a list of record names in the form:

[RECORD.*source_recname1*, RECORD.*target_recname1*
[, RECORD.*source_recname2*, RECORD.*target_recname2*]]. . .

Description

The CopyRowsetDelta method copies rows of data that have changed from the source rowset to the like-named records and like-named fields in the message object executing the method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. CopyRowsetDelta copies all the like-named fields from the changed *row*into the message. It is *not* copying just the changed like-named fields.

When the record names in the message and component do not match exactly, use the optional *record_list* to specify the records to be copied from the component into the message. The specified target records must be records in your message, while the specified source records must be records in a rowset that exists in the data buffer and is populated.

This is an easy way to populate a message when the records in the message match the records in the component that the message is published from.

In addition, the CopyRowsetDelta method sets the AUDIT_ACTN field in the PSCAMA table for every row in the message. The notification process can then use PSCAMA.AUDIT_ACTN to determine how to process every row that was published.

The set values match those used in audit trail processing, that is:

- A - Row inserted.
- D - Row deleted.
- C - Row changed (updated), but no key fields changed. The system copies the new value to the Message rowset.
- K - Row changed (updated), and at least one key field changed. The system copies the old value to the Message rowset and the new value (see "N").
- N - Row changed (updated), and at least one key field changed. The system copies the old value to the Message rowset and the new value (see "K").
- "" − blank value means that nothing on that row has changed. This is the default value, and is the value used to tag the parent rows of children that have changed.

Note. If a child row is inserted (or changed or deleted) CopyRowsetDelta also copies the parent row (and the parent row of that row, and so on, up to the top row) so the subscriber has a full picture of the transaction. A blank value is set in the AUDIT_ACTN field for these rows to let the subscriber know they don’t have to take action; the parent rows are there for reference only.

The Audit_Action values "A", "C", "D" are set when a record is added, changed, or deleted, respectively. In some cases such as effective-dated records, the user may change a key field value, such as Effective Date. In response to such an user action, two records are created, one with an Audit_Action value of "N", and the other with Audit_Action value "K". The "N" record has all the new values, while the "K" record retains the old values.

When you use the CopyRowsetDelta method to copy the contents from the source rowset to the message object, you are creating a *unique* copy of the object. If you change the original rowset, the message object is not be changed.

Note. You can execute CopyRowsetDelta against a message only once. After a message is populated, any other CopyRowsetDeltas are ignored. If you have to populate different levels of a message rowset separately, you can use the CreateRowset method to create a rowset that has the same structure as your message, populate the created rowset, then use CopyRowsetDelta to populate your message. You can also use the Rowset CopyTo method to populate a message rowset, then populate the PSCAMA record by hand.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
source_rowset	Specifies the name of the rowset to be copied into the message object.
record_list	Specifies source and target records to be copied into the message object. The target records must be records in your message, while the source records must be records in a rowset that exists in the data buffer and is populated.

Returns

None.

Example

The following example copies all the changed rows of data from a component into a message object.

```
&Component_Rowset = GetLevel0();
&Msg.CopyRowsetDelta(&Component_Rowset);
```

See Also

Chapter 22, "Message Classes," CopyRowset, page 741

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Understanding Supported Message Structures," PSCAMA

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Chapter 33, "Rowset Class," page 1421

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

CopyRowsetDeltaOriginal

Syntax

CopyRowsetDeltaOriginal(*source_rowset* [, *record_list*]);

Where *record_list* is a list of record names in the form:

```
[RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]]. . .
```

Description

The CopyRowsetDeltaOriginal method copies rows of data that have changed from the source rowset to the like-named records and like-named fields in the message. It also copies the original value of the changed rows.

Note. CopyRowsetDeltaOriginal copies all the like-named fields from the changed and original *rows* into the message. It is not copying just the changed like-named fields.

 Copyright © 1988-2007, Oracle. All rights reserved.

When the record names in the message and component do not match exactly, use the optional *record_list* to specify the records to be copied from the component into the message. The specified target records must be records in your message, while the specified source records must be records in a rowset that exists in the data buffer and is populated.

The CopyRowsetDeltaOriginal method sets the AUDIT_ACTN field in the PSCAMA table for every row in the message. The notification process can then use PSCAMA.AUDIT_ACTN to determine how to process every row that was published.

The set values match those used in audit trail processing, that is:

- A - Row inserted.
- D - Row deleted.
- C - Row changed (updated), but no key fields changed. The system copies the new value to the Message rowset and the original value (see "O" below).
- O - Prior value of changed row. The system copies the new value to the Message rowset and the original value (see "C").
- K - Row changed (updated), and at least one key field changed. The system copies the original value to the Message rowset and the new value (see "N").
- N - Row changed (updated), and at least one key field changed. The system copies the original value to the Message rowset and the new value (see "K").
- "" − blank value means that nothing on that row has changed. This is the default value, and is the value used to tag the parent rows of children that have changed.

Note. If a child row is inserted (or changed or deleted) CopyRowsetDeltaOriginal also copies the parent row (and the parent row of that row, and so on, up to the top row) so the subscriber has a full picture of the transaction. A blank value is set in the AUDIT_ACTN field for these rows to let the subscriber know they don't have to take action; the parent rows are there for reference only.

The Audit_Action values "A", "C", "D" are set when a record is added, changed or deleted, respectively. When a row is changed, two records are created, one with an Audit_Action of "C", the other with Audit_Action value of "O". The "C" record has all the new values, while the "O" record retains the original values. In some cases such as effective-dated records, a user may change a key field value, such as Effective Date. In response to such an user action, two records are created, one with an Audit_Action value of "N", and the other with Audit_Action value "K". The "N" record has all the new values, while the "K" record retains the original values. The "N"/"K" combination is also created if both a key change and a non-key change is made on the same row of data.

The following table details the output from CopyRowsetDeltaOriginal:

User Action	CopyRowsetDeltaOriginal Output	Comments
Change a key.	"N" and "K" rows created.	The "K" row contains the original Key value, the "N" row, the new.
Change a Key + change a Non-Key.	"N" and "K" rows created only (that is: no "C"/"O" rows in that case).	The Key and Non-Key Original Values are both contained in the "K" row.

Copyright © 1988-2007, Oracle. All rights reserved.

User Action	CopyRowsetDeltaOriginal Output	Comments
Change a Non-Key.	"C" and "O" rows created.	Original Value stored in "O" row.
Add non-effective dated row.	"A" row created.	No additional rows created.
Add effective-dated row and only 1 original effective-dated row exists.	"A" and "O" rows created.	"O" row contains the original effective dated row.
Add effective-dated row and more than 1 original effected dated rows exist.	"A" and "O" rows created.	"O" row contains data from row that cursor was on when the new row was added. For example: If a rowset contains two rows (01/01/1980 and 02/02/2000), the current row is the 01/01/1980 row and the user adds a new row with today's date. The original values contain the data from the 01/01/1980. Likewise, if a rowset contains two rows (01/01/1980 and 02/02/2000), the current row is the 02/02/2000 row and the user adds a new row with today's date. Then the original values contain the data from the 02/02/2000.
Delete a row	"D" row created.	No additional rows created.

When you use the CopyRowsetDeltaOriginal method to copy the contents from the source rowset to the message object, you are creating a unique copy of the object. If you change the original rowset, the message object is not changed.

Note. You can execute CopyRowsetDeltaOriginal against a message only once. After a message is populated, any other CopyRowsetDeltaOriginal PeopleCode statements are ignored. If you have to populate different levels of a message rowset separately, you can use the CreateRowset method to create a rowset that has the same structure as your message, populate the created rowset, then use CopyRowsetDeltaOriginal to populate your message. You can also use the Rowset CopyTo method to populate a message rowset, then populate the PSCAMA record manually.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
source_rowset	Specifies the name of the rowset to be copied into the message object.
record_list	Specifies source and target records to be copied into the message object. The target records must be records in your message, while the source records must be records in a rowset that exists in the data buffer and is populated.

Returns

None.

Example

```
Function Update_Dir(&MSGName As string)

  &MSGName = "OPERATION." | &MSGName;
  &MSG = CreateMessage(@(&MSGName));

  &PnlRowset = GetLevel0();
  &MSG.CopyRowsetDeltaOriginal(&PnlRowset);
  %IntBroker.Publish(&MSG);

End-Function;
```

See Also

Chapter 22, "Message Classes," CopyRowset, page 741 and Chapter 22, "Message Classes," CopyRowsetDelta, page 742.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Understanding Supported Message Structures," PSCAMA

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

Chapter 33, "Rowset Class," page 1421

CreateNextSegment

Syntax

CreateNextSegment()

Description

Use the CreateNextSegment method to divide a message into segments. You generally only divide asynchronous messages that contain a large amount of data. This method is used only when creating a message for publication, not after a message has been published.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

See Chapter 22, “Message Classes,” Message Segments, page 735.

See Also

Chapter 22, “Message Classes,” CreateNextSegment, page 747 , Chapter 22, “Message Classes,” DeleteSegment, page 748 , Chapter 22, “Message Classes,” SegmentRestart, page 762 , Chapter 22, “Message Classes,” UpdateSegment, page 768 , Chapter 22, “Message Classes,” CurrentSegment, page 771 and Chapter 22, “Message Classes,” SegmentCount, page 780.

DeleteSegment

Syntax

```
DeleteSegment(SegmentNumber)
```

Description

Use the DeleteSegment method to delete the specified segment. This method is used only when creating a message for publication, not after a message has been published.

Note. You should use DeleteSegment and UpdateSegment only when writing to memory, or when SegmentsByDatabase is set to False. These methods do not work when writing to the database, or when SegmentsByDatabase is set to True.

Parameters

Parameter	Description
SegmentNumber	Specify a number indicating which segment you want to delete from the unpublished message.

Returns

None.

See Also

Chapter 22, “Message Classes,” CreateNextSegment, page 747 , Chapter 22, “Message Classes,” GetSegment, page 757 , Chapter 22, “Message Classes,” UpdateSegment, page 768 , Chapter 22, “Message Classes,” CurrentSegment, page 771 and Chapter 22, “Message Classes,” SegmentCount, page 780.

ExecuteEdits

Syntax

```
ExecuteEdits([editlevels]);
```

where *editlevels* is a list of values in the form:

```
editlevel1 [+ editlevel2] . . .]);
```

and where *editleveln* is one of the following system constants:

 Copyright © 1988-2007, Oracle. All rights reserved.

%Edit_DateRange

%Edit_OneZero

%Edit_PromptTable

%Edit_Required

%Edit_TranslateTable

%Edit_YesNo

Description

The ExecuteEdits method executes the standard system edits on every field for every record in the message executing the method. All edits are based on the record edits defined for the record that the message is based on. The types of edits performed depends on the *editlevel*. If no *editlevel* is specified, all system edits defined for the record definition are executed, that is:

- Reasonable Date Range (Is the date contained within plus or minus 30 days from the current date?)
- 1/0 (Do all 1/0 fields only contain 1 or 0?)
- Prompt Table (Is field data contained in the specified prompt table?)
- Required Field (Do all required fields contain data? For numeric or signed fields, it checks that they do not contain NULL or 0 values.)
- Translate Table (Is field data contained in the translate table?)
- Yes/No (Do all yes/no fields only contain only yes or no data?)

If any of the edits fail, the status of the property IsEditError is set to True for the Message Object executing the method, and any Rowset, Row, Record, or Field Objects that could be instantiated from the same data as the Message Object. In addition, the Field class properties MessageNumber and MessageSetNumber are set to the number of the returned message and message set number of the returned message, for the field generating the error.

If you want to check the field values only for a particular record, you can use the ExecuteEdits method with a record object.

In addition, if you want to dynamically set the prompt tables used for a field (with the %EditTable function) you must use the SetEditTable method.

Considerations for ExecuteEdits and SetEditTable

If an effective date is a key on the prompt table, and the record being edited doesn't contain an effective-dated field, the current datetime is used as the key value.

If a setID is a key on the prompt table, and the record being edited doesn't contain a setID field, the system looks for the setID on the other records in the current row first, then search parent rows.

Considerations for ExecuteEdits and Numeric Fields

A 0 (zero) may or may not be a valid value for a numeric field. ExecuteEdits processes numeric fields in different ways, depending on whether the field is required and a prompt table is used:

- If the numeric field is required and there is no prompt edit: 0 is considered valid, so numeric fields never fail the required test.
- If the numeric field is required and there is a prompt edit: Prompt edit is run and the value must be valid.

Copyright © 1988-2007, Oracle. All rights reserved.

- If the numeric field is not required and there is a prompt edit: 0 is considered valid, so the prompt edit is skipped.

Parameters

Parameter	Description
editlevel	Specifies the standard system edits to be performed against every field on every record. If *editlevel* isn't specified, all system edits are performed. *editlevel* can be any of the following system constants: • %Edit_DateRange • %Edit_OneZero • %Edit_PromptTable • %Edit_Required • %Edit_TranslateTable • %Edit_YesNo

Returns

None.

Example

The following is an example of a call to execute Required Field and Prompt Table edits:

```
&MSG.ExecuteEdits(%Edit_Required + %Edit_PromptTable);
```

The following is an example showing how ExecuteEdits() could be used:

```
&MSG.ExecuteEdits();
If &MSG.IsEditError Then
  Exit(1);
End-If;
```

See Also

Chapter 22, "Message Classes," SetEditTable, page 762

Chapter 15, "Field Class," EditError, page 504

Chapter 15, "Field Class," MessageNumber, page 512

Chapter 15, "Field Class," MessageSetNumber, page 513

Chapter 31, "Record Class," IsEditError, page 1403

Chapter 31, "Record Class," ExecuteEdits, page 1386

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

 Copyright © 1988-2007, Oracle. All rights reserved.

GenXMLPartString

Syntax

GenXMLPartString(*PartIndex*)

Description

Use the GenXMLPartString to create an XML string based on the message part specified by *PartIndex*.

Note. This method only works with rowset-based container messages.

Parameters

Parameter	Description
PartIndex	Specify the number of the part that you want to generate an XML string for.

Returns

An XML string.

See Also

Chapter 22, "Message Classes," GenXMLString, page 751 , Chapter 22, "Message Classes," GetPartRowset, page 753 and Chapter 22, "Message Classes," LoadXMLPartString, page 758.

GenXMLString

Syntax

GenXMLString()

Description

The GenXMLString method creates an XML string based on the message after it's loaded from the message buffer.

Note. This method does not support nonrowset-based messages.

Parameters

None.

Returns

An XML string.

Example

In the following example, the first thee lines of code creates the message object and copies the data from a Component buffer into the message object. The last line creates an XML string based on that message object.

```
&MSG = CreateMessage(OPERATION.PO_INSERT);
&RS = GetLevel0();
&MSG.CopyRowset(&RS);
&XML_STRING = &MSG.GenXMLString();
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 22, “Message Classes,” LoadXMLString, page 759

GetContentString

Syntax
GetContentString([*SegmentIndex*])

Description

Use the GetContentString to return a string populated with the content from the specified message segment. If you don’t specify a segment, the data from the first segment is used.

Parameters

Parameter	Description
SegmentIndex	Specify the message segment that you want to access.

Returns

A string populated with the content of the specified segment if successful, null otherwise.

See Also

Chapter 22, “Message Classes,” CreateNextSegment, page 747

GetPartAliasName

Syntax
GetPartAliasName(*PartIndex*)

Description

Use the GetPartAliasName to access the alias of the specified message part.

Note. This method works only with container messages.

Parameters

Parameter	Description
PartIndex	Specify the message part that you want to access the alias.

Returns

A string containing the message part alias.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, "Message Classes," AliasName, page 769

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Specifying Record Aliases

GetPartName

Syntax

```
GetPartName(PartIndex)
```

Description

Use the GetPartName method to return the message name of the message specified by *PartIndex*.

Note. This method works only with container messages.

Parameters

Parameter	Description
PartIndex	Specify the number of the message part that you want to get the name for.

Returns

A string.

See Also

Chapter 22, "Message Classes," CopyPartRowset, page 740 and Chapter 22, "Message Classes," GenXMLPartString, page 751.

GetPartRowset

Syntax

```
GetPartRowset(PartIndex)
```

Description

Use the GetPartRowset to instantiate and populate a rowset object based on the specified message part in a container message.

Note. This method only works with rowset-based container messages.

Parameters

Parameter	Description
PartIndex	Specify the number of the message part that you want to instantiate and populate a rowset from.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a rowset object if successful, Null otherwise.

See Also

Chapter 22, "Message Classes," CopyPartRowset, page 740 and Chapter 22, "Message Classes," GenXMLPartString, page 751.

GetPartVersion

Syntax

GetPartVersion(*PartIndex*)

Description

Use the GetPartVersion method to return the version number of the message part specified by *PartIndex*.

Note. This method only works with container messages.

Parameters

Parameter	Description
PartIndex	Specify the number of the message part that you want to access the version of.

Returns

A string containing the version number.

See Also

Chapter 22, "Message Classes," PartCount, page 779

GetPartXMLDoc

Syntax

GetPartXMLDoc(*PartIndex*)

Description

Use the GetPartXMLDoc to retrieve the message part data as an XmlDoc object.

Note. This method can only be used for a nonrowset-based message parts.

Parameters

Parameter	Description
PartIndex	Specify the message part that you want to access the data for as an XmlDoc object.

Returns

An XmlDoc object populated with the message part data.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, “Message Classes,” GetXmlDoc, page 757 and Chapter 22, “Message Classes,” GetPartRowset, page 753.

GetQueryString

Syntax

GetQueryString(*Parameter_Name*)

Description

Use the GetQueryString method to obtain the parameter values of a query string.

Note. This method has been deprecated and remains for backward compatibility only. Use the ConnectorInfo class GetQueryStringArg method instead.

See Also

Chapter 22, “Message Classes,” GetQueryStringArgName, page 824

Parameters

Parameter	Description
Parameter_Name	Specify the parameter name you want to obtain.

Returns

A string containing the value of the parameter.

Example

```
Local Message &request;
Local Rowset &Rowset;
Local String &emplid;

&request = %IntBroker.GetMessage();
&Rowset = &request.GetRowset();
&emplid = &request.GetQueryString("EMPLID");
```

See Also

Chapter 22, “Message Classes,” GetQueryStringArgName, page 824, Chapter 22, “Message Classes,” GetQueryStringArgValue, page 825 and Chapter 22, “Message Classes,” SetQueryString, page 763.

GetRowset

Syntax

GetRowset([*version*])

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The GetRowset method returns a standard PeopleCode level zero rowset object for the specified message version. It creates an empty rowset for the specified message version if it doesn't already exist. If no message version is specified, the default message version is used.

When you use the GetRowset method, you are not creating a unique copy of the object. You are making only a copy of the reference. If you change the rowset, the original message is changed.

Parameters

Parameter	Description
version	An optional parameter, specifying an existing message version as a string. If *version* isn't specified, the default message version is used.

Returns

A Rowset object if successful.

Example

The following gets all the SET_CNTRL_REC rows related to the row on the page, then updates the field SETID with the value from the page. GetRowset is used to create a rowset based on the message structure, that is, an unpopulated rowset. Because the rowset is unpopulated, you can use the Fill method to populate with data from the page.

```
Local Message &MSG;
Local Rowset &MSG_ROWSET;

If FieldChanged(SETID) Then
  &MSG = CreateMessage(OPERATION.SET_CNTRL_REC);
  &MSG_ROWSET = &MSG.GetRowset();
  &MSG_ROWSET.Fill("where SETCNTRLVALUE =:1 and REC_GROUP_ID =:2", SETCNTRLVALUE,⇒
 REC_GROUP_ID);

  For &I = 1 To &MSG_ROWSET.ActiveRowCount
    &MSG_ROWSET.GetRow(&I).SET_CNTRL_REC.SETID.Value = SETID;
    &MSG_ROWSET.GetRow(&I).PSCAMA.AUDIT_ACTN.Value = "C";
  End-For;

  %IntBroker.Publish(&MSG);

End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, "Message Classes," Clone, page 739

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateMessage

Chapter 33, "Rowset Class," page 1421

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Objects and Classes in PeopleCode," Assigning Objects

GetSegment

Syntax

GetSegment(*SegmentNumber*)

Description

Use the GetSegment method to return the specified segment. This method doesn't actually return anything, but populates the current message object with the specified segment's data.

Parameters

Parameter	Description
SegmentNumber	Specify the number of the segment you want to access.

Returns

None.

See Also

Chapter 22, "Message Classes," CreateNextSegment, page 747 , Chapter 22, "Message Classes," DeleteSegment, page 748 , Chapter 22, "Message Classes," UpdateSegment, page 768 and Chapter 22, "Message Classes," SegmentCount, page 780.

GetXmlDoc

Syntax

GetXmlDoc()

Description

Use the GetXmlDoc method to retrieve the message data as an XmlDoc object.

Note. This method can only be used for a nonrowset-based message. If you use this method with a rowset-based message an error message is returned.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A XmlDoc object.

See Also

Chapter 22, "Message Classes," SetXmlDoc, page 765

Chapter 42, "XmlDoc Class," page 1745

InBoundPublish

Syntax

InBoundPublish(*PubNodeName*)

Description

Use the InBoundPublish method to send an asynchronous message based on a message rowset to simulate an inbound asyncronous request from an external node.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class InBoundPublish method instead.

Though you are sending a message to yourself, it goes through all the inbound message processing on *PubNodeName*.

See Also

Chapter 22, "Message Classes," InBoundPublish, page 793

Parameters

Parameter	Description
PubNodeName	Specify the name of the node for which you want to test inbound message publishing.

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," InboundPublishXmlDoc

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Applying Filtering, Transformation and Translation"

LoadXMLPartString

Syntax

LoadXMLPartString(*PartIndex, XmlString*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the LoadXMLPartString method to populate the part of the container message specified by *PartIndex* with the XML string *XmlString*.

Note. This method works only with container messages.

Parameters

Parameter	Description
PartIndex	Specify the number of the part of the container message that you want to populate with XML data.
XmlString	Specify the XML string to be loaded into the message part.

Returns

None.

See Also

Chapter 22, "Message Classes," GenXMLPartString, page 751 and Chapter 22, "Message Classes," GenXMLString, page 751.

LoadXMLString

Syntax

LoadXMLString(*XMLstring*)

Description

The LoadXMLString method loads the message object executing the method with the XML string *XMLstring*.

Parameters

Parameter	Description
XMLString	Specify the XML string to be loaded into the message object.

Returns

None.

Example

```
&MSG = CreateMessage(OPERATION.PO_HEADER);
&MSG.LoadXMLString(&XML_STRING);
```

See Also

Chapter 22, "Message Classes," GenXMLString, page 751

Copyright © 1988-2007, Oracle. All rights reserved.

Publish

Syntax

```
Publish([NodeName] [, Deferred_Mode] )
```

Description

The Publish method publishes a message to the application message queue for the default message version.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class Publish method instead.

This method does not publish the message if the IsActive property is False.

This method publishes a message asynchronously, that is, a reply message is not automatically generated. To publish a message synchronously, use the SyncRequest method.

Note. This method supports nonrowset-based messages only if the data is populated using the SetXmlDoc method.

If the Publish method is called and the message isn't populated (either using SetXmlDoc or one of the rowset methods,) an empty message is published.

The Publish method can be called from any PeopleCode event, but is generally called from an Application Engine PeopleCode step or from a component.

When publishing from a component, you should publish messages only from the SavePostChange event, either from record field or component PeopleCode. This way, if there's an error in your publish code, the data isn't committed to the database. Also, since SavePostChange is the last Save event fired, you avoid locking the database while the other save processing is happening.

However, until the message is published, the tables involved with the component are locked and are not saved. To avoid problems with this, specify the *Deferred_Mode* property as true, so the message is published after the table data has been committed to the database.

Generally, if the message is successfully published, the PubID, and PubNodeName properties are set to the publication ID and publishing system node name, respectively. The only exception is when the Publish is performed as part of your notification PeopleCode. The notification process is always executed in deferred mode, due to performance considerations, and so the PubID is not populated.

Note. If you're publishing a message from within an Application Engine program, the message won't actually be published until the calling program performs a Commit.

See Also

Chapter 22, "Message Classes," Publish, page 795

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
NodeName	Specify the node to which the message should be published.
Deferred_Mode	Specify whether the message should be published immediately or at the end of the transaction. This parameter takes a Boolean value: False, publish the message immediately, or True, defer the publication. The default value is False.

Returns

None.

Example

The following copies all the data from a page into a message and publishes it.

```
Local Message &MSG;
Local Rowset &ROWSET;

&MSG = CreateMessage(OPERATION.MESSAGE_PO):
&ROWSET = GetLevel0();
&MSG.CopyRowset(&ROWSET);
&MSG.Publish;
```

The following is an example of putting the Incremental Publish PeopleCode on a Record at Level 1 (or 2, 3). If you just do the normal publishing PeopleCode, you get as many messages as there are records at that level (because the code fires for each record).

To make sure the code fires only once, use the following code.

```
/*NAMES.EMPLID.WORKFLOW */
Local Message &MSG;
Local Rowset &RS;

&LEVEL = CurrentLevelNumber();
&CURRENT_ROW = CurrentRowNumber(&LEVEL);
&TOT_ROW = ActiveRowCount(EMPLID);
/* Only Publish the message once for all records on */
/* this level */
If &CURRENT_ROW = &TOT_ROW Then
  &MSG = CreateMessage(OPERATION.NAMES);
  &RS = GetRowset();
  &MSG.CopyRowsetDelta(&RS);
  &MSG.Publish();
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, "Message Classes," IsActive, page 773 and Chapter 22, "Message Classes," InBoundPublish, page 758.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," InboundPublishXmlDoc

SegmentRestart

Syntax

```
SegmentRestart(TransactionID, Segment_index, segmentByDB)
```

Description

Use the SegmentRestart method to restart a message segment. This is used only in a PeopleSoft Application Engine program, and only if check point logic is used.

Parameters

Parameter	Description
Transaction_ID	Specify the transaction ID of the message you want to restart.
Segment_Index	Specify the number of the segment that you want to restart.
SegmentByDB	Specify the segments by database.

Returns

None.

See Also

Chapter 22, "Message Classes," DeleteOrphanedSegments, page 787

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Understanding Message Segments

SetEditTable

Syntax

```
SetEditTable(%PromptField, RECORD.recordname)
```

Description

The SetEditTable method works with the ExecuteEdits method. It is used to set the value of a field on a record that has its prompt table defined as *%PromptField* value. *%PromptField* values are used to dynamically change the prompt record for a field.

There are several steps to setting up a field so that you can dynamically change its prompt table.

 Copyright © 1988-2007, Oracle. All rights reserved.

To set up a field with a dynamic prompt table:

1. Define a field in the DERIVED record called *fieldname*.
2. In the record definition for the field that you want to have a dynamic prompt table, define the prompt table for the field as *%PromptField*, where *PromptField* is the name of the field that you created in the DERIVED record.
3. Use SetEditTable to dynamically set the prompt table.

%PromptField is the name of the field on the DERIVED work record. **RECORD**.*recordname* is the name of the record to be used as the prompt table.

```
&MSG.SetEditTable("%EDITTABLE1", Record.DEPARTMENT);
&MSG.SetEditTable("%EDITTABLE2", Record.JOB_ID);
&MSG.SetEditTable("%EDITTABLE3", Record.EMPL_DATA);
&MSG.ExecuteEdits();
```

Every field on a record that has the prompt table field %EDITTABLE1 will have the same prompt table, such as, DEPARTMENT.

Parameters

Parameter	Description
%PromptField	Specifies the name of the field in the DERIVED record that has been defined as the prompt table for a field in a record definition.
RECORD.*recordname*	Specifies the name of the record definition that contains the correct standard system edits to be performed for that field in the message.

Returns

None.

See Also

Chapter 22, “Message Classes,” ExecuteEdits, page 748

Chapter 15, “Field Class,” EditError, page 504

Chapter 15, “Field Class,” MessageNumber, page 512

Chapter 15, “Field Class,” MessageSetNumber, page 513

Chapter 31, “Record Class,” IsEditError, page 1403

Chapter 31, “Record Class,” SetEditTable, page 1398

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, “Creating Record Definitions,” Creating a New Record

SetQueryString

Syntax

SetQueryString(*Parameter_Name, Value***)**

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetQueryString method to add a parameter value to the query string.

Note. This method has been deprecated and remains for backward compatibility only. Use the IBConnectorInfo class AddQueryStringArg method instead.

See Also

Chapter 22, "Message Classes," AddQueryStringArg, page 820

Parameters

Parameter	Description
Parameter_Name	Specify the name of the parameter that you want to add to the query string, as a string.
Value	Specify the value of the parameter you are adding to the query string.

Returns

None.

Example

```
Local Message &request;
&request = CreateMessage(OPERATION.QE_SALES_ORDER);
. . .

. . .

&request.SetQueryString("BUYER", "12345");
```

See Also

Chapter 22, "Message Classes," AddQueryStringArg, page 820 and Chapter 22, "Message Classes," GetQueryString, page 755.

SetStatus

Syntax

SetStatus(*Status*)

Description

Use the SetStatus method to set the status of a publication or subscription contract, much the same way you do in the message monitor.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class SetStatus method instead.

You may want to use this method after an end-user has finished fixing any errors in the message data.

You can set the status of a contract to one of the following statuses:

 Copyright © 1988-2007, Oracle. All rights reserved.

- Cancel
- Done
- Error
- New

The method is available only when the message instance has one of the following statuses:

- Cancel
- Done
- Error
- New

See Also

Chapter 22, “Message Classes,” SetStatus, page 798

Parameters

Parameter	Description
Status	Specify the status that you want to set the message to. Value are: • %Message_Error • %Message_New • %Message_Done • %Message_Started • %Message_Working • %Message_Retry • %Message_Timeout • %Message_Edited • %Message_Canceled

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Managing Messages,” Adding Message Definitions

SetXmlDoc

Syntax

SetXmlDoc(*&XmlDoc*)

Description

Use the SetXmlDoc method to load a nonrowset-based message with data from an XmlDoc object.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. This method can only be used for nonrowset-based message. If you use this method with a rowset-based message an error message is returned.

Considerations Using SetXmlDoc

In order to wrap non-XML data in cdata wraps as part of a message that would be automatically removed when posted to the gateway, the following cdata wraps are supported:

- `<?xml version="1.0"?>`
  `<data PsNonXml="Yes"><![CDATA..`
- `<?xml version="1.0"?>`
  `<data psnonxml="Yes"><![CDATA..`
- `<?xml version="1.0"?><add_some_root_tag>`
  `<data PsNonXml="Yes"><![CDATA...`
- `<?xml version="1.0"?><add_some_root_tag>`
  `<data psnonxml="Yes"><![CDATA...`

Parameters

Parameter	Description
&XmlDoc	Specify an already instantiated, populated XmlDoc object.

Returns

None.

See Also

Chapter 22, “Message Classes,” GetXmlDoc, page 757

Chapter 42, “XmlDoc Class,” page 1745

SyncRequest

Syntax

SyncRequest([*NodeName*])

Description

Use the SyncRequest method to send a synchronous message for the default message version.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class SyncRequest method instead.

Use the IsActive property to determine if a message is active or not before using this method.

This method sends a single message synchronously, that is a reply message is returned as a result of this method.

If you want to publish a message asynchronously, use the Publish method.

If you want to publish more than one message at a time synchronously, use the IntBroker SyncRequest method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. This method supports nonrowset-based messages only if the SetXmlDoc method is used to populate the message prior to using the SyncRequest method.

The SyncRequest method can be called from any PeopleCode event, but is generally called from an Application Engine PeopleCode step or from a component.

When sending a synchronous request from a component, you should send messages only from the SavePostChange event, either from record field or component PeopleCode. This way, if there's an error in your SyncRequest code, the data isn't committed to the database. Also, since SavePostChange is the last Save event fired, you avoid locking the database while the other save processing is happening.

If the message is successfully sent, a response message is returned. In addition, the GUID property is updated with a unique identifier.

See Also

Chapter 22, "Message Classes," SyncRequest, page 799

Parameters

Parameter	Description
NodeName	Specify the name of the node that you want to send this message to.

Returns

A response message.

Example

```
Local Message &request, &response;
Local Rowset &sales_order;

&sales_order = GetLevel0();
&request = CreateMessage(OPERATION.QE_SALES_ORDER);
&request.CopyRowsetDelta(&sales_order);
&response = &request.SyncRequest();
If (&response.ResponseStatus = 0) Then
   /* do processing */
End-If;
```

See Also

Chapter 22, "Message Classes," Publish, page 760

Chapter 22, "Message Classes," SyncRequest, page 799

Update

Syntax

Update([*versionlist*])

where *versionlist* is an arbitrary list of message versions in the following format:

Copyright © 1988-2007, Oracle. All rights reserved.

version1 [, *version2*] . . .

Description

The Update method updates a message in the message queue with the specified message versions.

Note. This method has been deprecated and remains for backward compatibility only. Use the IntBroker class Update method instead.

If *versionlist* isn't specified, the default message version is used. This method is commonly used in the OnRouteSend and OnRouteReceive PeopleCode events.

Note. This method does not support nonrowset-based messages. The Update method can't be called from notification PeopleCode.

See Also

Chapter 22, "Message Classes," Update, page 801

Parameters

Parameter	Description
versionlist	Specify an optional comma-separated list of message versions. If *versionlist* isn't specified, the default message version is used.

Returns

None.

See Also

Chapter 22, "Message Classes," GetRowset, page 755

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

UpdateSegment

Syntax

UpdateSegment()

Description

Use the UpdateSegment method to update the current segment with data from the message. This method is used only when creating a message for publication, not after a message has been published.

Note. You should use DeleteSegment and UpdateSegment only when writing to memory, or when SegmentsByDatabase is set to False. These methods do not work when writing to the database, or when SegmentsByDatabase is set to True.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

None.

See Also

Chapter 22, "Message Classes," CreateNextSegment, page 747 , Chapter 22, "Message Classes," DeleteSegment, page 748 , Chapter 22, "Message Classes," CurrentSegment, page 771 and Chapter 22, "Message Classes," SegmentCount, page 780.

Message Class Properties

In this section, we discuss the Message class properties. The properties are discussed in alphabetical order.

ActionName

Description

This property returns the name of the action associated with the message, as a string.

This property is read-only.

AliasName

Description

This property returns the name of the alias associated with the message part executing the property.

This property is read-only.

See Also

Chapter 22, "Message Classes," GetPartAliasName, page 752

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Specifying Record Aliases

ChannelName

Description

This property references the name of the channel associated with the message definition.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class QueueName property instead.

See Chapter 22, "Message Classes," QueueName, page 780.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is set when the message is created. The message instance keys are a subset of the publication and subscription contract keys. This property is one of the message instance keys: the others are PubID and PubNodeName for asynchronous messages, GUID and PubNodeName for synchronous messages.

This property is valid for both synchronous and asynchronous messages.

This property is read-only.

Example

```
&CHANNEL = &MSG.ChannelName
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

Cookies

Description

This property returns or sets the cookies associated with a message.

Note. This property has been deprecated and remains for backward compatibility only. Use the IBInfo class Cookies property instead.

See Chapter 22, "Message Classes," Cookies, page 811.

You can accept a synchronous response message containing cookies, save those cookies in a global variable, and later return them to the target node in an outbound synchronous or asynchronous request message.

You can access this property only in an inbound synchronous response message or an outbound request message.

This property is read-write.

Example

The following is an example of receiving cookies:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* send the synchronous request; the return value is */
/* the response message object */
&SalesResponse = &SalesRequest.SyncRequest();

/* Retrieve cookies from the response message */
&SalesCookies = &SalesResponse.Cookies;
```

The following is an example of returning cookies:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Insert the cookies in the request message */
&SalesRequest.Cookies = &SalesCookies;

/* Send the asynchronous request */
&SalesRequest.Publish();
```

CurrentSegment

Description

This property returns a number, indicating which segment is the current segment.

This property is read-only.

DefaultMessageVersion

Description

This property returns the default version of the message as a string.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class OperationVersion property instead.

See Chapter 22, "Message Classes," OperationVersion, page 778.

This is the message version from the pub/sub contract, not the default message version of the message defined in the current system.

This property is valid for both synchronous and asynchronous messages.

This property is read-only.

Example

```
Local Message &MSG;
Local String &VER;

&MSG = %IntBroker.GetMessage();
&VER = &MSG.DefaultMessageVersion;
```

Copyright © 1988-2007, Oracle. All rights reserved.

DoNotPubToNodeName

Description

Use this property to set the node name of a node that you do not want to publish this message to. For example, a single node may publish and subscribe to the same message. You can use this property to prevent the system from subscribing to the same message it publishes. This can help prevent circular processing where the original publisher eventually receives the same message.

This property is only valid for asynchronous messages.

This property is read-write.

Example

```
&MSG.DoNotPubToNodeName = &MSG.PubNodeName;
&MSG.Publish();
```

GUID

Description

This property returns a string representing a global unique identifier generated when the message was sent.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class TransactionId property instead.

See Chapter 22, "Message Classes," TransactionId, page 783.

This property returns the unique identifier used with the message.

This property is valid only for synchronous messages.

This property is read-only.

Example

```
Local Message &request;
Local String &guid;

&request = %IntBroker.GetMessage();

&guid = &request.GUID;
```

IBException

Description

Use the IBException property to return a reference to an exception object. The object is populated with information about the integration broker error.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 14, “Exception Class,” page 481

IBInfo

Description

Use this property to return a reference to an IBInfo object.

This property is read-only.

IsActive

Description

Indicates whether the message definition has been set to inactive.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class IsOperationActive property instead.

See Chapter 22, “Message Classes,” IsOperationActive, page 776.

This property is True if the message definition is active, False if it’s been inactivated. If you have a lot of PeopleCode associated with publishing a message, you might use this property to check if the message is active before you publish it.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&MSG = CreateMessage(OPERATION.MY_MESSAGE)
If &MSG.IsActive Then
  /* do PeopleCode processing */
  &MSG.Publish();
Else
  /* do other processing */
End-if;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Managing Messages,” Adding Message Definitions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” IsMessageActive

Copyright © 1988-2007, Oracle. All rights reserved.

IsDelta

Description

This property indicates if any fields in a message populated from page data have been changed since they were first loaded into the component buffer.

This is generally used in the following scenario:

You want to publish a message only if the end-user has changed a value on a page. The problem is that if the end-user makes a change to a field in the level three rowset, the IsChanged property for the top level row remains False. If the rowset is large, and you don't want to iterate through it to find a single IsChanged, you can use this property to quickly determine if something has changed at a lower level and the message should be published.

Note. This property should be used only when the message and the component do not have the same structure (that is, the same records at the same levels). If the message and the component have the same structure use the CopyRowsetDelta method instead.

This property takes a Boolean value: True if there are changed fields in the message, False otherwise.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&Msg = CreateMessage(OPERATION.MY_MESSAGE);
&Msg_Rowset = &Msg.GetRowset(); /* get first level in Msg RS */
&Msg_Detail = &Msg_Rowset(1).GetRowset(); /* get detail in Msg */
&Page_Rowset = GetRowset(); /* get page */
For &I = &Page_Rowset.Rowcount
  &Page_Detail = &Page_Rowset.GetRow(&I).GetRowset();
  &Msg_Detail.CopyRowset(&Page_Detail);
End-For;
/* Find if any data changed and should publish message */
If &Msg.IsDelta Then
   &Msg.Publish();
Else
   /* don't publish message and do other processing */
End-If;
```

See Also

Chapter 22, "Message Classes," CopyRowsetDelta, page 742

IsEditError

Description

This property is True if an error has been found on any field in any record of the current message after executing the ExecuteEdits method on either a message object or a record object. This property can be used with the Field Class properties MessageSetNumber and MessageNumber to find the error message set number and error message number.

 Copyright © 1988-2007, Oracle. All rights reserved.

You can use this property in notification PeopleCode with the Exit function with a value of 1. This automatically rolls back any changes that were made to the database, saves the message errors to the message queue, and marks the message status as ERROR.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
&Rowset = &MSG.GetRowset();
&MSG.ExecuteEdits();
If &MSG.IsEditError Then
  Exit(1);
End-If;
```

See Also

Chapter 22, “Message Classes,” ExecuteEdits, page 748

Chapter 15, “Field Class,” MessageNumber, page 512

Chapter 15, “Field Class,” MessageSetNumber, page 513

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” Exit

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Sending and Receiving Messages,” Processing Inbound Errors

IsEmpty

Description

This property indicates whether the transaction occurred without error, but returned an empty XML document. An empty response object contains only the data tags:

```
<data></data>
```

This property returns a Boolean value: True, the response message is empty, False, the response message is not empty.

This property is valid only for the returned response message from a synchronous request message.

This property is read-only.

IsLocal

Description

This property is True if the message object that was just instantiated was published locally. This property takes a Boolean value.

This property is valid for both asynchronous and synchronous messages.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
If &MSG.IsLocal Then /*  LOCAL NODE */
  /* do processing specific to local node */
Else
  /* do processing specific to remote nodes */
End-If;
```

IsOperationActive

Description

Use the IsOperationActive property to determine if an operation is active or not.

This property returns a boolean value: true if the operation is still active, false otherwise.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Messages," Adding Message Definitions

IsParts

Description

Use the IsParts property to determine if the message is a container message, that is, composed of parts.

This property only determines if a message is a container message. If you'd like to know if the message is a rowset-based message with parts, use the IsPartsStructured property instead.

This property returns a boolean value: true if the message executing the property is a container message, false otherwise.

This property is read-only.

See Also

Chapter 22, "Message Classes," IsPartsStructured, page 776

IsPartsStructured

Description

Use the IsPartsStructured property to determine if the message is a container message, that is, composed of parts, as well as a rowset-based message.

This property determines if a message is a container message as well as rowset-based. If you only want to know if the message is a container message, use the IsParts property instead.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property returns a boolean value: true if the message executing the property is a container message that is rowset-based, false otherwise.

This property is read-only.

See Also

Chapter 22, “Message Classes,” IsParts, page 776

IsRequest

Description

Use the IsRequest property to determine whether a message is a request message (as opposed to a response message.)

This property returns a boolean value: true if the message object executing the property is a request message, false otherwise.

This property is read-only.

IsSourceNodeExternal

Description

Use the IsSourceNodeExternal property to determine whether the message came from an internal system or a third-party system.

This property returns a boolean value: true if the message originated from a third-party system, false otherwise.

This property is read-only.

IsStructure

Description

This property indicates whether the message is rowset-based, that is, based on a rowset, or nonrowset-based that is, based on an XmlDoc. This property returns a boolean value, true if the message is rowset-based, false otherwise.

This property is read-only.

MessageDetail

Description

This property specifies the message detail, as a string. The message detail can be used to further identify a message.

Note. This property has been deprecated. It is no longer supported.

Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property returns the name of the message as a string.

This property is valid with both asynchronous and synchronous messages.

This property is read-only.

NRId

Description

This property returns the non-repudiation ID as a string. This property is populated with a unique string when the message is published.

This property is only valid with messages that use non-repudiation.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Setting Up Secure Integration Environments," Implementing Nonrepudiation

OperationName

Description

Use the OperationName property to return the name of the operation associated with the message object executing the property, as a string.

This property is read-only.

OperationVersion

Description

Use the OperationVersion property to determine version for the operation associated with the message object executing the property, as a string.

This property is read-only.

ParentTransactionId

Description

Use the ParentTransactionId property to return the parent transaction Id, that is, the Id that is generated with the original request message from a third-party system.

This property is read-only.

See Also

Chapter 22, "Message Classes," TransactionId, page 783

 Copyright © 1988-2007, Oracle. All rights reserved.

PartCount

Syntax

Use the PartCount property to determine the number of parts in a container message. This property is only valid with container messages. You will receive an error if you try to use it on a message that isn't a container message.

This property is read-only.

PubID

Description

This property refers to the publication identifier, which is a number.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class QueueSeqId property instead.

See Chapter 22, "Message Classes," QueueSeqId, page 780.

It's generated for asynchronous messages when the message is published, and is sequential, based on the number of messages published for that channel. The message instance keys are a subset of the publication and subscription contract keys. This property is one of the message instance keys: the others are ChannelName and PubNodeName.

Note. The PubID property is not populated when the message is published from a notification.

This property is valid only with asynchronous messages.

This property is read-only.

Example

```
&MSG.Publish();
&PUBID = &MSG.PubID;
```

PubNodeName

Description

This property refers to a string that contains the name of the publishing node. It is generated by the system when the message is published. The message instance keys are a subset of the publication and subscription contract keys.

This property is valid for both asynchronous and synchronous messages.

For a synchronous message, this property returns the name of the requesting node.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
&PUBNODE = &MSG.PubNodeName;
```

Copyright © 1988-2007, Oracle. All rights reserved.

QueueName

Description

Use the QueueName property to return the name of the queue associated with the message.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Service Operation Queues"

QueueSeqId

Description

Use the QueueSeqId property to return the sequence number of the message in the message queue.

This property is read-only.

ResponseStatus

Description

This property is valid only for the returned response message from a synchronous request message.

This property indicates whether a response was successful or not.

Note. This is not the same as SetStatus, which is used only with the message monitor.

This property returns a numeric value: 0, the response was successful, any other number indicates an error.

This property is read-only.

SegmentCount

Description

This property returns the total number of segments for the message.

This property is read-only.

SegmentsByDatabase

Description

Use this property to specify whether segments are stored in memory while being processed.

If you specify SegmentsByDatabase as false, you can only have the number of segments as specified in PSADMIN (Message Segment From DB). If you specify this property as true, you can have as many segments as you need.

 Copyright © 1988-2007, Oracle. All rights reserved.

The SegmentsByDatabase property also specifies whether the segments are kept in memory or written to the database when they are received. If you specify true, the segments are automatically written to the database. If you specify false, the segments are held in memory. If you specify true, then cancel out of the program processing the segments, the changes are not committed to the database.

This property is automatically set to true for message segments being processed using a PeopleSoft Application Engine program.

This property is read-write.

See Also

Chapter 22, "Message Classes," Message Segments, page 735

Chapter 22, "Message Classes," SegmentsUnOrder, page 817

Size

Description

This is the approximate size of the uncompressed XML data generated when the message is published. The size is approximate for the following reasons:

- The maximum size of fields is used except for the case of long text and image fields.
- The active row count is used, without regard to whether the rows have been changed. In a CopyRowsetDelta, rows that are not changed, are not published.
- It doesn't include the size of the XML tags.

This property can be used with the system property %MaxMessageSize in a batch Application Engine program to prevent the application from publishing a message that is too large.

This property is valid for both asynchronous and synchronous messages.

This property is read-only.

Example

```
If &MSG.Size > %MaxMessageSize Then
  &MSG.Publish();
Else
  /*Move more data into the message object */
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "System Variables," %MaxMessageSize

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

Copyright © 1988-2007, Oracle. All rights reserved.

SubName

Description

This property refers to a string that contains the name of the notification process currently processing the message.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class ActionName property instead.

See Chapter 22, "Message Classes," ActionName, page 769.

It is available only when GetMessage is used in a notification PeopleCode program.

This property is only valid for asynchronous messages.

This property is read-only.

Example

```
&MSG = %IntBroker.GetMessage();
&SUBNAME = &MSG.SubName
```

SubQueueName

Description

Use the SubQueueName to return or specify the name of the sub-queue associated with the message, as a string.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Managing Service Operation Queues"

SubscriptionProcessId

Description

This property returns a string containing the subscription process identifier. This is a unique sequential number.

Note. This property has been deprecated and remains for backward compatibility only. Use the Message class TransactionId property instead.

See Chapter 22, "Message Classes," TransactionId, page 783.

This property is populated only if the "Generate Subscription Process Instance" option is turned on in the Subscription Process definition.

The subscription process identifier corresponds to the subscription process instance, not the message instance. If there are multiple subscription processes for the same message each subscription process will have a different, unique ID.

 Copyright © 1988-2007, Oracle. All rights reserved.

If the subscription process terminated abnormally, its process instance is lost, and a new one is generated on the next retry (that is, there will be a gap in the sequence.)

This property is valid only for asynchronous messages.

This property is read-only.

Considerations for Using SubscriptionProcessId

Consider the following when using SubscriptionProcessId:

- Using the Run PeopleCode option from Message Subscription does not generate the number. It is generated only if the Application Server is running the PeopleCode.
- If for some reason you have to rerun the subscription, a new number is assigned.
- Because generating the number is optional, your program must contain code to support the option being turned off. Here is a code example from ITEM_SYNC:

```
If All(&MSG.SubscriptionProcessId) Then
  &EIP_CTL_ID = Generate_EIP_CTL_ID(4, &MSG.SubscriptionProcessId);
Else
  &EIP_CTL_ID = Generate_EIP_CTL_ID(1, "Random");
End-If;
```

- If the flag is off, this property is blank. In this case, you may want to adopt a standard for generating a random number, as shown in the previous example.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

TransactionId

Description

Use the TransactionId property to return the transaction ID for an already published message, as a string. This is a unique transaction identifier for the message.

This property is read-only.

Version

Description

Use the Version property to determine the version of the message executing the property, as a string.

This property is read-only.

IntBroker Class

An IntBroker object is returned from the system variable %IntBroker.

Copyright © 1988-2007, Oracle. All rights reserved.

IntBroker Class Methods

In the following section, we discuss the IntBroker class methods. The methods are described in alphabetical order.

Cancel

Syntax

Cancel(*TransactionId*, *QueueName*, *DataType*, *SegmentIndex* | *TransactionIdArray*, *QueueNameArray*, *DataTypeArray*, *SegmentIndexArray*)

Description

Use the Cancel method to programmatically cancel either a message, or a list of messages, from the message queue, much the same as you can do in the message monitor.

This method is only available when the message has one of the following statuses:

- Edited
- Error
- New
- Retry
- Timeout

If you are specifying an array of messages to be canceled, and any message in the array fails for any reason (for example, you specify a message that doesn't have the correct status) no messages are canceled and the method return value is false.

Parameters

Parameter	Description
TransactionId | *TransactionIdArray*	Specify either a single transaction Id as a string, or specify an array of string containing the transaction Ids of the messages you want to cancel.
QueueName | *QueueNameArray*	Specify either a single queue name as a string, or specify an array of string containing the queue names that contain the messages you want to cancel.
DataType | *DataTypeArray*	Specify either a single data type, or an array of string containing the data types. Valid data types are:

Value	Description
%IntBroker_BRK	Cancel the message for the web services gateway.
%IntBroker_PUB	Cancel the message for the publication contract.
%IntBroker_SUB	Cancel the message for the subscription contract.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameter	Description
SegmentIndex | *SegmentIndexArray*	Specify either a specific segment, or an array of integer containing the segment numbers you want to cancel.

Returns

A boolean value: true if the message or messages are canceled successfully, false otherwise.

See Also

Chapter 22, "Message Classes," Resubmit, page 796

ConnectorRequest

Syntax

ConnectorRequest(*&Message*)

Description

Use the ConnectorRequest method to return a response message from the connector, based on the message passed in.

You would only use this method after you had set the connector information.

Parameters

Parameter	Description
&Message	Specify an already instantiated message object to return a response message from the connector.

Returns

A reference to a message object.

Example

```
   Local string &nonXmlData = "<?xml version=""1.0""?><data psnonxml=""yes""><!⇒
[CDATA[" | &encoded | "]]></data>";
   &soapMsg = CreateXmlDoc(&nonXmlData);

   &reqMsg.SetXmlDoc(&soapMsg);

   Local Message &respMsg;

   %This.SetConnectorProperties(&reqMsg, &url);

   &respMsg = %IntBroker.ConnectorRequest(&reqMsg);

   If &respMsg = Null Then /* Throw exception */
      %This.HandleNullSoapResponse(&soapMsg, &url);
   End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, “Message Classes,” ConnectorRequestUrl, page 786

ConnectorRequestUrl

Syntax

ConnectorRequestUrl(*URL*)

Description

Use the ConnectorRequestUrl method to return the URL for a connector.

You must have already set the connector parameters before executing this command.

Parameters

Parameter	Description
URL	Specify an absolute URL as a string.

Returns

A string.

Example

```
/**
 * Get XML Doc from URL
 *
 * @param String - Location
 * @return XmlDoc - retrieved XmlDoc
**/

method getXmlDocumentFromURL
   /+ &location as String +/
   /+ Returns XmlDoc +/
   Local XmlDoc &xmldoc;
   Local string &xmlstr;

   If Substring(LTrim(RTrim(Upper(&location))), 1, 4) = "HTTP" Then
      &xmlstr = %IntBroker.ConnectorRequestUrl(&location);
   Else
      &xmlstr = %This.getXmlContentFromFile(&location);
   End-If;

   &xmldoc = CreateXmlDoc("");

   If &xmldoc.ParseXmlString(&xmlstr) Then
      Return &xmldoc;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
      End-If;

   end-method;
```

See Also

Chapter 22, "Message Classes," ConnectorRequest, page 785

DeleteOrphanedSegments

Syntax

DeleteOrphanedSegments(*TransactionId*)

Description

Use the DeleteOrphanedSegments method to delete segments that might have been orphaned if you were processing message segments using a PeopleSoft Application Engine program that had to be restarted. Since each segment is committed to the database, if the application engine program is aborted, these segments need to be deleted from the database.

Parameters

Parameter	Description
TransactionId	Specify the transaction Id that identifies the message which contains the segments you want to delete.

Returns

A boolean value: true if the method completes successfully, false otherwise.

See Also

Chapter 22, "Message Classes," SegmentRestart, page 762

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Understanding Message Segments

GetArchData

Syntax

GetArchData(*TransactionId, SegmentIndex*)

Description

Use the GetArchData method to retrieve an archived message from the message queue.

Note. This method shouldn't be used in standard message processing. It should only be used when accessing archived messages.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TransactionId	Specify the transaction Id of the archived message you want to retrieve.
SegmentIndex	Specify the number of the segment of the archived message that you want to retrieve.

Returns

An XML string containing the archived data.

See Also

Chapter 22, "Message Classes," GetArchIBInfoData, page 788

GetArchIBInfoData

Syntax

GetArchIBInfoData(*TransactionId, ParentTransactionId*)

Description

Use the GetArchIBInfoData method to retrieve return archived IBInfo data.

Note. This method shouldn't be used in standard message processing. It should only be used when accessing archived messages.

Parameters

Parameter	Description
TransactionId	Specify the transaction Id of the archived message you want to retrieve.
ParentTransactionId	Specify the parent transaction Id, for the message.

Returns

An XML string containing the archived data.

See Also

Chapter 22, "Message Classes," GetArchData, page 787

GetIBInfoData

Syntax

GetIBInfoData(*TransactionId, DataType*)

Description

Use the GetIBInfoData to return the specified IBInfo data.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TransactionID	Specify the transaction ID of the message that you want to access.
DataType	Specify the data type of the message you want to access. Valid values are:

Value	Description
%IntBroker_BRK	Get the message for the web services gateway.
%IntBroker_PUB	Get the message for the publication contract.
%IntBroker_SUB	Get the message for the subscription contract.

Returns

An XML string containing the message data.

See Also

Chapter 22, "Message Classes," GetArchIBInfoData, page 788

GetMessage

Syntax

```
GetMessage([TransactionId, DataType])
```

Description

Use the GetMessage method to return a message object.

If you specify a transaction ID and data type, the GetMessage method populates the message object with the data from that message.

If you do not specify any parameters, the GetMessage method doesn't populate the message with data. Instead, it creates a new instance of a message object. In this case, you must use another method, such as GetRowset, to populate the message object. You must always populate the message object with data before running any methods on it.

Parameters

Parameter	Description
TransactionId	Specify the transaction ID for the message that you want returned.
DataType	Specify the type of message that you want returned. Valid values are:

Value	Description
%IntBroker_BRK	Get the message for the web services gateway.
%IntBroker_PUB	Get the message for the publication contract.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
%IntBroker_SUB	Get the message for the subscription contract.

Returns

A reference to a message object if successful, Null if not successful.

Example

The following example returns a populated message object:

```
Local Message &MSG;
Local string &TransactionId;

&TransactionId = "0f3617dl-c6f4-11d9-a4bd-c12cbalbc2f9"
&MSG = %IntBroker.GetMessage(&TransactionId, &IntBroker_BRK);
```

See Also

Chapter 22, “Message Classes,” GetRowset, page 755

Chapter 22, “Message Classes,” TransactionId, page 783

GetMessageErrors

Syntax

GetMessageErrors(*TransactionId*)

Description

Use the GetMessageErrors method to return an array that contains the errors that an application has set for the message, and have been written to the error queue. This method is generally only used with asynchronous messages.

Parameters

Parameter	Description
TransactionId	Specify the transaction Id for the message, as a string.

Returns

An array of string. If there are no error messages, the Len property of the array is 0.

Example

The following is an example of using the method for returning web services gateway error messages:

```
&MyErrors = %IntBroker.GetMessageErrors(&TransactionId);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, "Message Classes," SetErrorMessage, page 797

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

GetMsgSchema

Syntax

```
GetMsgSchema(MsgName, MsgVersion)
```

Description

Use the GetMsgSchema method to access the saved schema for the specified message.

If you specify a message that does not have a saved schema, you receive a null value.

Parameters

Parameter	Description
MsgName	Specify the name of the message for which you want to access a schema. You can either specify the name of the message as a string, or preface the message name with the keyword **Message.**
MsgVersion	Specify the message version.

Returns

A string containing the message schema. If you try to get a schema for a message that does not have a saved schema, returns null.

See Also

Chapter 22, "Message Classes," MsgSchemaExists, page 794

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Building Message Schemas"

GetSyncIBInfoData

Syntax

```
GetSyncIBInfoData(TransactionId, %LogType [, Archive])
```

Description

Use the GetSyncIBInfoData method to return a log containing information about synchronous transactions.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TransactionId	Specify the transaction ID of the published message.
LogType	Specify the type of log you want returned, based on the type of message. Valid values are:

Value	Description
%Sync_RequestOrig	Gets the log for a sync request original data message.
%Sync_RequestTrans	Gets the log for a sync request transformed data message.
%Sync_ResponseOrig	Gets the log for a sync response original data message.
%Sync_ResponseTrans	Gets the log for a sync response transformed data message.

Parameter	Description
Archive	Specify whether to retrieve any archived logs as well. This parameter takes a boolean value: true to return archived logs, false otherwise. The default value is false.

Returns

An XML string containing the log data.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Using the Service Operations Monitor"

GetSyncLogData

Syntax

```
GetSyncLogData(TransactionId, %LogType [, Archive])
```

Description

Use the GetSyncLogData method to return a log containing information about the specified synchronous message.

You can use this information for debugging. Using this method, you can obtain the request and response data in a synchronous request, both pre- and post-transformation.

This function is used in the PeopleCode for the Message Monitor.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TransactionId	Specify the transaction ID of the published message.
LogType	Specify the type of log you want returned, based on the type of message. Valid values are:

Value	Description
%Sync_RequestOrig	Gets the log for a sync request original data message.
%Sync_RequestTrans	Gets the log for a sync request transformed data message.
%Sync_ResponseOrig	Gets the log for a sync response original data message.
%Sync_ResponseTrans	Gets the log for a sync response transformed data message.

Parameter	Description
Archive	Specify whether to retrieve any archived logs as well. This parameter takes a boolean value: true to return archived logs, false otherwise. The default value is false.

Returns

An XML string containing the log data.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Using the Service Operations Monitor"

InBoundPublish

Syntax

```
InBoundPublish(&Message)
```

Description

Use the InBoundPublish method to send an asynchronous message based on the specified message to simulate an inbound asynchronous request from an external node.

Though you are sending a message to yourself, it goes through all the inbound message processing that the specified message would.

Parameters

Parameter	Description
&Message	Specify an already instantiated message object that you want to use for the message simulation.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A boolean value, true if the message is published successfully, false otherwise.

See Also

Chapter 22, "Message Classes," Publish, page 795 and Chapter 22, "Message Classes," SyncRequest, page 799.

IsOperationActive

Syntax

IsOperationActive(*OperationName* [,*OperationVersion*])

Description

Use the IsOperationActive method to determine if an operation is active or not.

Parameters

Parameter	Description
OperationName	Specify the name of the operation that you want to check the status of.
OperationVersion	Specify the version of the specified operation that you want to check the status of, as a string.

Returns

A boolean value: true if the operation is active, false otherwise.

MsgSchemaExists

Syntax

MsgSchemaExists(*MsgName, MsgVersion*)

Description

Use the MsgSchemaExists method to determine if a schema has been created and saved for the specified message.

Parameters

Parameter	Description
MsgName	Specify the name of the message for which you want to access a schema. You can either specify the name of the message as a string, or preface the message name with the keyword **Message.**
MsgVersion	Specify the message version.

Returns

A boolean value: true if the message schema exists, false otherwise.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, "Message Classes," GetMsgSchema, page 791

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Building Message Schemas"

Publish

Syntax

```
Publish(&Message [, &ArrayofNodeNames] [,IsEnqueued])
```

Description

The Publish method publishes a message to the message queue for the default message version.

This method does not publish the message if the IsOperationActive property for the message is False.

This method publishes a message asynchronously, that is, a reply message is not automatically generated. To publish a message synchronously, use the SyncRequest method.

Note. This method supports nonrowset-based messages only if the data is populated using the SetXmlDoc method.

If the Publish method is called and the message isn't populated (either using SetXmlDoc or one of the rowset methods,) an empty message is published.

The Publish method can be called from any PeopleCode event, but is generally called from an Application Engine PeopleCode step or from a component.

When publishing from a component, you should publish messages only from the SavePostChange event, either from record field or component PeopleCode. This way, if there's an error in your publish code, the data isn't committed to the database. Also, since SavePostChange is the last Save event fired, you avoid locking the database while the other save processing is happening.

However, until the message is published, the tables involved with the component are locked and are not saved. To avoid problems with this, specify the *Deferred_Mode* property as true, so the message is published after the table data has been committed to the database.

Note. If you're publishing a message from within an Application Engine program, the message won't actually be published until the calling program performs a Commit.

Parameters

Parameter	Description
&Message	Specify an already instantiated message object to be published.
&ArrayofNodeNames	Specify an already instantiated array of string containing the names of the nodes that you want to publish to.
IsEnqueued	Specify whether the message is to be enqueued or not. This parameter takes a boolean value: true if the message is to be enqueued, false otherwise. The default value is false.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

```
Local Message &Msg;
Local Rowset &rs;
Local IntBroker &Ib;

&Flight_profile = GetLevel0();

&Msg = CreateMessage(Operation.ASYNC, %IntBroker_Request);

&Msg.CopyRowset(&FlightProfile);

&Ib = %IntBroker;

&Ib.Publish(&Msg);
```

See Also

Chapter 22, “Message Classes,” SyncRequest, page 799

Resubmit

Syntax

Resubmit(*TransactionId*, *QueueName*, *DataType*, *SegmentIndex* | *TransactionIdArray*, *QueueNameArray*, *DataTypeArray*, *SegmentIndexArray*)

Description

Use the Resubmit method to programmatically resubmit either a message, or a list of messages, from the message queue, much the same as you can do in the message monitor.

You may want to use this method after an end-user has finished fixing any errors in the message data, and you want to resubmit the message, rerunning the PeopleCode.

This method is only available when the message has one of the following statuses:

- Canceled
- Edited
- Error
- Timeout

If you are specifying an array of messages to be resubmitted, and any message in the array fails for any reason (for example, you specify a message that doesn’t have the correct status) no messages are resubmitted and the method return value is false.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TransactionId | *TransactionIdArray*	Specify either a single transaction Id as a string, or specify an array of string containing the transaction Ids of the messages you want to resubmit.
QueueName | *QueueNameArray*	Specify either a single queue name as a string, or specify an array of string containing the queue names that contain the messages you want to resubmit.
DataType | *DataTypeArray*	Specify either a single data type, or an array of string containing the data types. Valid data types are:

Value	Description
%IntBroker_BRK	Resubmit the message for the web services gateway.
%IntBroker_PUB	Resubmit the message for the publication contract.
%IntBroker_SUB	Resubmit the message for the subscription contract.

Parameter	Description
SegmentIndex | *SegmentIndexArray*	Specify either a specific segment, or an array of integer containing the segment numbers you want to resubmit.

Returns

A boolean value: true if the message or messages are resubmitted successfully, false otherwise.

See Also

Chapter 22, "Message Classes," Cancel, page 784

SetErrorMessage

Syntax

```
SetErrorMessage(TransactionId, MsgSet, MsgNumber, Error_Location, ParmList)
```

Description

Use the SetErrorMessage method to write messages to the error queue. This method is used with asynchronous messages only.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TransactionId	Specify the transaction Id for the message, as a string.
MsgSet	Specify the number of the message set that the message associated with the error is associated with.
MsgNumber	Specify the message number that you want to associate with this error.
Error_Location	Specify where the error occurred, or any additional information you want to give about the error. This is a string, and allows 254 characters.
ParmList	Specify up to four values to be used as replacement values for the message text. The first value in *ParmList* is used to replace the first replacement character (that is, %1,) the second is used to replace the second (%2) and so on.

Returns

A boolean value, true if the error message was associated correctly, false otherwise.

Example

The following is an example of setting an error message for the web services gateway:

```
&Rslt = %IntBroker.SetErrorMessage(&TransactionId, &msgset, &msgnum, "error⇒
 location", &parm1, &parm2. &parm3, &parm4 );
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages," Processing Inbound Errors

Chapter 22, "Message Classes," GetMessageErrors, page 790

SetStatus

Syntax

SetStatus(*&Message*, *Status*)

Description

Use the SetStatus method to set the status of a publication or subscription contract, much the same way you do it in the message monitor.

You may want to use this method after an end-user has finished fixing any errors in the message data.

You can set the status of a contract to one of the following statuses:

- Canceled
- Done
- Error
- New

 Copyright © 1988-2007, Oracle. All rights reserved.

The method is available only when the message instance has one of the following statuses:

- Canceled
- Done
- Error
- New

Parameters

Parameter	Description
&Message	Specify an already instantiated message object that you want to set the status for.
Status	Specify the status that you want to set the message to. Valid status are:

Value	Description
%Operation_Canceled	Cancel the message.
%Operation_Done	Set the message to done.
%Operation_Error	Set the message to be in error.
%Operation_New	Specify the message as new.

Returns

None.

SyncRequest

Syntax

```
SyncRequest([&MsgArray, &NodeNames])
```

Description

Use the SyncRequest method to send multiple messages at the same time. You should only use this message with synchronous messages. You can also use this method to send a single synchronous message at a time, or you can use the SyncRequest Message class method.

If you want to publish messages asynchronously, use the Publish method.

Note. This method supports nonrowset-based messages only if the SetXmlDoc method is used to populate the message prior to using the SyncRequest method.

You must set the thread pool size of the application server on the receiving system to a value greater than 1 (the default) in order to process more than one message at a time.

How many threads you specify determines how many messages are worked on simultaneously. For example, if you have 10 messages in *&MsgArray*, and your thread pool size is 5, then only 5 messages are processed at a time.

The maximum number you can specify for your thread pool size is 10.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&MsgArray	Specify an already instantiated array of messages that contains the messages you want to publish.
&NodeNames	Specify an already instantiated array of string containing the names of the nodes that you want to publish to. The first message in the array of messages is published to the first node, the second message is published to the second node, and so on.

Returns

An array of messages. These are the corresponding messages returned for the published messages. The first message in the returned array corresponds to the first message in the published array, the second message with the second, and so on.

Example

The following is an example of how creating and receiving a series of messages:

```
Local Rowset &FLIGHTPLAN, &FLIGHTPLAN_RETURN;
Local Message &MSG;
Local File &BI_FILE;
Declare Function out_BI_results PeopleCode QE_FLIGHTDATA.QE_ACNUMBER FieldFormula;

Local array of Message &messages;
Local array of Message &return_mesages;

&messages = CreateArrayRept(&MSG, 0);
&return_mesages = CreateArrayRept(&MSG, 0);

&FLIGHT_PROFILE = GetLevel0();
&messages [1] = CreateMessage(OPERATION.QE_FLIGHTPLAN_SYNC);
&messages [1].CopyRowset(&FLIGHT_PROFILE);

&messages [2] = CreateMessage(OPERATION.QE_FLIGHTPLAN_SYNC);
&messages [2].CopyRowsetDelta(&FLIGHT_PROFILE);

&return_mesages = %IntBroker.SyncRequest(&messages);

&FLIGHTPLAN_RETURN = &return_mesages [1].GetRowset();
&temp = &return_mesages [1].GenXMLString();

out_BI_results(&temp);

&FLIGHTPLAN_RETURN = &return_mesages [2].GetRowset();
&temp = &return_mesages [2].GenXMLString();

out_BI_results(&temp);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, “Message Classes,” SyncRequest, page 766

Update

Syntax

```
Update(&Message)
```

Description

Use the Update method to update a message in the message queue with the specified message object.

Parameters

Parameter	Description
&Message	Specify an already instantiated and populated message object to be used to update an existing message in the message queue.

Returns

A boolean value, true if the message is updated successfully, false otherwise.

See Also

Chapter 22, “Message Classes,” UpdateXmlDoc, page 801

UpdateXmlDoc

Syntax

```
UpdateXmlDoc(&XmlDoc, TransactionId, DataType)
```

Description

Use the UpdateXmlDoc method to update a message in the message queue with the specified message data.

Note. This method shouldn’t be used in a subscription program.

Parameters

Parameter	Description
&XmlDoc	Specify an already instantiated and populated XmlDoc object.
TransactionId	Specify the transaction ID of the message you want to update.
DataType	Specify the data type of the message that you want to update. Valid values are:

Value	Description
%IntBroker_BRK	Update the message for the web services gateway.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
%IntBroker_PUB	Update the message for the publication contract.
%IntBroker_SUB	Update the message for the subscription contract.

Returns

A boolean value, true if the message was updated successfully, false otherwise.

See Also

Chapter 22, "Message Classes," Update, page 801

IBInfo Class

An IBInfo object is returned from the IBInfo message class property.

See Also

Chapter 22, "Message Classes," IBInfo, page 773

IBInfo Class Methods

In this section, we discuss the IBInfo class methods. The methods are discussed in alphabetical order.

All methods work with both asynchronous as well as synchronous messages unless specified

AddAttachment

Syntax

```
AddAttachment(Path)
```

Description

Use the AddAttachment method to add an attachment to a message.

Parameters

Parameter	Description
Path	Specify an absolute path, including the file name and type, as a string, of the file you want to attach to the message.

Returns

A string containing a content ID, which can be used for accessing the attachment with other methods, such as DeleteAttachment or SetAttachmentProperty.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example shows sending an attachment with an asynchronous message:

```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE;
Local String &Attachment_Id;
Local Boolean &Test;
Local IntBroker &IntBroker;

QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(Operation.ASYNC_RR);

&Attachment_Id = &MSG.IBInfo. AddAttachment("C:\\temp\\MyFile.txt");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Encoding,⇒
 "UTF-8");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Base,⇒
 "Standard");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Disposition,⇒
 "Pending");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Language,⇒
 "English");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Description,⇒
 "Parts data");

&MSG.CopyRowset(&FLIGHT_PROFILE);

&IntBroker = %IntBroker

&IntBroker.Publish(&MSG);
```

See Also

Chapter 22, “Message Classes,” ClearAttachments, page 803 , Chapter 22, “Message Classes,” DeleteAttachment, page 804 and Chapter 22, “Message Classes,” SetAttachmentProperty, page 809.

ClearAttachments

Syntax

ClearAttachments()

Description

Use the ClearAttachments method to clear all attachments. If you want to delete a particular attachment, use DeleteAttachment instead.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 22, "Message Classes," AddAttachment, page 802 and Chapter 22, "Message Classes," DeleteAttachment, page 804.

DeleteAttachment

Syntax

DeleteAttachment(*Index* | *Content_Id*)

Description

Use the DeleteAttachment method to remove the specified attachment from the message. You can either specify the number of the attachment, or the content ID associated with the attachment (generated when the attachment was added to the message with AddAttachment.)

If you want to clear all attachments, instead of a particular one, use the ClearAttachments methods instead.

Parameters

Parameter	Description
Index | *Content_Id*	Specify either the number of the attachment, or the content ID associated with the attachment, for the attachment you want to delete.

Returns

A boolean value: true if the attachment was deleted successfully, false otherwise.

See Also

Chapter 22, "Message Classes," AddAttachment, page 802 and Chapter 22, "Message Classes," ClearAttachments, page 803.

GetAttachmentContentID

Syntax

GetAttachmentContentID(*Index*)

Description

Use the GetAttachmentContentID to return the content ID for the specified attachment. The content ID is associated with an attachment when it is added to a message using AddAttachment.

You can use the content ID with other methods, such as AddAttachmentProperty and DeleteAttachment.

Parameters

Parameter	Description
Index	Specify the number of the attachment that you want to access the content ID for.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A string containing the content ID.

See Also

Chapter 22, "Message Classes," AddAttachment, page 802 and Chapter 22, "Message Classes," GetAttachmentProperty, page 805.

GetAttachmentProperty

Syntax

GetAttachmentProperty(*Content_ID*, *Property_Type*)

Description

Use the GetAttachmentProperty method to return the value of an attachment property.

Parameters

Parameter	Description
Content_ID	Specify the content ID of the attachment that you want to access, as a string.
Property_Type	Specify the type of property that you want to access. Valid values are:

Value	Description
%Attachment_Base	Specifies the base name of the attachment, that is, if the actual document name you want is different, such as if it is zipped or otherwise compressed as part of other documents.
%Attachment_Description	Specifies the description of the attachment.
%Attachment_Disposition	Specifies the disposition of the attachment, whether it's displayed inline or as an attachment.
%Attachment_Encoding	Specify the encoding of the attachment. For example, `Content-type: text/plain` or `charset=Big5`.
%Attachment_Language	Specify the language used in the attachment, as a string.
%Attachment_Location	Specify an additional location for the attachment.
%Attachment_Type	Specify the attachment type.
%Attachment_URL	Specifies the URL for the attachment. The URL must be an absolute URL.

Returns

A string containing the value of the specified property.

Example

The following example processes an attachment from a notification PeopleCode program:

```
Import PS_PT:Integration:INotificationHandler;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Class FLIGHTPROFILE implements PS_PT:Integration:INotificationHandler
   method FLIGHTPROFILE();
   method OnNotify(&MSG As Message);

end-class;

/* Constructor */
method FLIGHTPROFILE
   %Super = create PS_PT:Integration:INotificationHandler();
end-method;

method OnNotify
   /+ &MSG as Message +/
   /+ Extends/implements PS_PT:Integration:INotificationHandler.OnNotify +/

Local rowset &RS;
Local integer &Count;
Local string &Attachment_Id;
Local array of array of string &Result;

&RS = &MSG.GetRowset();

&Count = &MSG.IBInfo.NumberOfAttachments;
If &Count > 0 Then

   For &I = 1 to &Count
      &Attachment_ID = &MSG.IBInfo.GetAttachmentContentID(&I);
      &Result[&I][1] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Encoding);
      &Result[&I][2] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Type);
      &Result[&I][3] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_URL);
      &Result[&I][4] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Base);
      &Result[&I][5] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Location);
      &Result[&I][6] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Disposition);
      &Result[&I][7] = &MSG.IBInfo.GetAttachmentProperty(&Attachment_Id,⇒
 %Attachment_Description);

   End-For;

End-if;

/* process data from message */

end-method;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, "Message Classes," SetAttachmentProperty, page 809

LoadConnectorProp

Syntax

LoadConnectorProp(*ConnectorName*)

Description

Use the LoadConnectorProp method to load connector properties to the specified connector. The properties are contained in the message executing the method.

Note. Use this method in the message OnSend event.

Parameters

Parameter	Description
ConnectorName	Specify the name of the connector that you want to load properties for from the message.

Returns

A Boolean value: true if properties are loaded successfully, false otherwise.

Example

```
LOCAL MESSAGE &MSG;
&MSG = %IntBroker.GetMessage();
&Rowset = &MSG.GetRowset();
&MSG.IBInfo.LoadConnectorProp("HTTP TargetConnector");
/* add connector properties */
&MSG.IBInfo.ConnectorOverride= true;
ReturnToServer(&MSG);
```

See Also

Chapter 22, "Message Classes," ConnectorOverride, page 811

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReturnToServer

LoadConnectorPropFromNode

Syntax

LoadConnectorPropFromNode(*NodeName*)

Description

Use the LoadConnectorPropFromNode method to load connector properties into the message executing the method. Then you can use the LoadConnectorProp method to load the specified connector with the properties.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. Use this method in the message OnSend event.

Parameters

Parameter	Description
NodeName	Specify the node that contains the connector properties you want to use. You can either specify the node name as a string, or prefix the node name with the reserved word **Node**.

Returns

A Boolean value: true if the properties are loaded successfully, false otherwise.

See Also

Chapter 22, "Message Classes," ConnectorOverride, page 811

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReturnToServer

LoadConnectorPropFromRouting

Syntax

```
LoadConnectorPropFromRouting(RoutingDefnName)
```

Description

Use the LoadConnectorPropFromRouting method to load connector properties into the message executing the method. Then you can use the LoadConnectorProp method to load the specified connector with the properties.

Parameters

Parameter	Description
RoutingDefnName	Specify the routing definition name that contains the connector properties you want to use, as a string.

Returns

A boolean value: true if the method completes successfully, false otherwise.

See Also

Chapter 22, "Message Classes," LoadConnectorProp, page 807 and Chapter 22, "Message Classes," LoadConnectorPropFromNode, page 807.

LoadConnectorPropFromTrx

Syntax

```
LoadConnectorPropFromTrx(NodeName, TransactionType, MsgName, MsgVersion)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Note. This method is no longer supported.

SetAttachmentProperty

Syntax

SetAttachmentProperty(*Content_ID*, *Property_Type*, *Value*)

Description

Use the SetAttachmentProperty to specify the value and type of a property associated with an attachment.

Parameters

Parameter	Description
Content_ID	Specify the content ID associated with the attachment.
Property_Type	Specify the type of property. Valid values are:

Value	Description
%Attachment_Encoding	Specify the encoding type.
%Attachment_Type	Specify the attachment type.
%Attachment_URL	Specifies the URL for the attachment. The URL must be an absolute URL.
%Attachment_Base	Specify the attachment base.
%Attachment_Location	Specify an additional location for the attachment.
%Attachment_Disposition	Specifies the disposition of the attachment, whether it's displayed inline or as an attachment.
%Attachment_Description	Specifies the description of the attachment.

Parameter	Description
Value	Specify the value associated with this property.

Returns

A boolean value: true if the property is set successfully, false otherwise.

Example

The following example shows sending an attachment with an asynchronous message:

```
Local Message &MSG;
Local Rowset &FLIGHT_PROFILE;
Local String &Attachment_Id;
Local Boolean &Test;
Local IntBroker &IntBroker;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
QE_FLIGHTDATA.QE_ACNUMBER.Value = QE_FLIGHTDATA.QE_ACNUMBER + 1;

&FLIGHT_PROFILE = GetLevel0();

&MSG = CreateMessage(Operation.ASYNC_RR);

&Attachment_Id = &MSG.IBInfo. AddAttachment("C:\\temp\\MyFile.txt");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Encoding,⇒
 "UTF-8");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Base,⇒
 "Standard");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Disposition,⇒
 "Pending");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Language,⇒
 "English");
&Test = &MSG.IBInfo.SetAttachmentProperty(&Attachment_Id, %Attachment_Description,⇒
 "Parts data");

&MSG.CopyRowset(&FLIGHT_PROFILE);

&IntBroker = %IntBroker

&IntBroker.Publish(&MSG);
```

See Also

Chapter 22, “Message Classes,” GetAttachmentProperty, page 805

IBInfo Class Properties

In this section, we discuss the IBInfo class properties. The properties are discussed in alphabetical order.

AppServerDomain

Description

This property can be used to set the application server domain for the connector, as a string.

This property is read-write.

IBConnectorInfo

Description

This property returns a reference to a ConnectorInfo object.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

ConnectorOverride

Description

This property specifies whether the connector override is specified for the transaction. This property takes a Boolean value: true, override the connection properties, false otherwise.

This property is read-write.

ConversationID

Description

This property returns the conversation Id associated with the request/response message transaction.

This property is read-write.

Cookies

Description

Use this property to access the cookies associated with a message.

You can accept a synchronous response message containing cookies, save those cookies in a global variable, and later return them to the target node in an outbound synchronous or asynchronous request message.

You can access this property only in an inbound synchronous response message or an outbound request message.

This property is read-write.

Example

The following is an example of receiving cookies:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;

&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* send the synchronous request; the return value is */
/* the response message object */
&SalesResponse = %IntBroker.SyncRequest(&SalesRequest);

/* Retrieve cookies from the response message */
&SalesCookies = &SalesResponse.IBINfo.Cookies;
```

The following is an example of returning cookies:

```
Local Message &SalesRequest, &SalesResponse;
Local Rowset &SALES_ORDER;
Global string &SalesCookies;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&SALES_ORDER = GetLevel0();
&SalesRequest = CreateMessage(OPERATION.SALES_ORDER_SYNC);
&SalesRequest.CopyRowsetDelta(&SALES_ORDER);

/* Insert the cookies in the request message */
&SalesRequest.IBInfo.Cookies = &SalesCookies;

/* Send the asynchronous request */
%IntBroker.Publish(&SalesRequest);
```

DestinationNode

Description

This property returns the name of the destination node that the request was sent to, as a string.

This property is read-only.

ExternalMessageID

Description

This property returns the external ID of the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-only.

Example

```
&str = &MSG.IBInfo.ExternalMessageID;
```

ExternalOperationName

Description

This property returns the external operation name of the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-write.

ExternalUserName

Description

This property returns the external user name associated with the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

ExternalUserPassword

Description

This property returns the external user password associated with the message. This property is used in testing, and to resolve duplicate message issues from third-party systems.

This property is read-write.

FinalDestinationNode

Description

When the message is passed across several nodes, this property specifies the ultimate target of the message, as a string.

This property is read-only.

HTTPSessionId

Description

Use the HTTPSessionId property to specify the HTTP session Id, as a string.

This property is read-write.

InReplyToID

Description

Use the InReplyToID property to specify the reply to Id contained in the message.

This property is read-write.

MessageChannel

Description

This property references the name of the channel associated with the message definition, as a string.

Note. This property has been deprecated and remains for backward compatibility only. Use the IBInfo class MessageQueue property instead.

This property is set in when the message is created.

This property is read-only.

See Also

Chapter 22, “Message Classes,” MessageQueue, page 814

See Also

Chapter 22, “Message Classes,” ChannelName, page 769

Copyright © 1988-2007, Oracle. All rights reserved.

MessageName

Description

This property returns the name of the message, as a string.

This property is read-only.

See Also

Chapter 22, "Message Classes," Name, page 778

MessageQueue

Description

This property returns the name of the queue associated with the message, as a string.

This property is read-only.

MessageType

Description

This property returns the type of the message, as a string.

Note. This property has been deprecated and remains for backward compatibility only. Use the IBInfo class OperationType property instead.

See Chapter 22, "Message Classes," OperationType, page 815.

Valid types are:

Value	Description
Sync	Indicates that the message is synchronous.
Async	Indicates that the message is asynchronous.
Ping	Indicates that the message is used to test the application server to make sure it is available and accepting requests.

This property is read-only.

MessageVersion

Description

This property returns the message version as a number.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

NodeDN

Description

For incoming requests, this property gives the distinguished name (DN) extracted from the certificate authentication process, as a string.

This property is read-write.

NonRepudiationID

Description

This property returns the non-repudiation ID as a string. This property is populated with a unique string when the message is published.

This property is only valid with messages that use non-repudiation.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Setting Up Secure Integration Environments," Implementing Nonrepudiation

NumberOfAttachments

Description

This property returns the number of attachments associated with a message.

This property is read-only.

See Also

Chapter 22, "Message Classes," AddAttachment, page 802

OperationType

Description

This property returns the type of the operation, as a string.

This property is read-only.

OperationVersion

Description

This property returns the version of the operation, as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

OrigNode

Description

For requests that cross multiple nodes, this property identifies the node that initiated the request as a string.

The OrigNode property returns the originating node of the message. If the message is not going across nodes, the OrigNode and SoureNode properties return the same value. However, if the message is going across nodes, the source node is the node that most recently published the message.

For example, if A publishes to B, then B publishes the message to C, from C's perspective, A is the original node and B is the source node.

This property is read-only.

See Also

Chapter 22, "Message Classes," SourceNode, page 817

OrigProcess

Description

This property returns the name of the process where the publish originated, as a string. For example, a message published from the Inventory definitions page would have a process name of INVENTORY DEFIN.

This property is read-only.

OrigTimeStamp

Description

This property returns the time stamp that corresponds to the time that the request was created, as a string. For requests that cross nodes, this is the time that the first request was created.

This property is read-only.

OrigUser

Description

This property returns the user ID login where the message was initially generated, as a string.

This property is read-only.

PublicationID

Description

This property returns the unique identifier for the message, as a string.

Note. This property has been deprecated. It is no longer supported.

 Copyright © 1988-2007, Oracle. All rights reserved.

RequestingNodeName

Description

This property returns the name of the node making the request, as a string.

This property is read-write.

RequestingNodeDescription

Description

This property returns the description of the node making the request, as a string.

This property is read-write.

ResponseAsAttachment

Description

Use the ResponseAsAttachment property to specify whether the response should be returned as an attachment or inline.

This property is read-write.

SegmentsUnOrder

Description

The SegmentUnOrder property is only applicable for asynchronous messages. If you specify the SegmentUnOrder property as true, the receiving node processes the segments in parallel.

This property is read-write.

See Also

Chapter 22, “Message Classes,” SegmentsByDatabase, page 780

Chapter 22, “Message Classes,” Message Segments, page 735

SourceNode

Description

This property returns the name of the publishing node as a string.

The OrigNode property returns the originating node of the message. If the message is not going across nodes, the OrigNode and SoureNode properties return the same value. However, if the message is going across nodes, the source node is the node that most recently published the message.

For example, if A publishes to B, then B publishes the message to C, from C’s perspective, A is the original node and B is the source node.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 22, “Message Classes,” OrigNode, page 816

SyncServiceTimeout

Description

This property takes a time (in seconds). This time overrides the default http timeout that is used for all requests. If you set this property, you can use this to read back the time, that is, set the time before the SyncRequest is executed, then see when it is changed in an implementation of the OnSend method.

Note. This property is only for syncronous requests.

Generally, you use SyncServiceTimeout to dynamically set the timeout value for a specific transaction. The http header file is modified to take this new parameter. In addition this value is sent to the gateway to be used for the http timeout. Use this so that a long running transaction will not timeout. In addition, you don’t have to wait through the default period for a specific transaction to timeout.

The following is a typical example of using this property:

```
&MSG.SetXmlDoc((&xmlReq);
&MSG.IBInfo.LoadConnectorPropFromNode(Node.EAI)
&MSG.IBInfo.SyncServiceTimeout(360000);
&MSG.IBInfo.ConnectorOverride = true;
&MSG_Resp = SyncRequest(&MSG, Node.EAI);
&xmlResponseDoc = &MSG.GetXmlDoc();
```

Setting the XML directly is not valid. You need to use the message object to set the value. In order for this to work you must override the connector properties, which means you must set up the connector properties for this transaction, using one of the load methods (such as LoadConnectorPropFromNode, LoadConnectorPropFromTrx, and so on.)

This property is read-write.

TransactionID

Description

This property returns the transaction ID as a string. This is used to uniquely identify a request.

This property is read-only.

UserName

Description

This property returns the user name associated with the message, as a string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

VisitedNodes

Description

This property returns an array of string containing the names of all the nodes visited by the message. This is useful when a message is being propagated across multiple nodes.

This property is read-only.

WSA_MessageID

Description

Use the WSA_MessageID property to specify the web service addressing (WSA) message Id for the message.

This property is read-write.

WSA_ReplyTo

Description

Use the WSA_ReplyTo property to specify the web service addressing (WSA) reply to address for the message.

This property is read-write.

ConnectorInfo Collection

A ConnectorInfo collection object is returned from the IBConnectorInfo IBInfo class property.

See Also

Chapter 22, “Message Classes,” IBConnectorInfo, page 810

ConnectorInfo Collection Methods

In this section, we discuss the ConnectorInfo collection methods. The methods are discussed in alphabetical order.

AddConnectorProperties

Syntax

AddConnectorProperties(*Name*, *Value*, *Type*)

Description

Use the AddConnectorProperties method to add a set of connector properties to a connector.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of the property as a string.
Value	Specify the value of the property as a string.
Type	Specify the type of the property as a string. Values are:

Value	Description
%Property	Add a property type property.
%Header	Add a header type property.

Returns

A Boolean value: true if the connector properties are added successfully, false otherwise.

Example

The following are example of typical name/value pairs.

```
&MSG.IBInfo.IBConnectorInfo.AddConnectorProperties("URL", "http:⇒
//finance.yahoo.com/d/quotes.txt/?symbols=PSFT&format=l1c1d1t1", %Property);

&MSG.IBInfo.IBConnectorInfo.AddConnectorProperties("sendUncompressed", "Y",⇒
 %Header);

&MSG.IBInfo.IBConnectorInfo.AddConnectorProperties("FilePath", "C:\Temp",⇒
 %Property);
```

See Also

Chapter 22, “Message Classes,” DeleteConnectorProperties, page 822 and Chapter 22, “Message Classes,” ClearConnectorProperties, page 821.

AddQueryStringArg

Syntax

AddQueryStringArg(*Name*, *Value*)

Description

Use the AddQueryStringArg method to add query string arguments to the outbound request. The query string arguments are used by the HTTP connector to step parameters in the URL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of the query string argument as a string.
Value	Specify the value for the query string argument as a string.

Returns

A Boolean value: true if the query string argument is added successfully, false otherwise.

See Also

Chapter 22, "Message Classes," ClearQueryStringArgs, page 821 , Chapter 22, "Message Classes," DeleteQueryStringArg, page 822 , Chapter 22, "Message Classes," GetNumberOfQueryStringArgs, page 824 , Chapter 22, "Message Classes," GetQueryStringArgName, page 824 and Chapter 22, "Message Classes," GetQueryStringArgValue, page 825.

ClearConnectorProperties

Syntax

```
ClearConnectorProperties()
```

Description

Use the ClearConnectorProperties method to clear all the properties in a connector before setting them.

Parameters

None.

Returns

None.

See Also

Chapter 22, "Message Classes," AddConnectorProperties, page 819 and Chapter 22, "Message Classes," DeleteConnectorProperties, page 822.

ClearQueryStringArgs

Syntax

```
ClearQueryStringArgs()
```

Description

Use the ClearQueryStringArgs method to clear all of the existing query string arguments.

Use the DeleteQueryStringArg method if you want to remove a specific query string argument.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 22, “Message Classes,” DeleteQueryStringArg, page 822

DeleteConnectorProperties

Syntax

DeleteConectorProperties(*Name*)

Description

Use the DeleteConnectorProperties to delete a specific connector property.

Use the ClearConnectorProperties method to remove all the properties.

Parameters

Parameter	Description
Name	Specify the name of the connector property you want to delete.

Returns

A Boolean value: true if the property is deleted successfully, false otherwise.

See Also

Chapter 22, “Message Classes,” ClearConnectorProperties, page 821

DeleteQueryStringArg

Syntax

DeleteQueryStringArg(*Name*)

Description

Use the DeleteQueryStringArg method to delete a specific query string argument.

Use the ClearQueryStringArg method to delete all of the query string arguments.

Parameters

Parameter	Description
Name	Specify the name of the query string argument that you want to delete.

Returns

A Boolean value: true if the query string argument is deleted successfully, false otherwise.

See Also

Chapter 22, “Message Classes,” ClearQueryStringArgs, page 821

 Copyright © 1988-2007, Oracle. All rights reserved.

GetConnectorPropertiesName

Syntax

GetConnectorPropertiesName(*Index*)

Description

Use the GetConnectorPropertiesName to return the name of the connector property in the numeric position specified by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the connector property name that you want to access.

Returns

A string containing the name of a connector property.

Example

```
For &I = 1 to &Msg.IBInfo.IBConnectorInfo.GetNumberOfConnectorProperties();
  &PropName = &Msg.IBInfo.IBConnectorInfo.GetConnectorPropertiesName(&I)
  /* do processing */
End-For;
```

GetConnectorPropertiesType

Syntax

GetConnectorPropertiesType(*Index*)

Description

Use the GetConnectorPropertiesType method to return the type of the connector property specified by its numeric position by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the connector property type that you want to access.

Returns

A string containing the type of the specified connector.

GetConnectorPropertiesValue

Syntax

GetConnectorPropertiesValue(*Index*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetConnectorPropertiesValue method to return the value of the connector property specified by its numeric position by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the connector property type that you want to access.

Returns

A string containing the value of the specified connector.

GetNumberOfConnectorProperties

Syntax

```
GetNumberOfConnectorProperties()
```

Description

Use the GetNumberOfConnectorProperties method to determine the number of connector properties.

Parameters

None.

Returns

A number.

GetNumberOfQueryStringArgs

Syntax

```
GetNumberOfQueryStringArgs()
```

Description

Use the GetNumberOfQueryStringArgs method to determine the number of query string arguments.

Parameters

None.

Returns

A number.

GetQueryStringArgName

Syntax

```
GetQueryStringArgName(Index)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetQueryStringArgName method to access the name of the query string argument by its numeric position as specified by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the query string argument name that you want to access.

Returns

A string containing the name of a query string argument.

GetQueryStringArgValue

Syntax

```
GetQueryStringArgValue(Index)
```

Description

Use the GetQueryStringArgValue method to access the value of the query string argument by its numeric posistion as specified by *Index*.

Parameters

Parameter	Description
Index	Specify the numeric position of the query string argument value that you want to access.

Returns

A string containing the value of a query string argument.

ConnectorInfo Collection Properties

In this section, we discuss the ConnectorInfo collection properties. The properties are discussed in alphabetical order.

ConnectorClassName

Description

Use this property to identify the name of the target connector to invoke as a string.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

ConnectorName

Description

Use this property to identify the target connector to invoke to send to the message, as a string.

This property is read-write.

Cookies

Description

Use this property to either return or set the cookies associated with this connector. The cookies are specified as a comma separated string. This is the cookie string found when the request was received by the HTTPListeningConnector.

This property is read-write.

PathInfo

Description

This property is specific to incoming HTTP requests. This is the path information extracted from the request, represented as a string.

This property is read-write.

RemoteFrameworkURL

Description

Use this property to identify the URL to which to send a message, as a string. This value overrides the server URL specified in the Project Definitions section.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 23

Mobile Classes

This chapter provides an overview of the Mobile classes and discusses the following topics:

- Mobile classes overview
- Mobile terms
- Mobile hierarchy
- HTML areas used in mobile
- Debugging in mobile
- Error handling
- Data types for mobile
- Scope of a mobile object
- Mobile reference

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Mobile Agent

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Mobile Pages and Classes"

Chapter 39, "SyncServer Class," page 1585

Mobile Classes Overview

PeopleSoft Mobile Agent extends the functionality of PeopleSoft Pure Internet Architecture to disconnected mobile devices, allowing users to continue working with their PeopleSoft applications on a laptop computer or personal digital assistant (PDA) while disconnected from the Internet or a local network.

When you create mobile pages for display on a mobile device (a laptop or PDA), you create the following in PeopleSoft Application Designer:

- synchronizable Component Interfaces.
- mobile pages based on your synchronizable Component Interfaces.

In addition, you may create PeopleCode programs that manipulate the data. You use the Mobile Classes in a mobile application to work with the data on a mobile device.

This document provides an overview of using the Mobile Classes, including:

- Understanding the Mobile Device and the Component Interface
- Understanding differences between Mobile and PIA

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 39, “SyncServer Class,” page 1585

Understanding the Mobile Device and the Component Interface

When the synchronization server and the mobile device are synchronized, the following items are downloaded to the mobile device:

- instances of synchronizable Component Interfaces.
- mobile pages.
- associated mobile PeopleCode programs.

When a user of Mobile Agent synchronizes with the synchronization server the first time, the relevant data for the synchronizable Component Interface is loaded to the mobile device.

During subsequent synchronizations, the data on the mobile device is synchronized with the database on the synchronization server.

This section discusses the items that are downloaded to the mobile device. For more information about the synchronization process, see the Synchronization Server documentation.

See Chapter 39, “SyncServer Class,” page 1585.

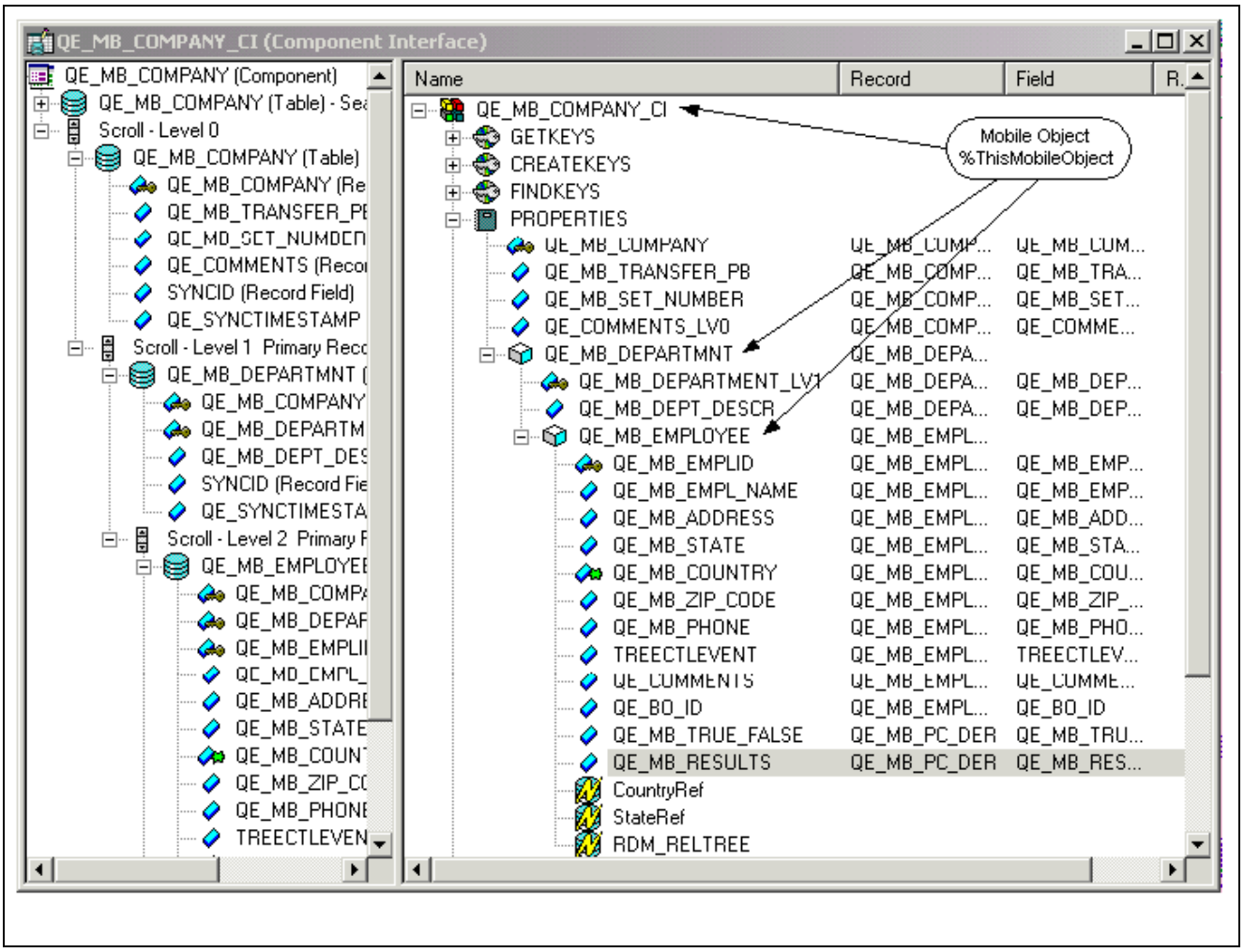
Synchronizable Component Interface

A synchronizable Component Interface is a Component Interface that is created specifically for use with a mobile device. This means that the Component Interface definition has been specified as synchronizable on the Component Interface properties page.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Mobile Agent*.

The following shows a synchronizable Component Interface.

 Copyright © 1988-2007, Oracle. All rights reserved.



Synchronizable Component Interface

Level zero of the synchronizable component interface, as well as the data in the collections, are known as *mobile objects,* and are described in the following sections.

Mobile Page

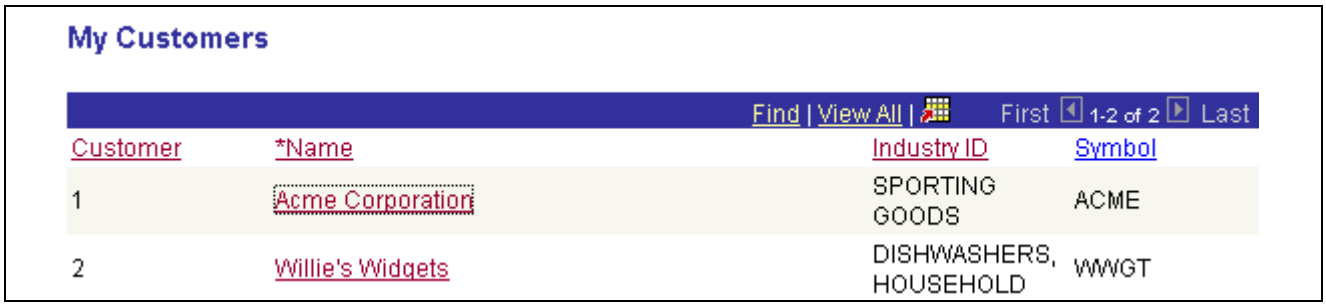
The Mobile Page enables a mobile device user to change data in the synchronizable Component Interface that underlies the Mobile Page. The changes to the Component Interface data are sent back to the PeopleSoft database when the mobile device is synchronized.

Mobile pages have *list views* and *detail views*.

- List view pages are created from the top level of a component interface (the level zero), or a collection in a component interface. A list view can also be thought of as a search page.
- Detail view pages are created from the content properties (fields) in a mobile page. Where and how a property displays on a page is determined by the content properties page for that property (field).

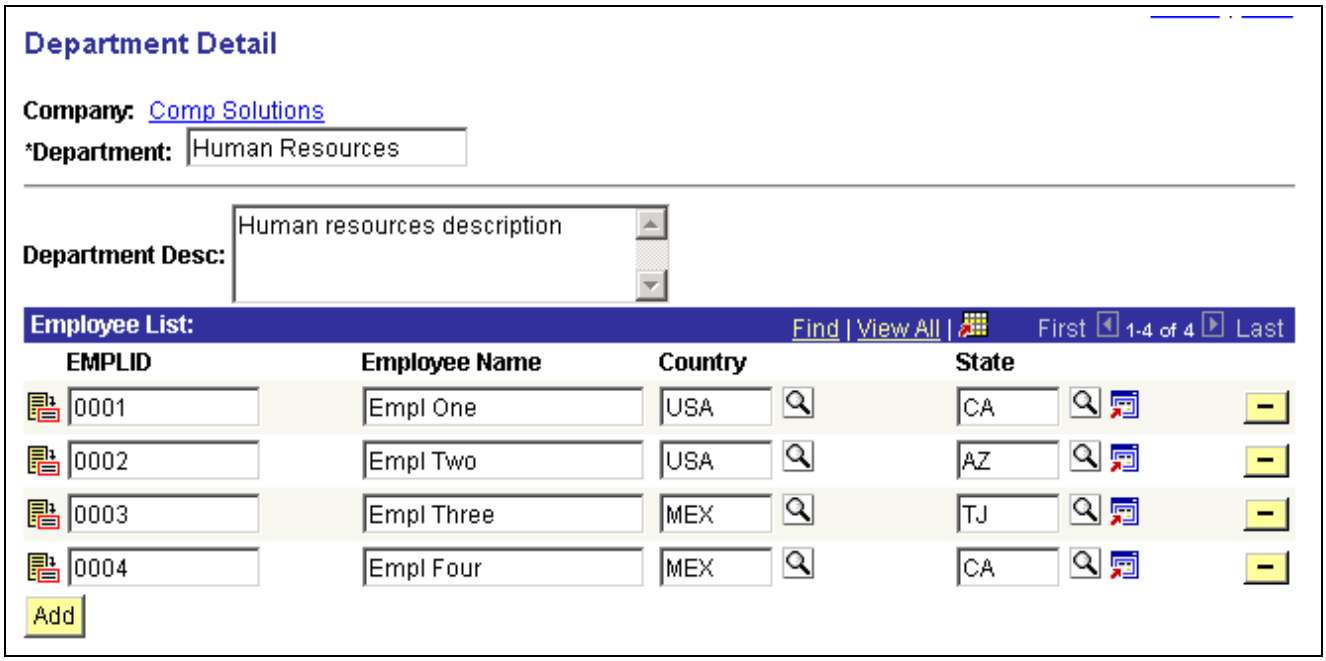
The following is the list view for a mobile page, generated by the level zero of the synchronizable component interface.

Copyright © 1988-2007, Oracle. All rights reserved.



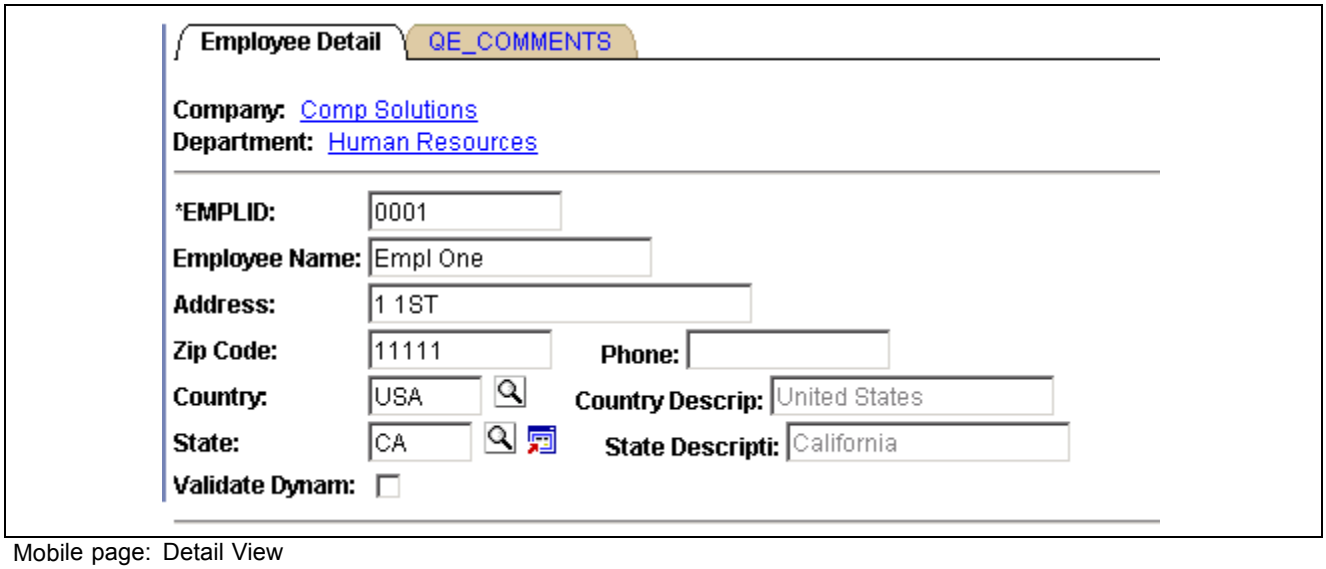
Mobile Page: level zero List View

The following is a list view generated by a collection in a synchronizable component interface.



Mobile page: Collection List View

The following is the detail view generated by the properties (fields) of the synchronizable component interface.



Mobile page: Detail View

Copyright © 1988-2007, Oracle. All rights reserved.

Associated Mobile PeopleCode Programs

Access PeopleCode associated with a Component Interface by right clicking on an open synchronizable Component Interface in PeopleSoft Application Designer and selecting ViewPeopleCode.

The following is a PeopleCode program associated with the OnChange Mobile Component Interface event on a synchronizable Component Interface. The OnChange event is associated with a property on the page. The PeopleCode program uses a *mobile object* to access the QE_MB_EMPLID property on level zero on the synchronizable Component Interface that has been downloaded to the mobile device. The mobile object represents the PIA equivalent of a row of data, and can be used at any level in the data hierarchy, level zero, level one, level two, and so on. It is what is displayed in the Detail View of a Mobile Page.

When a mobile event occurs on a mobile object, such as a mobile object property being changed (the OnChange event) or a mobile object being loaded into the working set (the OnInit event), the mobile object is then available in the event's PeopleCode program using the %ThisMobileObject system variable.

```
If %ThisMobileObject.IsNew() Then

/* do further check before save */

End-if;
```

Considerations Using the Mobile Object

A mobile object is the means in PeopleCode to access the synchronizable Component Interface. Mobile objects exist in relationship to each other in a hierarchy, which has a Component Buffer-like structure. A mobile object is just one row. A Data Collection is a collection of rows, each row being a mobile object.

The mobile object hierarchy has the equivalent of rowsets as data collections, rows as mobile objects, records and fields as properties. The topmost mobile object for a Component Interface is the equivalent of level zero on the underlying Component Interface. A Component Interface can also contain levels, which map to mobile object data collections. (These levels, in PIA, map to scrolls on a page, or in the Component Buffer, to rowsets.)

In the example in this section, the hierarchy of the mobile objects is:

```
QE_MB_COMPANY
    QE_MB_DEPARTMNT
        QE_MB_EMPLOYEE
```

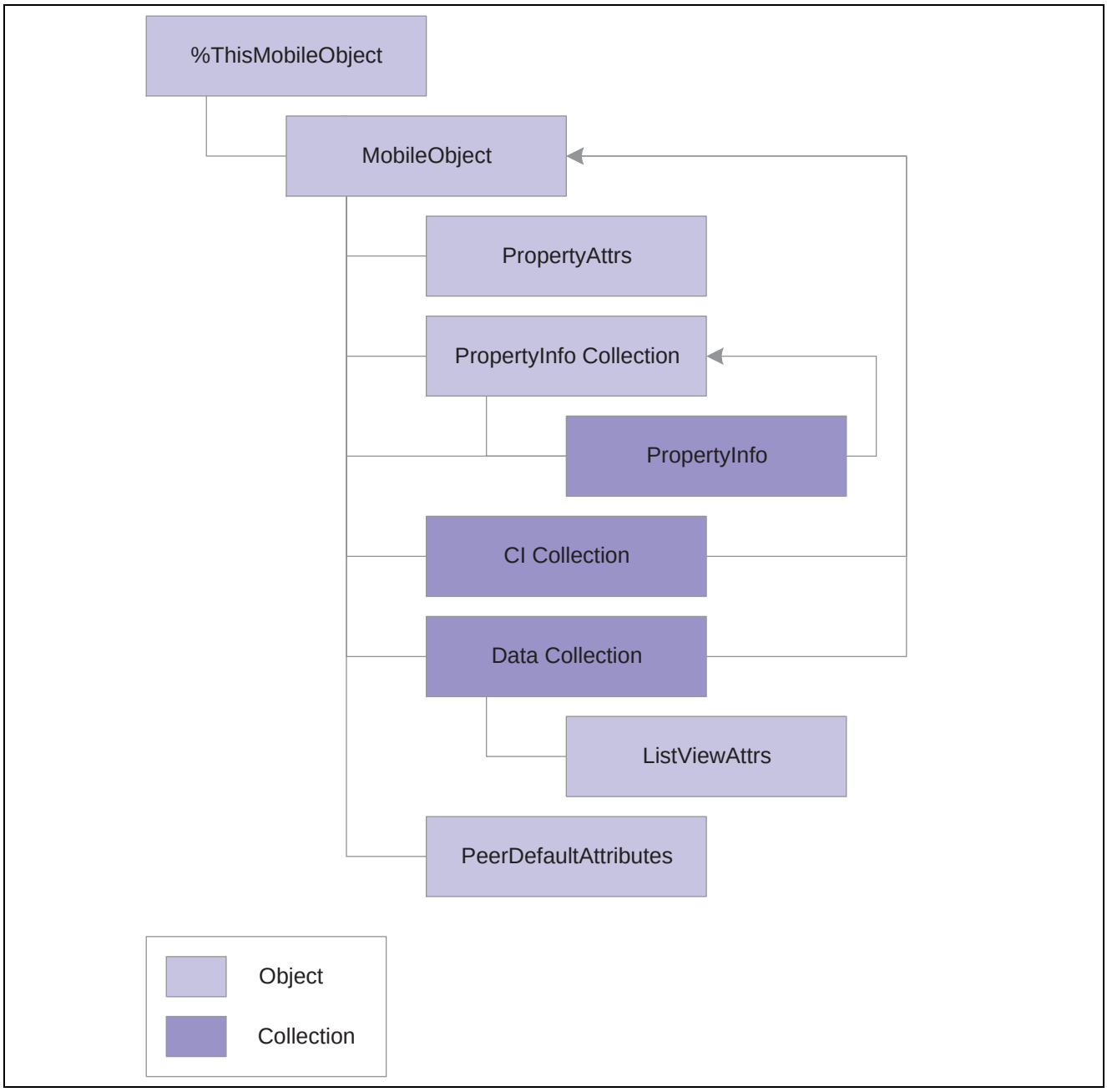
- QE_MB_COMPANY — level zero.
- QE_MB_DEPARTMNT — level one. This is a collection of mobile data objects.
- QE_MB_EMPLOYEE — level two. This is a collection of mobile data objects.

Mobile Class Hierarchy

Mobile objects exist in relationship to each other in a hierarchy. The following diagram shows the different classes, and how one class can be returned from a method or a property in another class.

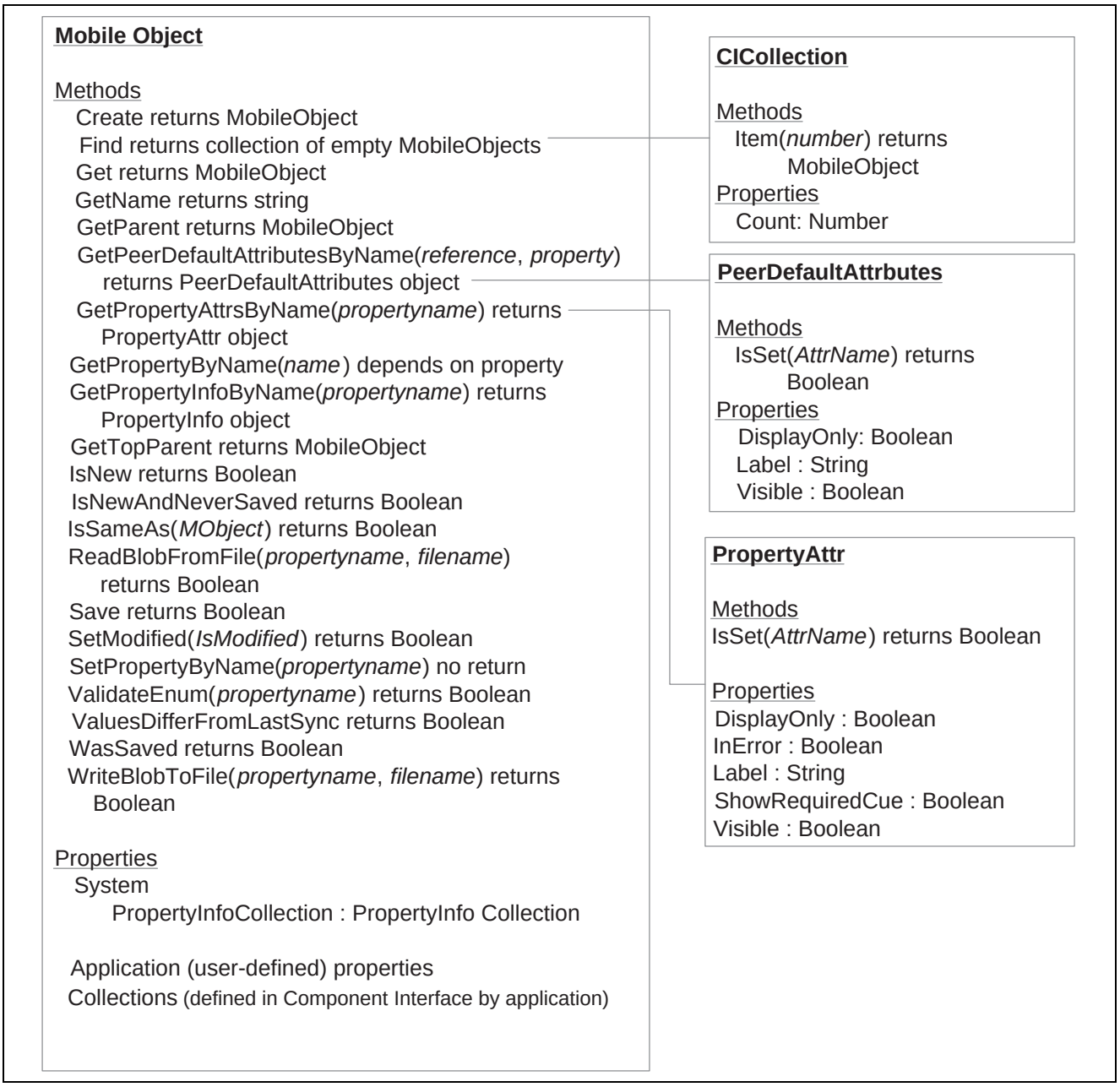
The following presents the mobile object hierarchy in a class diagram:

Copyright © 1988-2007, Oracle. All rights reserved.



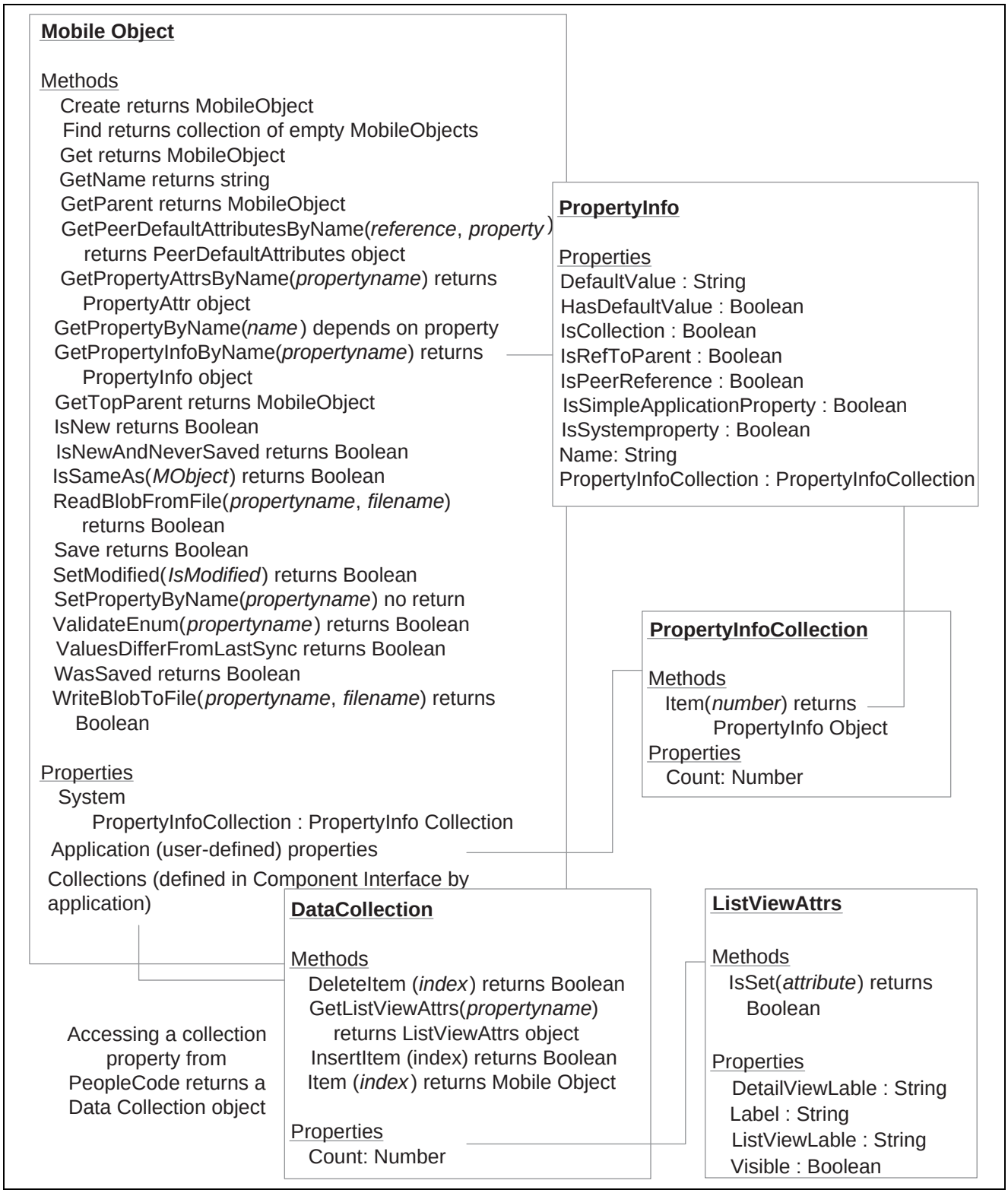
Mobile classes hierarchy

Copyright © 1988-2007, Oracle. All rights reserved.



Mobile Class hierarchy — class diagram 1 of 2

Copyright © 1988-2007, Oracle. All rights reserved.



Mobile class hierarchy — class diagram 2 of 2

The following is a list of the various classes, with a brief description of each.

Copyright © 1988-2007, Oracle. All rights reserved.

Mobile Class	Description
Mobile object	Represents the PIA equivalent of a row of data, and can be used at any level in the data hierarchy, level zero, level one, level two, and so on. It is what is displayed in the Detail View of a mobile page.
CI collection	Contains a collection of empty mobile objects.
Data collection	Used to access a level on a mobile page, such as level one, level two, and so on.
ListViewAttrs	Used to access the display attributes for a list view.
PeerDefaultAttributes	Used to access the default display attributes for a peer reference.
PropertyAttrs	Used to access the display attributes for a property.
PropertyInfo	Used to access the attributes of a property (field) on a mobile page.
PropertyInfoCollection	Contains a collection of PropertyInfo objects. Used when a property on a mobile page is actually a collection (such as a reference to level one, level two, and so on.)

Debugging in Mobile

The psmobile.ini file contains debug settings for mobile PeopleCode.

For a laptop, you can edit the psmobile.ini file on the mobile device. This file is either in the C:\Windows or C:\WinNT directory.

For a PDA, you must copy the psmobile.ini file to the laptop or desktop using ActiveSync, edit it there, then copy it back.

Use the DebugEnabled value in the psmobile.ini file to control whether debugging is enabled for your application. If you specify a 1 for this value, you can use the PeopleCode debugger with your mobile application for debugging.

In addition, you can use the TracePC value in the psmobile.ini file to specify that a trace is to be run, as well as what type of operations in the application should be traced.

The following is an example of the psmobile.ini file.

```
[PSMOBILE]
Language=ENG
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Port=8080
HomeDirectory=C:\Program Files\PeopleSoft\PeopleSoft Mobile Agent
MaxCachedStatements=100
MaxCachedObjects=100
debug=1
[SYNC]
SyncGateway=http://fsampson040502.corp.peoplesoft.com:80/SyncServer
SyncLogVerbose=0
[PCDEBUG]
DebugEnabled=0
DebugDatabase=Your Database Name
DebugOprid=Your OPRID
DebugIPAddress=nnn.nnn.nnn.nnn
DebugPort=9500
[Trace]
;-------------------------------------------------------------
; PeopleCode Tracing Bitfield
;
; Bit       Type of tracing
; ---       ---------------
; 1         - Trace instructions
; 2         - List the program
; 4         - Show assignments to variables
; 8         - Show fetched values
; 16        - Show stack
; 64        - Trace start of programs
; 128       - Trace external function calls
; 256       - Trace internal function calls
; 512       - Show parameter values
; 1024      - Show function return value
; 2048      - Trace each statement in program
; Dynamic change allowed for TracePC and TracePCMask
TracePC=0
```

Debug Settings

The following are the debug settings you can set in the psmobile.ini file.

DebugEnabled Specify whether debugging is enabled or not. Use a 0 to turn off debug mode, a 1 to turn on debug mode.

DebugDatabase Specify the actual database (not the domain) that application server uses with the PeopleTools PeopleCode debugger (this should be the database that is used for synchronizing).

DebugOprid Specify the userId you use to log onto the PeopleTools PeopleCode debugger.

DebugIPAddress Specify the IP address or machine name of the application server.

DebugPort Specify the port number that the application server (PSDBGSRV) has been configured to use.

 Copyright © 1988-2007, Oracle. All rights reserved.

Trace Settings

Use the TracePC value in the psmobile.ini file to specify that a trace is to be run, as well as what type of operations in the application should be traced.

All trace information is written to a file named `psmobile.log`, located in the directory defined by `%temp%`.

Note. Be aware that tracing logs can grow to a large size. Only specify tracing when necessary, and only trace critical operations.

The following are the trace settings you can specify using TracePC. You can specify more than one trace by adding the numbers together.

0	Turn all tracing off.
1	Trace all instructions.
2	List the program being traced.
4	Show all assignments to all variables.
8	Show all fetched values.
16	Show stack.
64	Trace start of programs.
128	Trace external function calls.
256	Trace internal function calls.
512	Show all parameter values.
1024	Show function return value.
2048	Trace each statement in program.

Error Handling

For initial error handling, check the return values of methods and functions as they are used. When you want to check for errors depends on your application. However, if you check for errors after every assignment, you may see a performance degradation.

Mobile does not use collections of errors or messages (PSMESSAGES), as is the case with PeopleSoft Pure Internet Architecture. However, you can specify a message set and number for use with a mobile page and with mobile properties. This is done in PeopleSoft Application Designer, to ensure that the correct messages get downloaded to the mobile device.

Once the messages have been distributed to the mobile device, you can access them using PeopleCode the same way you would access any message in the message catalog, using the following functions:

- MessageBox
- MsgGet
- MsgGetText

Copyright © 1988-2007, Oracle. All rights reserved.

- MsgGetExplainText

In the following example, the program checks for an invalid value for a field, and issues a message if the value is incorrect. This code would be in an OnChangeEdit event associated with a property.

```
&Value = %ThisMobileObject.EmailID;

If Not(Find("@", &Value)) Then
     /* invalid email address probably */
     MessageBox(0, "", 525, 2, "Message Not Found");

     &PropAttrs = %ThisMobileProperty.GetPropertyAttrsByName("EmailID");
     &PropAttrs.InError = True

End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Mobile Agent, "Using Synchronizable Component Interfaces," Creating Synchronizable Component Interfaces

Data Types for Mobile

All mobile objects are declared as type ApiObject. For example:

```
Local ApiObject &MyMobileCollection;
Local ApiObject &MyMobileObject = %ThisMobileObject;
```

Note. Mobile objects can be declared only as Local.

Scope of Mobile Objects

A mobile object can be instantiated from PeopleCode. In addition, it can only be instantiated within the mobile events.

Variables in Mobile PeopleCode

Local variables are the only type of PeopleCode variables supported in mobile.

Mobile Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GenerateMobileTree

 Copyright © 1988-2007, Oracle. All rights reserved.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransferMobilePage

Session Class Method

The following is the Session class method used with Mobile.

See Also

Chapter 36, "Session Class," page 1511

GetCompIntfc

Syntax

GetCompIntfc([**COMPINTFC.**]*ComponentName*)

Description

The GetCompIntfc method returns a reference to an empty mobile object. Mobile objects are the means of accessing the synchronizable Component Interface in PeopleCode, so this is similar to creating an empty Component Interface in PIA.*ComponentName*, when used by itself, takes a string value. If you use COMPINTFC.*ComponentName* it is not a string value; it is a constant that automatically is renamed in your code if the Component Interface definition is ever renamed.

Parameters

Parameter	Description
ComponentName	Specify the name of the synchronizable Component Interface. You can either use *ComponentName* by itself or prefaced with the keyword **COMPINTFC**.

Returns

This method returns an empty mobile object if successful, or Null on error (such as *ComponentName* specifies a nonexistent synchronizable Component Interface).

Example

```
Local ApiObject &SaveCustomer, &Customer;

&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;

&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";

if (&Customer.Get() <> true) then;
   if (&Customer.Create() <> true) then;
      ErrorHandler();
   Else
      &Customer.Save();
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   End-If;
End-if;

%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

Mobile Class

The mobile object represents the PIA equivalent of a row of data. It is what is displayed in the Detail View of a Mobile Page.

The mobile object is accessed using the %ThisMobileObject system variable.

Mobile Class Methods

The following are the mobile class methods, in alphabetical order.

Note. With the mobile class methods, and unlike the Component Interface class methods, all properties that have been set are used in the search. This provides more flexibility, and does not require setting up specific Get, Create, or Find keys.

Create

Syntax

Create()

Description

The Create method associates the mobile object with a new, open mobile object. It is available only on an empty mobile object that is returned by the GetCompIntfc session method.

Parameters

None.

Returns

A Boolean value: True if object was successfully created, False otherwise.

Example

```
Local ApiObject &SaveCustomer, &Customer;

&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;

&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";

if (&Customer.Get() <> true) then;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
    if (&Customer.Create() <> true) then;
      ErrorHandler();
    Else
      &Customer.Save();
    End-If;
End-if;

%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

See Also

Chapter 23, "Mobile Classes," GetCompIntfc, page 839

Find

Syntax

```
Find()
```

Description

The Find method returns a reference to a CI collection that is populated with a collection of empty mobile objects. The properties that the application developer sets within the empty mobile object are used as the Find keys. This method is available only on an empty mobile object that is returned by the GetCompIntfc session method. Using it on a non-empty mobile object causes an error.

This collection may have zero elements in it if no items matching the key values you set were found. If you set no keys, all of the mobile objects of the type specified by GetCompIntfc are returned.

While the PIA Component Interface Find method limits the number of returned instances to 300, the mobile Find method has no limit.

The Find method returns only the topmost parent mobile objects. You do not have to set values for all the key values. After you have a collection, you can use the CI Collection method Item to return the empty mobile object, and then use Get on the mobile object to instantiate it.

Note. Wildcards are not implemented for this release.

Parameters

None.

Returns

A CICollection collection of mobile objects. You must use the Get method on these mobile objects before you can use them.

Example

```
/* Error checking left out for clarity */

&MyCI = %Session.GetCompIntfc(CompIntfc.QE_MB_COMPANY_CI);
&MyCI.QE_MB_COMPANY = "Acme Appliances";
&MyCIFindCollection = &MyCI.Find();
&Acme = &MyCIFindCollection.Item(1);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&Acme.Get();
```

Get

Syntax

Get()

Description

The Get method associates a mobile object with the user-defined properties that have been explicitly set on the object.

Parameters

None.

Returns

A Boolean value: True if mobile object was successfully retrieved, False otherwise.

Example

```
Local ApiObject &SaveCustomer, &Customer;

&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;

&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";

if (&Customer.Get() <> true) then;
    if (&Customer.Create() <> true) then;
      ErrorHandler();
  Else
      &Customer.Save();
  End-If;
End-if;

%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

GetName

Syntax

GetName()

Description

The GetName method returns the class name of the mobile object.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

The class name of the mobile object as a string. This is the name of the synchronizable Component Interface, or one of the levels.

Example

```
&GetName = &qe_mb_obj_setup.GetName();

/*   Write the name of the Object to the Results field */
&qe_mb_obj.QE_MB_RESULTS = "The name of this Mobile Object is: " | &GetName;
```

GetParent

Syntax

GetParent()

Description

The GetParent method returns a mobile object that is the parent of the Mobile object that is executing the method. If you are at the top level (level zero), it returns itself.

Parameters

None.

Returns

The parent mobile object.

Example

```
/* This loop will continue getting parent level, until level 0 is reached.  */
&obj = %ThisMobileObject;
Repeat
    &obj = &obj.getparent();
    /* Do some processing */
until (&obj.IsSameAs(%ThisMobileObject.gettopparent()));
```

GetPeerDefaultAttributesByName

Syntax

GetPeerDefaultAttributesByName(*Reference*, *Property*)

Description

When a peer reference does not have a value associated with it, the display attributes can only be accessed using the PeerDefaultAttributes class. After a peer reference has values, it can be accessed using the ListViewAttrs class.

Use the GetPeerDefaultAttributesByName method to access the default peer reference attributes for the specified property on the peer reference.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Reference	Specify the name of a peer reference, as a string.
Property	Specify a property on the peer reference, as a string.

Returns

A reference to a PeerDefaultAttributes class.

Example

```
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").Visib⇒
le = True;
```

See Also

Chapter 23, “Mobile Classes,” PeerDefaultAttributes Class, page 873

GetPropertyAttrsByName

Syntax

GetPropertyAttrsByName(*propertyname*)

Description

The GetPropertyAttrsByName method returns a reference to a PropertyAttrs object, used to modify display attributes of properties.

Parameters

Parameter	Description
propertyname	The name of the property.

None.

Returns

A PropertyAttrs object, used to modify display attributes of properties.

Example

```
%ThisMobileObject.GetPropertyAttrsByName("QE_MB_SET_DATE").Visible = False;
```

See Also

Chapter 23, “Mobile Classes,” PropertyAttrs Class, page 869

 Copyright © 1988-2007, Oracle. All rights reserved.

GetPropertyByName

Syntax

GetPropertyByName(*Name*)

Description

The GetPropertyByName method returns the value of a property. Generally this function is used only in applications that cannot get the names of the mobile object user-defined properties until runtime.

Parameters

Parameter	Description
Name	Specify the name of the property you want to access. The PeopleCode program terminates if this property does not exist.

Returns

The value of the property specified by *Name*. For a collection, it returns a reference to the collection.

Example

```
&GetPropertyByName_Char = &qe_mb_obj_setup.GetPropertyByName(&CharacterProperty⇒
Name);
/*Write the results of the GetPropertyByName method to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The value of the Character Field: "
| &GetPropertyByName_Char | "
```

GetPropertyInfoByName

Syntax

GetPropertyInfoByName(*propertyname*)

Description

The GetPropertyInfoByName method returns a reference to a PropertyInfo object.

Parameters

Parameter	Description
propertyname	The name of the property.

Returns

A PropertyInfo object.

Example

```
&pinfo = %ThisMobileObject.GetPropertyInfoByName("QE_MB_SET_DATE");
if &pinfo.HasDefaultValue = True then
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   %ThisMobileObject.QE_MB_SET_DATE = &pinfo.DefaultValue;
end-if;
```

See Also

Chapter 23, "Mobile Classes," PropertyInfo Class, page 856

GetTopParent

Syntax

GetTopParent()

Description

The GetTopParent method returns a mobile object that is the topmost parent of the Mobile object.

Parameters

None.

Returns

A reference to the topmost parent mobile object as a mobile object. If the current mobile object is the topmost object, this method returns a reference to the current mobile object.

Example

```
/* Check if the created Mobile Object is at Level 0  */
If (&qe_mb_obj_setup.IsSameAs(&qe_mb_obj_setup.GetTopParent())) Then

   /* Set the Parent Level variable to 0.
   Results for this get set below the looping statement  */
   &Parent_Level = 0;

End-If;
```

IsNew

Syntax

IsNew()

Description

The IsNew method indicates if this object has never been synchronized.

Parameters

None.

Returns

A Boolean value: True if this object was created on the device and never synchronized, False if this object has been synchronized.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&IsNew = &qe_mb_obj_setup.IsNew();

/*Write the results of the IsNew to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &IsNew | ": The Mobile Object is⇒
 new, and has never been synced.";
```

IsNewAndNeverSaved

Syntax

IsNewAndNeverSaved()

Description

The IsNewAndNeverSaved method indicates if this object was created on the device and has not yet been saved.

Parameters

None.

Returns

A Boolean value: True if this object was created on the device and has not yet been saved, False otherwise.

Example

```
&IsNewAndNeverSaved = &qe_mb_obj_setup.IsNewAndNeverSaved();

/*Write the results of the IsNewAndNeverSaved method to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &IsNewAndNeverSaved | ": This⇒
 Mobile Object is new, and has never been saved. "
```

IsSameAs

Syntax

IsSameAs(*MObject*)

Description

The IsSameAs method determines if one mobile object is the same as another. This is useful when you are trying to determine if the %ThisMobileObject is the topmost mobile object.

Parameters

Parameter	Description
MObject	Specify the mobile object to which this mobile object is being compared.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if this mobile object is the same as the mobile object, False otherwise.

Example

```
/*Check if the created Mobile Object is at Level 0  */
If (&qe_mb_obj_setup.IsSameAs(&qe_mb_obj_setup.GetTopParent())) Then

/* Set the Parent Level variable to 0. Results for this get set below
the looping statement  */
&Parent_Level = 0;

EndIf;
```

ReadBlobFromFile

Syntax

ReadBlobFromFile(*blobpropertyname*, *filename*)

Description

The ReadBlobFromFile method reads a file on your mobile device and writes it into the attachment property. The attachment property is a reference property on the synchronizable Component Interface; its purpose is to contain attachment data.

When the mobile user synchronizes from the mobile page, the attachment file data contained in the property is sent to the synchronization server. The size of the attachment on the synchronization server is updated to reflect the new contents.

If you use this method, your synchronized Component Interface must use an attachment: the synchronization server must use its attachment events, and must run the attachment method PutAttachment.

See Chapter 39, "SyncServer Class," Using Attachment Events, page 1593.

Working With Relative Paths

You can specify either an absolute or a relative path for *filename* .

If you specify a relative path, the path that is appended is the working directory for the PSMobile.exe file. The exact path depends on the installation of Mobile on the device.

For example, if you specified `\files\MyFile.ext` for *filename*, the actual path might be something like `c:\Program Files\PeopleSoft\PeopleSoft Mobile Agent\files\Myfile.ext`.

Parameters

Parameter	Description
blobpropertyname	A string specifying the name of the reference property used for the attachment.
FileName	This parameter is a string specifying the path and filename of the file that you use as an attachment.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

This is a Boolean value: True if the attachment was read successfully; False otherwise.

Example

```
%ThisMobileObject.ReadBlobFromFile("Attachment", %ThisMobileObject.QE_MB_READ_⇒
WRITE | %ThisMobileObject.ATTACHUSERFILE);
```

Save

Syntax

```
Save()
```

Description

The Save method saves the entire mobile object hierarchy in which the mobile object is contained.

You do not generally call the Save method on %ThisMobileObject; that saving is done when the user clicks on the Save button.

Using this method does not save associated peer references; you must use the Save method on each peer references separately.

The standard PeopleSoft save business rules (that is, any PeopleCode programs associated with OnSaveEdit, OnSavePreChange, and so on) are executed after you execute this method.

If you instantiate another mobile object with PeopleCode (using Get, Create) these objects are not automatically saved when the end user clicks the Save button. You must use the Save method with these instances separately.

Using the Save method on developer-instantiated mobile objects does not guarantee they will be saved. The save does not take effect until the end user clicks the Save button.

If you make changes to a developer-instantiated mobile object, the end user is not automatically prompted to save when trying to leave the page because it is not part of the current component.

Note. If a developer uses the Save method with a developer-instantiated component interface, and if the end user did not make any other changes to data in the current component interface, when the user leaves the current unit of work, they are *not* prompted to save their data.

To ensure that the data actually gets committed to the database, the developer can force a prompt to occur. This can be done using the following code:

```
%ThisMobileObject.SetModified(True);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding Mobile Pages and Classes," Mobile Component Transaction Model

Parameters

None.

Returns

A Boolean value: True if component was successfully saved, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local ApiObject &SaveCustomer, &Customer;
&SaveCustomer = %ThisMobileObject.MyFavoriteCustomer;
&Customer = %Session.GetCompIntfc(COMPINTFC.CUSTOMER);
&Customer.Name="Acme Applicances";
if (&Customer.Get() <> true) then;
    if (&Customer.Create() <> true) then;
      ErrorHandler();
  Else
      %ThisMobileObject.SetModified(True)
      &Customer.Save();
End-If;
End-if;
%ThisMobileObject.MyFavoriteCustomer = &Customer;
```

SetModified

Syntax

SetModified(*IsModified*)

Description

The SetModified method forces the mobile object to have the specified modified value of True or False.

Note. Not only does using this method change the overall modified state of the object, but when changing the modified state to False it also clears the modified flag on any properties that had been modified.

Parameters

Parameter	Description
IsModified	Specify whether this mobile object has been modified. This parameter takes a Boolean value: True, the mobile object has been modified, False, the object has not been modified.

Returns

The current setting of this mobile object *before* being modified by this method: True for modified before this setting, False for not modified before this setting.

Example

```
/*   Get the value of the TrueFalse check box, to see how to set
the modified setting.  */
If &qe_mb_obj.QE_MB_TRUE_FALSE = True Then
   &True_or_False = True;
   &SetModified = &qe_mb_obj_setup.SetModified(True);
Else
   &True_or_False = False;
   &SetModified = &qe_mb_obj_setup.SetModified( False);
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

SetPropertyByName

Syntax

SetPropertyByName(*propertyname*, *value*)

Description

The SetPropertyByName method sets the value of a property. Generally this function is used only in applications that cannot set the names of the component interface properties until runtime.

Parameters

Parameter	Description
propertyname	The name of the property.
value	This is the value to which the property is set.

Returns

None. The PeopleCode program terminates if there is an error.

Example

```
%ThisMobileObject.SetPropertyByName("QE_MB_NUMBER", 77777);
```

ValidateEnum

Syntax

ValidateEnum(*propertyname*)

Description

The ValidateEnum method verifies if the value of a property is a valid enumeration value. Enumeration values are originally set in PeopleSoft Application Designer, when the original records for the synchronizable Component Interface are designed.

Parameters

Parameter	Description
propertyname	Specify the name of the property as a string.

Returns

A Boolean value: True if the value is a valid dynamic enumeration, False otherwise.

Example

```
/* Validate if the value entered into Results field is a valid dynamic enumeration⇒
 */
&ob = %ThisMobileObject;

&Validate_DE = &ob.ValidateEnum(&ob.QE_MB_RESULTS);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/*Enter Results into Results Field  */
&ob.QE_MB_RESULTS = "The following statement is " | &Validate_DE | ": The Property⇒
 " | &ob.QE_MB_RESULTS | " is a valid Dynamic Enumeration.";
```

ValuesDifferFromLastSync

Syntax

ValuesDifferFromLastSync()

Description

The ValuesDifferFromLastSync method indicates if at least one property has a value that differs from its value since the last synchronization.

If this object has been created on the mobile device, that is, when there is no previous sync value, the first saved values are used as the comparison values for this property.

This is a true member by member comparison of data values, not modified flags. The SetModified method has no effect on this method.

Parameters

None.

Returns

A Boolean value: True if at least one property in this object has a value that differs from its original value, False otherwise.

Example

```
&ValuesDifferFromLastSync = &qe_mb_obj_setup.ValuesDifferFromLastSync();

/*Write the results of the ValuesDifferFromLast Sync method to the results Field*/

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &ValuesDifferFromLastSync | ": At⇒
 least one property has a value that differs from the values last synced."
```

WasSaved

Syntax

WasSaved()

Description

The WasSaved method indicates if this object has been saved since it was created or last synchronized.

Parameters

None.

Returns

A Boolean value: True if this object has been saved since it was created or last synchronized, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&WasSaved = &qe_mb_obj_setup.WasSaved();

/*Write the results of the WasSaved method to the results Field   */

&qe_mb_obj.QE_MB_RESULTS = "The following is " | &WasSaved | ": This Mobile Object⇒
 has been saved since it was last synced, or created. "
```

WriteBlobToFile

Syntax

WriteBlobToFile(*blobpropertyname*, *filename*)

Description

The WriteBlobToFile method reads the attachment property and writes it into a file on your mobile device. The attachment property is a reference property on the synchronizable Component Interface; its purpose is to contain attachment data.

When the mobile user synchronizes from the mobile page, this attachment file data contained in the property is received from the synchronization server.

If you use this method, your synchronized Component Interface must use an attachment: the synchronization server must use its attachment events, and must run the attachment method GetAttachment.

See Chapter 39, “SyncServer Class,” Using Attachment Events, page 1593.

Working With Relative Paths

You can specify either an absolute or a relative path for *filename* .

If you specify a relative path, the path that is appended is the working directory for the PSMobile.exe file. The exact path depends on the installation of Mobile on the device.

For example, if you specified `\files\MyFile.ext` for *filename*, the actual path might be something like `c:\Program Files\PeopleSoft\PeopleSoft Mobile Agent\files\Myfile.ext`.

Parameters

Parameter	Description
blobpropertyname	A string specifying the name of the reference property that is used for the attachment.
FileName	A string specifying the path and filename of the attachment file. This parameter takes a string value.

Returns

This is a Boolean value: True if the attachment was written successfully; False otherwise.

Example

```
%ThisMobileObject.WriteBlobToFile("Attachment", %ThisMobileObject.QE_MB_READ_WRITE⇒
 | %ThisMobileObject.ATTACHUSERFILE);
```

Copyright © 1988-2007, Oracle. All rights reserved.

Mobile Class Property

The following is the Mobile Class property.

Note. Normally, properties for mobile objects are the equivalent of the user-define properties on the synchronizable Component Interface. The properties described here should not be used for user-defined properties when one designs a synchronizable Component Interface in the PeopleSoft Application Designer.

PropertyInfoCollection

Description

The PropertyInfoCollection property returns a reference to a collection of PropertyInfo objects that describe the shape of the mobile object.

This property is read-only.

See Also

Chapter 23, "Mobile Classes," PropertyInfo Class, page 856

PropertyInfoCollection

The PropertyInfoCollection objects contains a collection of PropertyInfo objects.

A PropertyInfoCollection is returned from the following:

- a mobile object
- a PropertyInfo object

See Also

Chapter 23, "Mobile Classes," PropertyInfoCollection, page 854

Chapter 23, "Mobile Classes," PropertyInfoCollection, page 859

PropertyInfoCollection Method

The following is the method for the PropertyInfoCollection property.

Item

Syntax

```
Item(number)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Item method returns a PropertyInfo object that exists at the *number* position in the PropertyInfoCollection.

Parameters

Parameter	Description
number	A position in the PropertyInfoCollection collection. The first position is 1.

Returns

A reference to the PropertyInfo object that exists at the *number* position in the PropertyInfoCollection collection. If the specified item is a collection, the returned PropertyInfo object is invalid and is set to NULL.

Example

```
/* Copy all like properties from &Item to &TargetItem */
for &i = 1 to &Item.PropertyInfoCollection.Count
   if &Item.PropertyInfoCollection.Item(&i).IsSimpleApplicationProperty then
     try &TargetItem.SetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name,
        &Item.GetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name))
     catch Exception &xException;

     end-try;
   end-if;
end-for;
```

PropertyInfoCollection Property

The following is the property for the PropertyInfoCollection property.

Count

Description

This property returns the number of PropertyInfo objects in the PropertyInfoCollection .

This property is read-only.

Example

```
/* Copy all like properties from &Item to &TargetItem */
for &i = 1 to &Item.PropertyInfoCollection.Count
   if &Item.PropertyInfoCollection.Item(&i).IsSimpleApplicationProperty then
    try &TargetItem.SetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name,
       &Item.GetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name))
       catch Exception &xException;
     end-try;
   end-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
end-for;
```

PropertyInfo Class

The PropertyInfo class objects are returned by the following:

- the Item method of a PropertyInfo collection
- the GetPropertyInfoByName of a mobile object

PropertyInfo Class Properties

The following are the properties for the PropertyInfo object.

DefaultValue

Description

This property returns the default value for this mobile object property.

This property is read-only.

Note. Definitional defaults are applied only when an Object is Created, just before the OnInit PeopleCode is run.

Example

```
&pinfo = %ThisMobileObject.GetPropertyInfoByName("QE_MB_SET_DATE");
if &pinfo.HasDefaultValue = True then
   %ThisMobileObject.QE_MB_SET_DATE = &pinfo.DefaultValue;
end-if;
```

HasDefaultValue

Description

This property returns a Boolean value: True if this mobile object property has a default value; False otherwise.

This property is read-only.

Note. Definitional defaults are applied only when an object is created, just before the OnInit PeopleCode is run.

Example

```
&pinfo = %ThisMobileObject.GetPropertyAttrsByName("Text");
if &pinfo.HasDefaultValue = True then;
   %ThisMobileObject.Text = &pinfo.DefaultValue;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
end-if;
```

IsCollection

Description

This property returns True if the property that the PropertyInfo object is describing is a Data Collection, False otherwise. If IsCollection is False, the object represents a mobile object user-defined or system property.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number field
to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The following is " | &PropertyInfo.IsCollecton | ":⇒
 This Property represents a collection";
```

IsRefToParent

Description

This property is True if this is a reference to this object's parent, False otherwise. A property that is RefToParent is also a System Property.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number
field to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/* Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The following is " | &PropertyInfo.IsRefToParent | ":⇒
 This is a reference to this Object' Parent. "
```

IsPeerReference

Description

This property is True if this is a peer reference, False otherwise.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

IsSimpleApplicationProperty

Description

This property is True is this is a simple, non-reference application-defined property; False otherwise. In other words, if True, it is not a system property, collection, nor peer reference; it is a property that was defined on the synchronized Component Interface in PeopleSoft Application Designer.

This property is read-only.

Example

```
/* Copy all like properties from &Item to &TargetItem */
for &i = 1 to &Item.PropertyInfoCollection.Count
   if &Item.PropertyInfoCollection.Item(&i).IsSimpleApplicationProperty then
      try &TargetItem.SetPropertyByName(&Item.PropertyInfoCollection.Item⇒
(&i).Name,
        &Item.GetPropertyByName(&Item.PropertyInfoCollection.Item(&i).Name))
      catch Exception &xException;

      end-try;
   end-if;
end-for;
```

IsSystemProperty

Description

This property is True if this is a System Property; False otherwise. System properties include the OID and VERSION properties, and references to Parents.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number
field to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The following is " | &PropertyInfo.IsSystemProperty |⇒
 ": This is a system property. "
```

Name

Description

The name of the property.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number
field to determine the item method from the PropertyInfoCollection class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field  */
&qe_mb_obj.QE_MB_RESULTS = "The name of the accessed Property is: " | &Property⇒
Info.Name;
```

PropertyInfoCollection

Description

This property returns a reference to another PropertyInfoCollection collection if the object executing the property is a collection.

This property is read-only.

Example

```
/* Use the PropertyInfoCollection Property of the mobile object class, and use the⇒
 Item number field to determine the item method from the PropertyInfoCollection⇒
 class. */
&PropertyInfo = &qe_mb_obj_setup.PropertyInfoCollection.item(&qe_mb_obj.QE_MB_PC_⇒
ITEM);

/*Write the results to the results field. Since this property returns a CompIntf⇒
PropertyInfoCollect Object, use the Count porperty of the PropertyInfoCollection⇒
 class, to test this.*/
&qe_mb_obj.QE_MB_RESULTS = "The property count of the CompIntfProperyInfo⇒
Collection returned by this property is: " | &PropertyInfo.PropertyInfoCollection⇒
.count;
```

CI Collection Class

A CI Collection is returned by the mobile object Find method. A CI Collection is a collection of empty mobile objects.

See Also

Chapter 23, “Mobile Classes,” Find, page 841

Copyright © 1988-2007, Oracle. All rights reserved.

CI Collection Method

The following is the CI Collection method.

Item

Syntax

Item(*index*)

Description

The Item method returns a Mobile Component Interface object. Once you've returned a reference, to instantiate it, use the Get method on the returned object.

Parameters

Parameter	Description
index	A number value: the index number for the Component Interface to fetch. Indexes start at 1.

Returns

An empty mobile object that was fetched. You must use the Get method to get the data for that object.

Example

```
/* Error checking left out for clarity */
&MyCI = %Session.GetCompIntfc(CompIntfc.QE_MB_COMPANY_CI);
&MyCI.QE_MB_COMPANY = "Acme Appliances";
&MyCIFindCollection = &MyCI.Find();
&Acme = &MyCIFindCollection.Item(1);
&Acme.Get();
```

CI Collection Properties

The following is the CI Collection property.

Count

Description

The number of Mobile Component Interface objects in the CI Collection collection. This property is read-only.

Example

```
/* Error checking left out for clarity */
&MyCI = %Session.GetCompIntfc(CompIntfc.QE_MB_COMPANY_CI);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&total_in_collection = &MyCI.count;
```

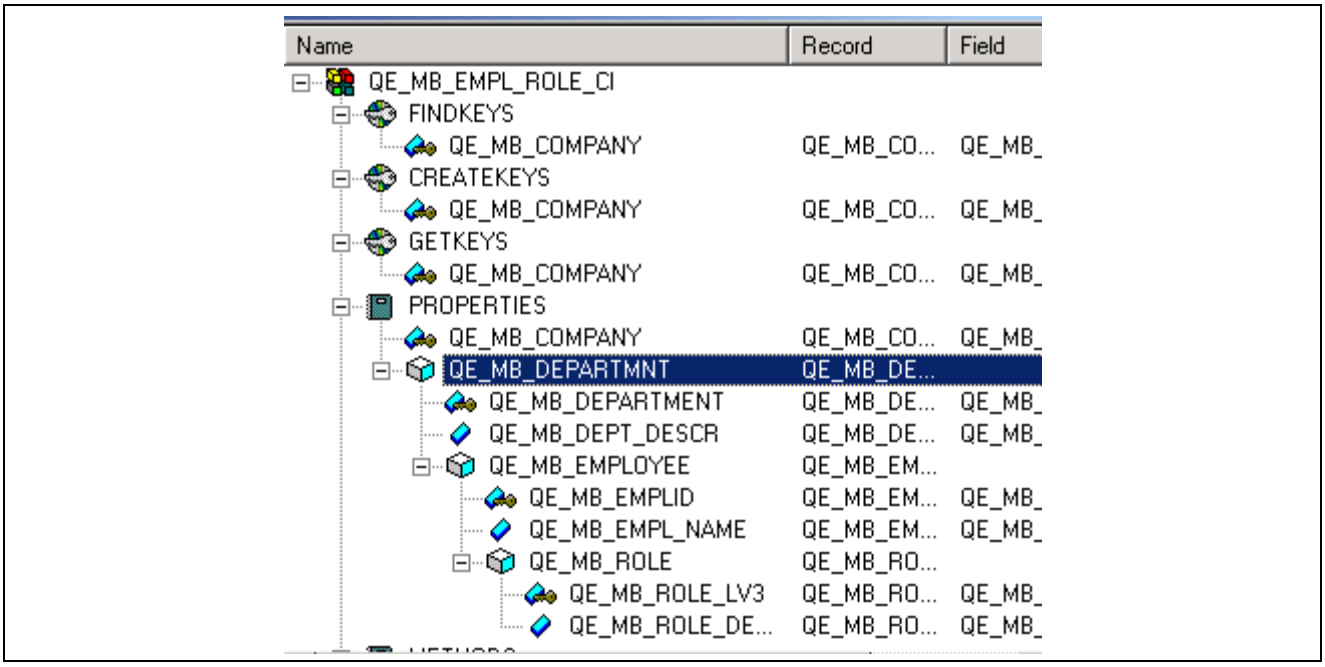
DataCollection Class

A Data Collection object is created when you have a Mobile Page that is based upon a synchronizable Component Interface that contains levels. You do not directly instantiate the DataCollection object.

The synchronizable Component Interface upon which the following Mobile Page is based has the following levels:

- level one: QE_MB_DEPARTMNT
- level two: QE_MB_EMPLOYEE
- level three: QE_MB_ROLE

Use the DataCollection methods and property to access the mobile objects that map to the rows on these levels.



Component Interface Data Collection Levels

Data Collection Methods

Following are the methods for the DataCollection class.

GetListViewAttrs

Syntax

GetListViewAttrs(*propertyname*)

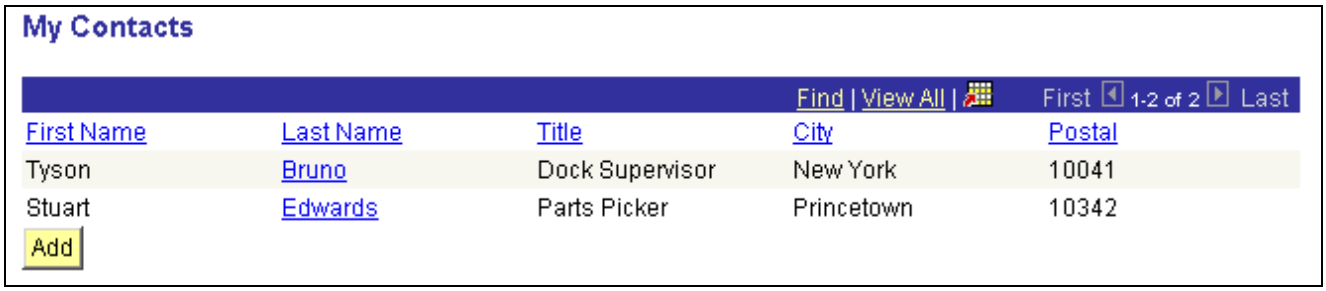
Copyright © 1988-2007, Oracle. All rights reserved.

Description

This method returns the List View attributes for this property. These attributes describe display attributes for a column on a List View.

In the following screen shot, the columns include:

- First Name
- Last Name
- Title
- City
- Postal code



List View columns

Using the GetListViewAttrs method, you can return a reference to one of these columns, then using the ListViewAttrs properties, you can change how the column displays.

Parameters

Parameter	Description
propertyname	This is the name of the property for which the List View attributes are retrieved.

Returns

A ListViewAttrs object containing information about the property column.

Example

```
/* Change the Label for the City Property which is a member of the objects
in the QE_DEMO_SITE Collection  to "Town"*/
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.GetListViewAttrs("City");
&ListAttrs.Label = "Town";
```

Item

Syntax

Item(*index*)

Description

The Index method returns a reference to the mobile object referenced at *index*. This mobile object on the Mobile Page is the equivalent of one row on a level in the synchronizable Component Interface upon which the Mobile Page is based.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
index	Specify the index number for the mobile object to return. Indexes start at 1.

Returns

A reference to the mobile object that was returned.

Example

```
Local ApiObject &ob, &Session;
Local ApiObject &PropertyInfo;

&ob = %ThisMobileObject;
/***********************************************
* Insert some children                         *
***********************************************/
&ChildCount = &ob.MyChild1.Count;

&ob.MyChild1.InsertItem(&ChildCount);
&ChildObject = &ob.MyChild1.Item(&ChildCount + 1);

/***********************************************
* Manipulate Collections                       *
***********************************************/
&ob.MyChild1.InsertItem(0);
&ChildObject = &ob.MyChild1.Item(1);
&ChildCount = &ob.MyChild1.Count;
If (&ChildCount > 0) Then;
   For &index = 1 To &ChildCount
     &ChildObject = &ob.MyChild1.Item(&index);
     &PropertyInfo = &ChildObject.GetPropertyInfoByName("version");
     &PropertyInfo.Label = "Version: ";
     &ob.MyChild1.Item(&index).version = &index + 100;
     &ob.MyChild1.Item(&index).GetPropertyInfoByName("version").visible = True;
   End-For;
End-If;
```

DeleteItem

Syntax

DeleteItem(*location*)

Description

The DeleteItem method removes the specified item from the collection and deletes the mobile object. This mobile object on the Mobile Page is the equivalent of one row on a level in the synchronizable Component Interface upon which the Mobile Page is based.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. Once you delete an item, it is deleted immediately. You no longer have access to it.

Parameters

Parameter	Description
location	Specify the location within the collection of the mobile object to delete. This parameter takes a numeric value.

Returns

A Boolean value: True if the mobile object was successfully deleted, False otherwise.

Example

```
Local ApiObject &ob, &Session;
Local ApiObject &PropertyInfo;

&ob = %ThisMobileObject;
/***********************************************
* Insert some children                         *
***********************************************/
&ChildCount = &ob.MyChild1.Count;

&ob.MyChild1.InsertItem(&ChildCount);
&ChildObject = &ob.MyChild1.Item(&ChildCount + 1);

/***********************************************
* Manipulate Collections                       *
***********************************************/
&ob.MyChild1.DeleteItem(1);
&ChildObject = &ob.MyChild1.Item(1);
&ChildCount = &ob.MyChild1.Count;
```

InsertItem

Syntax

InsertItem(*location*)

Description

The InsertItem method inserts a new mobile object item at the specified location, plus 1. This mobile object on the Mobile Page is the equivalent of one row on a level in the synchronizable Component Interface upon which the Mobile Page is based.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
location	Specify the location of the new item. The mobile object is inserted at one past the number specified with *location*. For example, InsertItem(0) places the item in the first location, which is 1. InsertItem(count) places the new item in the last location: count + 1.

Returns

A Boolean value: True if the mobile object was successfully inserted, False otherwise.

Example

```
Local ApiObject &ob;

&ob = %ThisMobileObject;


If &MobileObjectValueExists = False Then
   &qe_mb_empl.create();
   /*Populate Level 1 data.  */
   &NewCICounter_LV1 = 0;
   &ob_top_parent = &ob.gettopparent();
   While &NewCICounter_LV1 <> &ob_top_parent.QE_MB_DEPARTMNT.count;
      /*Insert Level 1 Row  */
       &qe_mb_empl.QE_MB_DEPARTMNT.InsertItem(&NewCICounter_LV1 + 1);
       &qe_mb_empl.QE_MB_DEPARTMNT.Item(&NewCICounter_LV1 + 1).QE_MB_DEPARTMENT =
       &ob_top_parent.QE_MB_DEPARTMNT.Item(&NewCICounter + 1).QE_MB_DEPARTMENT_LV1;
       &qe_mb_empl.QE_MB_DEPARTMNT.Item(&NewCICounter_LV1 + 1).QE_MB_DEPT_DESCR =
       &ob_top_parent.QE_MB_DEPARTMNT.Item(&NewCICounter_LV1 + 1).QE_MB_DEPT_DESCR;
       &NewCICounter_LV2 = 0;
    End-While;
End-If;
```

Data Collection Properties

Following is the property for DataCollection.

Count

Description

The number of mobile objects in the collection. This is the equivalent of the number of rows on a level in the synchronizable Component Interface upon which the Mobile Page is based.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local ApiObject &ob, &Session;
Local ApiObject &PropertyInfo;

&ob = %ThisMobileObject;
/*********************************************
* Insert some children                       *
*********************************************/
&ChildCount = &ob.MyChild1.Count;

&ob.MyChild1.InsertItem(&ChildCount + 1);
&ChildObject = &ob.MyChild1.Item(&ChildCount + 1);

&ob.MyChild1.InsertItem(1);
&ChildObject = &ob.MyChild1.Item(1);
&ChildCount = &ob.MyChild1.Count;
```

ListViewAttrs Class

The ListViewAttrs class objects are returned from the GetListViewAttrs data collection method.

The ListViewAttrs class has methods and properties that provide information about the properties for the synchronizable Component Interface upon which the Mobile Page is based.

ListViewAttrs Method

Following is the ListViewAttrs method.

IsSet

Syntax

IsSet(*attribute*)

Description

The IsSet method indicates if the given attribute had been explicitly set from PeopleCode, or if it is in an un-initialized state.

Parameters

Parameter	Description
attribute	Specify one of the ListViewAttrs properties, which contains display information about a mobile property column in a List View.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

This method returns a Boolean value: True if this attribute has been set, False otherwise.

Example

```
/* Write the results to the results string  */

&results_string = "The following is " |
&ThisMobileObject.QE_DEMO_SITE("City").IsSet("Visible") |
": City is visible. "
```

ListViewAttrs Properties

Following are the ListViewAttrs properties.

DetailViewLabel

Description

Use this property to change the label shown at the top of a Mobile Page Detail View. DetailViewLabel is only available to a ListViewAttrs object that has been fetched with an empty string.

This property is read-write.

Example

Note that in this example, GetListViewAttrs uses an empty string as input. This is how to call GetListViewAttrs for use with DetailViewLabel.

```
/* Pass empty string */
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.Item(1).QE_DEMO_SITE_H.GetListViewAttrs⇒
("");
&ListAttrs.DetailViewLabel = "My Site Details";
```

Label

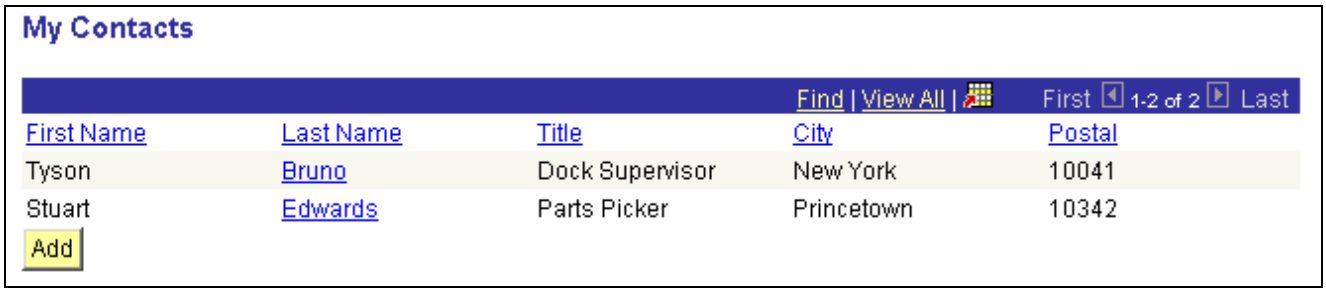
Description

Use this property to change the label shown at the top of a column in a List View; it is the label for a property for mobile objects in a Data Collection.

This property is read-write.

In the following example, there are five columns, hence five labels: First Name, Last Name, Title, City and Postal.

Copyright © 1988-2007, Oracle. All rights reserved.



List View columns

Example

```
/* Change the Label for the City Property which is a member of the objects
in the QE_DEMO_SITE Collection to "Town"*/
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.GetListViewAttrs("City");
&ListAttrs.Label = "Town";
```

ListViewLabel

Description

Use this property to change the tab label. The tab label is indirectly exposed through PeopleCode; it overwrites the List View Label, which is the default if no tab label is defined. However, if the ListViewLabel property is set for a property that has a tab label, the tab label is overridden with the PeopleCode ListViewLabel value.

ListViewLabel is only available to a ListViewAttrs object that has been fetched with an empty string.

This property is read-write.

Example

Note that in this example, GetListViewAttrs uses an empty string as input. This is how to call GetListViewAttrs for use with ListViewLabel.

```
/* Pass empty string */
&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.Item(1).QE_DEMO_SITE_H.GetListViewAttrs⇒
("");
&ListAttrs.ListViewLabel = "My Sites";
```

Visible

Description

This property is True if this column for a property in a List View is visible. Setting this property to False hides the column.

Note. Even if a property has been marked as Invisible on the mobile page definition in PeopleSoft Application Designer, you can still override this property using PeopleCode.

This property is read-write.

Example

```
/* Make invisible the City property column which is a member of
```


Copyright © 1988-2007, Oracle. All rights reserved.

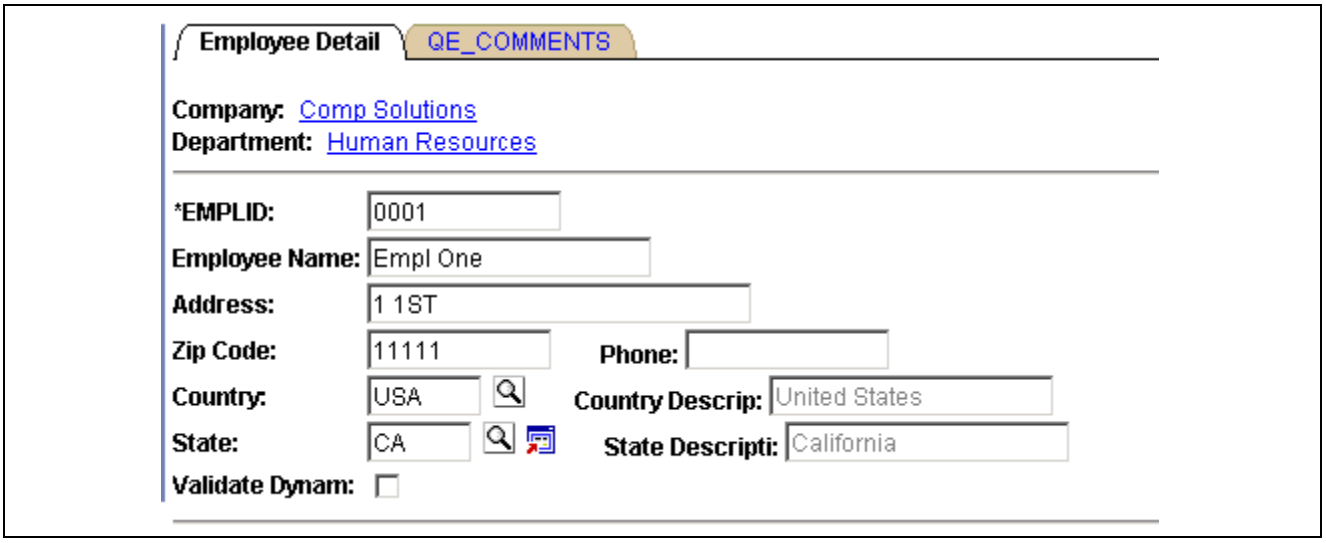
```
the objects in the QE_DEMO_SITE Collection */

&ListAttrs = %ThisMobileObject.QE_DEMO_SITE.GetListViewAttrs("City");
&ListAttrs.Visible = False;
```

PropertyAttrs Class

The PropertyAttrs class objects are returned from the GetPropertyAttrsByName method. This class has methods and properties that provide information about the properties for the synchronizable Component Interface upon which the Mobile Page is based.

For example, to hide the QE_MB_ADDRESS property on the synchronizable Component Interface upon which the current Mobile Page and its mobile objects are based, you can use the following PeopleCode on an OnChange event.

```
%ThisMobileObject.GetPropertyAttrsByName("QE_MB_ADDRESS").Visible = False;
```



Mobile Page Detail View

PropertyAttrs Method

Following is the PropertyAttrs method.

IsSet

Syntax

IsSet(*attribute*)

Description

The IsSet method indicates if the given attribute had been explicitly set, or if it is in an un-initialized state.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
attribute	Specify one of the PropertyAttrs properties, which contains display information about a mobile property.

Returns

This method returns a Boolean: True if this attribute has been set, False otherwise.

Example

```
/*Write the results to the results field  */

&qe_mb_obj.QE_MB_RESULTS = "The following is " |
&qe_mb_obj_setup.GetPropertyAttrsByName(&CharacterPropertyName).IsSet("Visible") |
": " | GetPropertyAttrsByName(&CharacterPropertyName) | "is visible. ";
```

PropertyAttrs Properties

Following are the PropertyAttrs properties.

DisplayOnly

Description

This property is True if the mobile property is display only; False otherwise. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

Note. If a property has been marked as Display Only on the mobile page definition in PeopleSoft Application Designer, you can override this property using PeopleCode. If a property has been marked as Display Only on the Component Interface, you can *not* override this property using PeopleCode.

If you use this property to make a collection display only, all the fields will be display only, except for those that are displayed as pushbuttons. To make a pushbutton display only, you must set the pushbutton property specifically to be display only, either in Application Designer or with PeopleCode. You must set the pushbutton property for every row to be display only.

This property is read-write.

Example

```
Declare Function Hide_or_Unhide_WorkingSetProps_SetValues PeopleCode QE_MB_PC_⇒
DER.QE_MB_GET FieldFormula;

/*If Get() checkbox is selected, unhide Set Value Properties, and grey
Create Check Box. Also maske sure Get() check box is ungrayed. */
If %ThisMobileObject.QE_MB_GET = True Then
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   /*Make sure Create() Checkbox now has value "N"  */
   %ThisMobileObject.QE_MB_CREATE = False;
   Hide_or_Unhide_WorkingSetProps_SetValues( True);
   %ThisMobileObject.GetPropertyAttrsByName("QE_MB_CREATE").DisplayOnly = True;

Else
   /* If Get() checkbox is unselected, hide Set Value Properties, and ungrey
   Create Check Box. */
   If %ThisMobileObject.QE_MB_GET = False Then
      Hide_or_Unhide_WorkingSetProps_SetValues( False);
      %ThisMobileObject.GetPropertyAttrsByName("QE_MB_CREATE").DisplayOnly = False;
   End-If;
End-If;
```

InError

Description

Set this property within an OnSaveEdit event to cause any field on a Detail View that contains incorrect information to be displayed as red. This alerts the user to correct the data in that field.

This field is automatically reset. The developer does not have to reset this field.

This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
%ThisMobileObject.GetPropertyAttrsByName("QE_MB_ADDRESS").InError = True;
```

Label

Description

Use this property to manipulate the label for this mobile property. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
&ob = %ThisMobileObject;
&PropertyInfo = &ob.GetPropertyAttrsByName("text");
&PropertyInfo.Label = "This is text";
```

ShowRequiredCue

Description

An asterisk (*) is displayed beside fields that are defined as Required in PeopleSoft Application Designer. You can use this property to specify whether this asterisk, also called the required field cue, is displayed for a particular field.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is True if the mobile property displays the RequiredCue; False otherwise. This only affects the Mobile Page; this is not saved to the mobile database and is not synchronized.

Use this to make a field be required that was not specified as being required when the synchronized Component Interface was created in the PeopleSoft Application Designer. (You cannot make a required field be not required.)

For example, many fields are made required through PeopleCode. This means they aren't defined as Required in PeopleSoft Application Designer, and the end user may be confused. For these fields, you can use this property.

Note. This property affects only fields where a required field cue is otherwise permissible. That is, regardless of the setting of the property, no cue is ever shown on a push button, a display-only field, and so on.

This property is read-write.

Example

```
&ob = %ThisMobileObject;
&PropertyInfo = &ob.GetPropertyAttrsByName("text");
&PropertyInfo.ShowRequiredCue = True;
```

Visible

Description

This property is True if this field is visible in the Mobile Page Detail View displaying it. Setting this property to False hides the field.

This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

Note. Even if a property has been marked as Invisible on the mobile page definition in PeopleSoft Application Designer, you can still override this property using PeopleCode.

If you set this value to False for a collection property, the collection is no longer visible.

Note. You can only make collections invisible using PeopleCode. You cannot set a collection to be invisible in Application Designer.

This property is read-write.

Example

```
If %ThisMobileObject.QE_MB_PROP_METH = "Visible" Or
   %ThisMobileObject.QE_MB_PROP_METH = "DisplayOnly" Or
   %ThisMobileObject.QE_MB_PROP_METH = "ShowRequiredCue" Or
   %ThisMobileObject.QE_MB_PROP_METH = "SetModified" Then
   %ThisMobileObject.getpropertyattrsbyname("QE_MB_TRUE_FALSE").Visible = True;
End-If;

If %ThisMobileObject.QE_MB_PROP_METH = "Item" Or
   %ThisMobileObject.QE_MB_PROP_METH = "DeleteItem" Or
   %ThisMobileObject.QE_MB_PROP_METH = "InsertItem" Then
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   %ThisMobileObject.getpropertyattrsbyname("QE_MB_PC_ITEM").Visible = True;
End-If;
```

PeerDefaultAttributes Class

PeerDefaultAttributes are returned by the GetPeerDefaultAttributesByName method of a mobile object.

When a peer reference does not have a value associated with it, the display attributes can only be accessed using the PeerDefaultAttributes class. After a peer reference has values, it can access using the ListViewAttrs class.

PeerDefaultAttributes Class Method

The following is the PeerDefaultAttributes class method.

IsSet

Syntax

IsSet(*Attribute*)

Description

The IsSet method indicates if the given attribute had been explicitly set, or if it is in an un-initialized state.

Parameters

Parameter	Description
Attribute	Specify one of the PeerDefaultAttributes properties.

Returns

A Boolean value: True if the attribute has been explicitly set, False otherwise.

Example

```
&PeerAtt = %ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB",⇒
 "ADDRESS");

If &PeerAtt.IsSet(Visible) Then

/* some code */

End-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

PeerDefaultAttributes Properties

This section discusses the PeerDefaultAttributes properties in alphabetical order.

DisplayOnly

Description

This property is True if the peer reference is display only; False otherwise. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

Note. If a property has been marked as Display Only on the mobile page definition in PeopleSoft Application Designer, you can override this property using PeopleCode. If a property has been marked as Display Only on the Component Interface, you can *not* override this property using PeopleCode.

This property is read-write.

Example

```
&IsVisible =
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").Displ⇒
ayOnly;

If &DisplayOnly Then

/* some code */

Else

/* some other code */

End-if;
```

Label

Description

Use this property to manipulate the label for this peer reference. This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").Label⇒
 = &Label;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Visible

Description

This property is True if this peer reference is visible in the Mobile Page Detail View displaying it. Setting this property to False hides the peer reference.

This only affects the Mobile Page; this value is not saved to the mobile database and is not synchronized.

This property is read-write.

Example

```
%ThisMobileObject.GetPeerDefaultAttributesByName("CI_CUSTOMER_MB", "ADDRESS").Visib⇒
le = True;
```

Classes and Functions and System Variables Used in Mobile

Mobile uses a subset of the classes, functions, and system variables used in PIA. Here is a list of those used in mobile.

PeopleCode Classes Used in Mobile

The following is a list of the PeopleCode classes used in mobile.

- Application Package
- Array
- Component Interface
- Exception
- Request (part of iScript)

Considerations Using the Request Class

Only the following request class method (%Request) is supported for mobile:

GetParameter

Only the following request class properties (%Request) are supported for mobile:

- BrowserType
- BrowserVersion
- HTTPMethod
- Protocol

Note. Other request class methods and properties may not return an error when used in mobile, but do return a blank value, an empty array, False, and so on, according to the datatype of the value returned.

See Chapter 18, “Internet Script Classes (iScript),” Request Class Methods, page 596.

Copyright © 1988-2007, Oracle. All rights reserved.

System Variables Used in Mobile

The following is a list of the system variables used in mobile, by category.

- APIs: %Session, %Super, %This, %CompIntfcName
- Data and Time: %Date, %DateTime, %Request, %Time
- Mobile specific: %DeviceType, %MobilePage, %ThisMobileObject

Considerations Using Default Values

For a record field, you can specify constants for a default, such as %Date. Mobile only supports the following for record field defaults. In addition, these are only supported for date, datetime, and time type fields (as appropriate):

- %ClientDate
- %Date
- %DateTime
- %Time

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "System Variables"

Built-In Functions Used in Mobile

The following is a list of the built-in functions used in mobile, by category.

- Arrays: CreateArray, CreateArrayAny, CreateArrayRept, Split
- Conversion: Char, Code, String, Value
- Date and Time: AddToDate, AddToDateTime, AddToTome, Date, Date3, DatePart, DateTime6, DateTimeValue, DateValue, Day, Days, Days360, Days365, Hour, Minute, Month, Second, Time, Time3, TimePart, TimeValue, Weekday, Year
- Internet: GetHTMLText
- Language Constructs: Break, Declare Function, Evaluate, Exit, For, Function, If, Local, Repeat, Return, While
- Math: Abs, Acos, Asin, Atan, Cos, Cot, Degrees, Exp, Fact, Int, Integer, Ln, Log10, Nod, Product, Radians, Rand, Round, Sign, Sin, Sqrt, Tan, Truncate
- Message Catalog and Message Display: MessageBox, MsgGet, MsgGetExplainText, MsgGetText
- Mobile: TransferMobilePage, GenerateMobileTree
- OLE: CreateObject, ObjectDoMethod, ObjectGetProperty, ObjectSetProperty
- Saving: DoSave

Note. DoSave can only be used in the OnChange and OnChangeEdit events.

- String: Clean, Code, Exact, Find, Left, Len, Lower, LTrim, Replace, Rept, Right, Rtrim, String, Substitute, Substring, Upper
- Validation: Error

Copyright © 1988-2007, Oracle. All rights reserved.

- Workflow: Warning

Considerations Using DoSave

The DoSave built-in function is supported in the following Mobile events:

- OnChange
- OnChangeEdit

Save processing (including OnSaveEdit, OnSavePreChange, and OnSavePostChange) is executed exactly as if the end user clicked the save button. OnObjectChangeEdit and OnObjectChange are executed prior to initiating the save processing.

If an error occurs during save processing, it is handled the same as if the user clicked the save button. For example, if save processing encounters an OnSaveEdit error, it displays the appropriate message box. If the error is on the current detail view, the detail view is rendered with the error highlighted. If the error is not on the current detail view, the user is asked if they would like to transfer to the object with the error. The user can correct the error and continue working with the component. If the PeopleCode calls DoSave followed by TransferMobilePage and an error occurs during save processing, the TransferMobilePage is not executed.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions"

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 24

Notification Classes

This chapter provides an overview of the Notify classes and discusses the following topics:

- Scope of the notification classes.
- Data types of the notification classes.
- How to import the notification classes.
- How to create a notification object.
- Notification class constructors.
- Notification class reference.

Understanding Notification Classes

The Notification classes are used for ad-hoc notification. These classes enable you to create and send a notification to someone. The notifications can either be a Peoplesoft Worklist item or an SMTP email message. The Notification class also stores notifications in a table.

You can also use web services to create Worklist entries, then send notifications of completion using application messages.

The Notification classes can be called from a component, an internet script, or an Application Engine program. The WSWorklistEntry class can be used incase you are developing your own PT_WORKLIST notification handler. .

Most of the classes, as well as most of the properties and methods that make up the Notification classes have a GUI representation in Peoplesoft Workflow. This document assumes that the reader is familiar with PeopleSoft Workflow.

See Also

Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology, "Understanding PeopleSoft Workflow"

Scope of the Notification Classes

The Notification classes can be called from a component, an internet script, Integration Broker, or an Application Engine program.

Notification classes can be of Local, Global, or Component scope.

Copyright © 1988-2007, Oracle. All rights reserved.

Data Types of the Notification Classes

Every Notification class is its own data type, that is, Notifications are declared as data type Notification, Notification addresses are declared as type NotificationAddress, and so on.

The following are the data types of the Notification classes:

- Notification
- NotificationAddress
- NotificationTemplate
- WorklistEntry
- Worklist
- WSWorkListEntry

How to Import the Notification Classes

The Notification classes are *not* built-in classes, like Rowset, Field, Record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the Notification classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_WF_NOTIFICATION contains the following classes:

- Notification
- NotificationAddress
- NotificationTemplate

The application package PT_WF_WORKLIST contains the following classes:

- Worklist
- WorklistEntry
- WSWorkListEntry

The import statements you should use are as follows:

```
import PT_WF_NOTIFICATION:*;
import PT_WF_WORKLIST:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

See Also

Chapter 6, “Application Classes,” page 153

 Copyright © 1988-2007, Oracle. All rights reserved.

How to Create a Notification Object

After you've imported the Notification classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of the WorklistEntry class, as the variable &MyWE, with local scope:

```
Local WorklistEntry &MyWE = Create WorklistEntry();
```

See Also

Chapter 24, "Notification Classes," Notification Classes Constructors, page 881

Notification Classes Constructors

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the Notification classes.

Notification

Syntax

Notification(*NotifyFrom*, *dttmCreated*, *language_cd*)

Description

Use Notification to instantiate a notification object.

Parameters

Parameter	Description
NotifyFrom	Specify the entity the notification is being sent from, as a string. Avoid overusing generic entities with this parameter, as this value, along with the value for *dttmCreated* are the key to understanding where notifications originated from.
dttmCreated	Specify the datetime the notification is being sent, as a datetime value.
language_cd	Specify the 3-character language code of the current user, as a string.

Returns

A notification object.

Example

```
import PT_WF_NOTIFICATION:*;
import PT_WF_WORKLIST:*;
Local Notification &MyNotify;
&MyNotify = Create Notification("Workflow_Admin", %Datetime, %Language);
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 24, "Notification Classes," Notification Class, page 885

NotificationAddress

Syntax
NotificationAddress(*Oprid*, *Description*, *Language*, *EmailId*, *Channel*)

Description

Use NotificationAddress to instantiate a notification address.

This address is used to build a notification using the Notification class. This class is required if you want to use the Notification data type.

Parameters

Parameter	Description
Oprid	Specify the UserID of the person receiving the notification, as a string.
Description	Specify the description of the entity receiving the notification (distribution group names, people's names, and so on) as a string.
Language	Specify the language code of the entity to be notified as a string.
EmailId	Specify the email address as a string.
Channel	Specify the 'channel' used for the notification, as a string. The values are:

Value	Description
Email	Notification is sent as email
Worklist	Notification is sent as a worklist

Returns

None.

Example

```
import PT_WF_NOTIFICATION:*;

import PT_WF_WORKLIST:*;

Local NotificationAddress &MyNotify_Addy;

&MyNotify_Addy = Create NotificationAddress(%UserID, &Description, %Language,⇒
 &EmailID, &Channel);
```

See Also
Chapter 24, "Notification Classes," NotificationAddress Class, page 890

 Copyright © 1988-2007, Oracle. All rights reserved.

NotificationTemplate

Syntax

NotificationTemplate(*ComponentId*, *Market*, *TemplateId*, *TemplateType*)

Description

Use NotificationTemplate to instantiate a notification template.

If you're specifying a generic template, the first two parameters do not have to have values, that is, you must specify a Null string ("") instead.

Parameters

Parameter	Description
ComponentId	Specify the name of the component the notification is coming from as a string. To specify a generic template, you must specify a Null string ("") for this value instead.
Market	Specify the name of the market of the component the notification is coming from as a string. To specify a generic template, you must specify a Null string ("") for this value instead.
TemplateId	Specify the template ID as a string.
TemplateType	Specify the type of template as a string. Values are: • G: generic template • C: component template

Returns

An instance of the NotificationTemplate class.

Example

```
import PT_WF_NOTIFICATION:*;

import PT_WF_WORKLIST:*;

Local NotificationTemplate &Notify_Temp;

/* create a generic template */

&Notify_Temp = Create NotificationTemplate("", "", "MyTemplate", "G");
```

See Also

Chapter 24, "Notification Classes," NotificationTemplate Class, page 891

Worklist

Syntax

Worklist()

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the Worklist method to instantiate a Worklist object.

Parameters

None.

Returns

An instance of the Worklist class.

Example

The following example code creates a worklist.

```
import PT_WF_WORKLIST:*;

Local Worklist &theWorklist = Create Worklist();
```

See Also

Chapter 24, “Notification Classes,” Worklist Class, page 895

WorklistEntry

Syntax

WorklistEntry()

Description

Use WorklistEntry to instantiate a WorklistEntry object.

Parameters

None.

Returns

An instance of the WorklistEntry class.

Example

```
import PT_WF_NOTIFICATION:*;

import PT_WF_WORKLIST:*;

Local WorklistEntry &MyWorklistEntry = Create WorklistEntry();
```

See Also

Chapter 24, “Notification Classes,” WorklistEntry Class, page 896

 Copyright © 1988-2007, Oracle. All rights reserved.

WSWorklistEntry

Syntax

```
WSWorklistEntry()
```

Description

Use the WSWorklistEntry constructor to instantiate a WSWorklistEntry object.

The WSWorklistEntry class is primarily used with web services.

Parameters

None.

Returns

A reference to a WSWorklistEntry object.

See Also

Chapter 24, “Notification Classes,” WSWorklistEntry Class, page 908

Notification Class

This class is used to send a notification to people or people entities, as channel independent as possible. This enables functionality such as letting a user determine how to receive workflow notifications as a preference. It does not necessarily limit a person or entity to one channel.

See Chapter 24, “Notification Classes,” Notification Classes Examples, page 910.

See *Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology*, “Using Notification Templates”.

Notification Class Import Statements

The Notification class imports the following classes:

- NotificationAddress
- WorklistEntry

Here are the import statements:

```
import PT_WF_NOTIFICATION:NotificationAddress;

import PT_WF_WORKLIST:WorklistEntry;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Notification Class Method

In this section, we discuss the Notification class method Send.

Send

Syntax

```
Send()
```

Description

Use the Send method to send a notification to the entities defined in the NotifyTo, NotifyCC, and NotifyBCC, using the Subject and Message properties.

Parameters

None.

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology, "Understanding PeopleSoft Workflow"

Notification Class Properties

In this section, we discuss the Notification class properties. The properties are discussed in alphabetical order.

ContentType

Description

This property returns the content type of the value specified by the Message property, as a string. This property is used only with email Notifications, that is, with SMTP email.

This property is read-write.

See Also

Chapter 24, "Notification Classes," Message, page 887

dttmCreated

Description

This property returns the Datetime the notification created as a datetime value. Together with NotifyFrom this makes a notification unique.

See Also

Chapter 24, "Notification Classes," NotifyFrom, page 888

 Copyright © 1988-2007, Oracle. All rights reserved.

EmailReplyTo

Description

If an email is sent as a result of the Send method, this property returns the email reply that should be used, as a string. This property can have a different value from NotifyFrom. This property is valid only with notifications that have "email" specified as the channel.

This property is read-write.

See Also

Chapter 24, “Notification Classes,” Send, page 886 and Chapter 24, “Notification Classes,” NotifyFrom, page 888.

FileNames

Description

This property returns an array of string, populated with the names of any file attachments sent with the notification. This property is valid only with notifications that have "email" specified as the channel.

This property is read-write.

FileTitles

Description

This property returns an array of string, populated with descriptions of any file attachments. The array order *must* be respective to the array order for FileNames, that is, file description matches file name one by using the same array index value. This property is valid only with notifications that have "email" specified as the channel.

This property is read-write.

See Also

Chapter 24, “Notification Classes,” FileNames, page 887

language_cd

Description

This property returns the 3-character language code used for the notification, as a string. This is used for tracking purposes.

This property is read-write.

Message

Description

This property returns the text of the message sent with the notification as a string.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

NotifyBCC

Description

This property returns an array of NotificationAddress objects, populated with the email addresses or worklist users of the people the notification is sent to. This list is used for notification information only, that is, the people on this list are generally not responsible for taking action specified by the notification. In addition, when you use the Notification Send method, other recipients cannot see the email addresses and or users that make up this array.

If you want to send a regular CC list, that is, one where the members of the list can see each other, use the NotifyCC property instead.

See Also

Chapter 24, “Notification Classes,” NotifyCC, page 888

NotifyCC

Description

This property returns an array of NotificationAddress objects, populated with the email addresses of the people the notification is sent to. This list is used for notification information only, that is, the people on this list are generally not responsible for taking action specified by the notification. When using the Notification Send method, the email address or worklist users of this array are exposed to the recipients of the notification.

If you want to send a blind-CC list, that is, where the members of the list cannot see each other, use the NotifyBCC property instead.

See Also

Chapter 24, “Notification Classes,” NotifyBCC, page 888

NotifyFrom

Description

This property returns the ID of the entity sending the notification as a string. For example, ad-hoc workflow uses the current UserID.

This property is read-write.

NotifyGuid

Description

This property returns the GUID of the notification. This is a unique identifier used for tracking purposes only.

This property is read-write.

NotifyTo

Description

This property returns an array of NotificationAddress objects, populated with the email addresses of the people the notification is sent to, and who are responsible for taking action if necessary.

 Copyright © 1988-2007, Oracle. All rights reserved.

To send the list to additional people, use the NotifyCC or NotifyBCC properties.

See Also

Chapter 24, "Notification Classes," NotifyCC, page 888 and Chapter 24, "Notification Classes," NotifyBCC, page 888.

SourceComponent

Description

This property returns the name of the source component, that is, that initiated the notification process. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

SourceMarket

Description

This property returns the name of the market of the source component, that is, that initiated the notification process. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

SourceMenu

Description

This property returns the name of the menu of the source component, that is, that initiated the notification process. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

Subject

Description

This property returns the subject used in the email notification as a string.

This property is read-write.

Template

Description

If the notification message originated from a template, this property returns the name of that template, as a string. This property is used for transaction tracking purposes only. This property is not required.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

NotificationAddress Class

The main purpose of the NotificationAddress is to define an entity to a 'channel', it is concerned at a high level only that a notification was sent.

See Chapter 24, "Notification Classes," Notification Classes Examples, page 910.

See *Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology*, "Using Notification Templates".

NotificationAddress Class Properties

In this section, we discuss the NotificationAddress class properties. The properties are discussed in alphabetical order.

Channel

Description

This property returns the 'channel' used for the notification, as a string. The values are:

Value	Description
Email	Notification is sent as email.
Worklist	Notification is sent as a worklist.

This property is read-write.

Description

Description

This property returns the description of the entity as a string. Entity descriptions include distribution group names or people's names.

This property is read-write.

EmailId

Description

This property returns the Email Address of the entity to be notified.

This property is read-write.

Language

Description

This property returns the language code of the entity to be notified.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Oprid

Description

This property returns the UserID of the person receiving the notification as a string.

This property is read-write.

NotificationTemplate Class

This class is useful for populating the message property of the Notification class.

See *Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology*, "Using Notification Templates".

NotificationTemplate Class Methods

In this section, we discuss the NotificationTemplate class methods. The methods are discussed in alphabetical order.

GetAndExpandTemplate

Syntax

```
GetAndExpandTemplate(Language, Vars)
```

Description

Use the GetAndExpandTemplate method to expand and map the variables into the Text property of the template.

If this is a component template, you must use the SetupCompVarsAndRcpts method first, before you use this method.

If this is a generic template, you must use the SetupGenericVars method first, before you use this method.

The value returned by the Setup methods must be used as the second parameter for this method.

Parameters

Parameter	Description
Language	Specify the 3-character language code used with the notification.
Vars	Specify the variable representing the XML string returned by either the SetupCompVarsAndRcpts method or the SetupGenericVars method.

Returns

Returns a Boolean value: True if the template expanded properly, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
import PT_WF_NOTIFICATION:NotificationTemplate;

Local NotificationTemplate & mynotifytemplate;

&mynotifytemplate = create NotificationTemplate(%Component, %Market, &template⇒
Name, "C");

&xmlVars = &mynotifytemplate.SetupCompVarsAndRcpts(GetLevel0());

&mynotifytemplate.GetAndExpandTemplate(%Language, &xmlVars);
```

See Also

Chapter 24, "Notification Classes," SetupCompVarsAndRcpts, page 892 and Chapter 24, "Notification Classes," SetupGenericVars, page 893.

SetupCompVarsAndRcpts

Syntax

SetupCompVarsAndRcpts(*&Rowset_Context*)

Description

Use the SetupCompVarsAndRcpts method to set up variables for the component from the provided rowset. This method also sets up any additional recipients defined in the component.

The rowset corresponds to the component that the notification template was instantiated with. For example, if you created a notification template for JOB, the rowset used with this method should be based on the JOB component.

If you want to expand a component template, you must use this method first, before expanding the template.

Parameters

Parameter	Description
&Rowset_Context	Specify an already instantiated and populated rowset containing the necessary variables.

Returns

A string containing XML that is used by the GetAndExpandTemplate method.

Example

```
import PT_WF_NOTIFICATION:NotificationTemplate;

Local NotificationTemplate & mynotifytemplate;

&mynotifytemplate = create NotificationTemplate(%Component, %Market, &template⇒
Name, "C");

&xmlVars = &mynotifytemplate.SetupCompVarsAndRcpts(GetLevel0());
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&mynotifytemplate.GetAndExpandTemplate(%Language, &xmlVars);
```

See Also

Chapter 24, "Notification Classes," GetAndExpandTemplate, page 891

SetupGenericVars

Syntax

SetupGenericVars(*&AryValues*)

Description

Use the SetupGenericVars method to pass values to a generic template.

To expand a generic template, you must use this method first, before expanding the template.

Parameters

Parameter	Description
&AryValues	Specify an array of string containing values used with the expanded template.

Returns

A string containing XML that is used by the GetAndExpandTemplate method.

Example

```
import PT_WF_NOTIFICATION:NotificationTemplate;

. . .

&mynotifytemplate = create NotificationTemplate("","", "MYTEMPLATE", "G");

/* Populate an array to contain the values needed by */
/* the template */
&aryValues = CreateArrayRept("", 0);
&aryValues.Push("FIRST VALUE");
&aryValues.Push("SECOND VALUE");
&xmlVars = &mynotifytemplate.SetupGenericVars(&aryValues);
&mynotifytemplate.GetAndExpandTemplate(%Language, &xmlVars);
```

See Also

Chapter 24, "Notification Classes," GetAndExpandTemplate, page 891

NotificationTemplate Class Properties

In this section, we discuss the NotificationTemplate class properties. The properties are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

ComponentId

Description

This property returns a component for the component type notification template as a string. This is not required for generic templates.

This property is read-write.

Instruction

Description

This property returns the notification instructions as a string.

This property is read-write.

Language

Description

This property returns the language used to retrieve template descriptions, as a string.

This property is read-write.

Market

Description

This property returns the market for component type notification templates as a string. This is not required for generic templates.

This property is read-write.

Priority

Description

This property returns the priority of the notification template as a string.

This property is read-write.

Responses

Description

This property is populated only if there are any RIMM responses.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Subject

Description

This property returns the subject of the notification template as a string.

This property is read-write.

TemplateId

Description

This property returns the template ID as a string. If you are creating a notification template, this property is always required.

This property is read-write.

TemplateType

Description

This property returns the template type as a string. Values are:

Value	Description
C	Component
G	Generic

If you are creating a notification template, this property is always required.

This property is read-write.

Text

Description

This property returns the Template Text as a string.

This property is read-write.

Worklist Class

The Worklist class is used to make mass modifications to worklist instances. Modifications to one worklist instance should be performed through the WorklistEntry class.

See Chapter 24, "Notification Classes," Worklist, page 883.

Copyright © 1988-2007, Oracle. All rights reserved.

Worklist Class Method

In this section, we discuss the Worklist class method Reassign.

Reassign

Syntax

Reassign(*FromOperid, ToOperid*)

Description

Use the Reassign method to reassign all worklist items with a state less than "Worked" from one user to another.

If you only want to reassign a single worklist entry, use the Reassign WorklistEntry class method.

Parameters

Parameter	Description
FromOperid	Specify the user ID of the user from which you want to assign a worklist, as a string.
ToOperid	Specify the user ID of the user to which you want to assign a worklist, as a string.

Returns

A boolean value: true if worklist was assigned successfully, false otherwise.

See Also

Chapter 24, "Notification Classes," Reassign, page 898

WorklistEntry Class

This class is used to create and or update a worklist entry.

See Chapter 24, "Notification Classes," Creating a WorklistEntry, page 910.

See Chapter 24, "Notification Classes," Updating a WorklistEntry, page 911.

See *Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology*, "Understanding PeopleSoft Workflow".

WorklistEntry Class Methods

In this section, we discuss the WorklistEntry class methods. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Create

Syntax

```
Create()
```

Description

Use the Create method to create a new WorklistEntry object.

This method is *not* the same as the Create *constructor*. The Create constructor only creates an instance of the WorklistEntry class. To add the entry to the database, you must use this method.

The following properties must be set before calling this method:

- busactivity
- buseventname
- busprocname
- worklist

If this method completes successfully, the following properties are populated:

- instanceid
- transactionid

Parameters

None.

Returns

This method returns a numeric value: 0 if there is an error, nonzero if entry created successfully.

Example

```
import PT_WF_WORKLIST:*;

Local WorklistEntry &worklist;
Local Number &Rslt;

&worklist = create WorklistEntry();

&worklist.busprocname  = "Administer Workflow";
&worklist.busactivity  = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";

&Rslt = &worklist.Create();
```

See Also

Chapter 24, “Notification Classes,” busactivity, page 903, Chapter 24, “Notification Classes,” buseventname, page 903, Chapter 24, “Notification Classes,” busprocname, page 903, Chapter 24, “Notification Classes,” instanceid, page 904 and Chapter 24, “Notification Classes,” transactionid, page 907.

Copyright © 1988-2007, Oracle. All rights reserved.

GetResponseStatus

Syntax

GetResponseStatus()

Description

Use the GetResponseStatus method to return the response status of the callback method. This is useful only for entries that are created by web services.

The following properties must be set before calling this method:

- busactivity
- buseventname
- busprocname
- requestmessageid
- worklistname

Parameters

None.

Returns

A number. Valid values are:

Value	Description
0	CreateWorklistEntry message not received yet or this entry was not created by any web service.
1	Message received but not processed yet.
2	Message processed and sent to message queue.
3	Error while sending message to the caller of create worklist entry.
4	Message was sent to the originating web server and was received successfully.

Reassign

Syntax

Reassign(*Operid*)

Description

The Reassign method assigns the worklist in the data buffers to the user specified by *Operid*. This method commits the version of the worklist entry in the data buffer to the database using the Save method.

If you want to reassign an entire worklist, use the Reassign Worklist class method.

Note. You must use the SelectByKey method to populate data buffers with the worklist entry.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Operid	Specify the user ID to which you want to assign the worklist entry, as a string.

Returns

A boolean value: true if the reassignment was successful, false otherwise.

Example

```
import PT_WF_WORKLIST:WorklistEntry;

        Local WorklistEntry &wl = create WorklistEntry();
        &wl.busprocname = WF_WORKLIST_VW2.BUSPROCNAME;
        &wl.busactivity = WF_WORKLIST_VW2.ACTIVITYNAME;
        &wl.buseventname = WF_WORKLIST_VW2.EVENTNAME;
        &wl.worklistname = WF_WORKLIST_VW2.WORKLISTNAME;
        &wl.instanceid = WF_WORKLIST_VW2.INSTANCEID;

        If (&wl.SelectByKey()) Then
           If Not (&wl.Reassign("TOPSUSER2")) Then
              /* Reassign error */
           End-If;
        Else
           /* SelectByKey error */
        End-If;
```

See Also

Chapter 24, “Notification Classes,” Reassign, page 896

Save

Syntax

Save()

Description

Use the Save method to save a WorklistEntry to the database.

PeopleCode Event Considerations

You must include this method within events that allow database updates. This includes the following PeopleCode events:

- SavePreChange (Page)
- SavePostChange (Page)
- Workflow
- FieldChange

Copyright © 1988-2007, Oracle. All rights reserved.

If this method results in a failure, all database updates are rolled back. All information the user entered into the component is lost, as if the user pressed ESC.

Considerations Using Web Services

This method saves the instance of the worklist to the database. If the worklist entry is marked as worked, (that is, if inststatus property is set to 2) and the worklist entry was originally created by a web service, the originating requestor is sent a response when this method is executed. If the requestor is expecting additional data, use the SaveWithCustomData method instead.

Parameters

None.

Returns

A numeric value: 0 if save didn't complete successfully, nonzero if save completed successfully.

Example

```
If (&worklist.Save() <> 0) Then
   /* success */
Else
   /* handle error */
End-If;
```

See Also

Chapter 24, "Notification Classes," Update, page 902

Chapter 24, "Notification Classes," SaveWithCustomData, page 900

Chapter 24, "Notification Classes," inststatus, page 904

SaveWithCustomData

Syntax

SaveWithCustomData(*&Message, &FieldNameArray, &FieldValueArray*)

Description

Use the SaveWithCustomData method to save the worklist entry to the database, as well as to pass additional data back to the requestor that created this worklist entry. You should only use this method with worklist entries that are created by web service. If you want to save a worklist entry without additional data, or a worklist entry that was not created using a web service, use the Save method instead.

If the worklist entry was not created by a web service, or if the worklist entry is not marked as worked (that is, if the inststatus property is not set to 2,) the additional data is ignored.

PeopleCode Event Considerations

You must include this method within events that allow database updates. This includes the following PeopleCode events:

- SavePreChange (Page)
- SavePostChange (Page)

 Copyright © 1988-2007, Oracle. All rights reserved.

- Workflow
- Message Subscription (the Integration Broker INotificationHandler interface)
- FieldChange

If this method results in a failure, all database updates are rolled back. All information the user entered into the component is lost, as if the user pressed ESC.

Parameters

Parameter	Description
&Message	Specify an already instantiated and populated response message containing the custom data to be sent back to the originating web service.
&FieldNameArray	Specify an already instantiated and populated array of string containing the field names to be used by the web service that created the worklist entry.
&FieldValueArray	Specify an already instantiated and populated array of string containing matching field values to be used by the web service that created the worklist entry.

Returns

A message object containing the final message that was sent to the web service if successful, a null object otherwise.

See Also

Chapter 24, "Notification Classes," Save, page 899

Chapter 24, "Notification Classes," inststatus, page 904

SelectByKey

Syntax

```
SelectByKey()
```

Description

The SelectByKey method uses the key field values that have been assigned to build and execute a Select SQL statement. The field values are then fetched from the database SQL table into the worklist entry and the Select statement is closed.

If you don't specify all the key fields, those you exclude are added to the Where clause with the condition equal to a blank value. If not all keys are set and more than one row is retrieved, you won't receive an error and SelectByKey won't fetch any data.

For worklist entries that were created by web services, use the SelectByMessageId method instead.

Parameters

None.

Returns

A boolean value: true if the properties are set based on the database values, false otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 24, “Notification Classes,” SelectByMessageId, page 902

SelectByMessageId

Syntax

```
SelectByMessageId()
```

Description

Use the SelectByMessageId method to select a worklist entry by the message id. This is useful only for entries that are created by web services.

The following properties must be set before calling this method:

- busactivity
- buseventname
- busprocname
- requestmessageid
- worklistname

Parameters

None.

Returns

A boolean value: true if this method completes successfully, false otherwise.

See Also

Chapter 24, “Notification Classes,” SelectByKey, page 901

Update

Syntax

```
Update()
```

Description

Use the Update method to update the worklist entry with any property values that have changed, and assign values to the associated record in the database. However, this method does not commit the changes to the record in the database. You must use the Save method to update the database.

Parameters

None.

Returns

A numeric value: 0 if update didn’t complete successfully, nonzero if update completed successfully.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
If (&worklist.Update() <> 0) Then
   &worklist.inststatus = "2"  /* mark entry worked */
   /* additional processing */
End-if;
```

See Also

Chapter 24, “Notification Classes,” Save, page 899

WorklistEntry Class Properties

In this section, we discuss the WorklistEntry class properties. The properties are discussed in alphabetical order.

actiondttm

Description

This property returns the Worklist action datetime as a datetime.

This property is read-write.

busactivity

Description

This property returns the source activity name of the source business process as a string.

This property is read-write.

buseventname

Description

This property returns the source event name of the source activity as a string.

This property is read-write.

busprocname

Description

This property returns the source business process name as a string.

This property is read-write.

commentshort

Description

This property returns the short comment the previous UserID gave when reassigning a worklist as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

do_replicate_flag

Description

This property returns a flag that indicates if the worklist entry must be replicated. Values are:

Value	Description
Y	Worklist entry needs to be replicated
N	Worklist entry does not need to be replicated

This property is read-write.

instanceid

Description

This property returns the Instance Id. This property is used with the following properties to make the worklist entry a unique number:

- busprocname
- busactivity
- buseventname
- worklistname

This property is read-write.

instselecteddttm

Description

This property returns the datetime that the worklist was selected to be worked as a datetime value.

This property is read-write.

inststatus

Description

This property returns the Worklist Entry status as a string. The values are:

Value	Description
0	new worklist entry
1	selected worklist entry
2	worked worklist entry
3	cancel string

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

insttimeoutddttm

Description

This property returns the datetime the worklist entry timed out. Only some worklist types use this property.

This property is read-write.

instworkeddttm

Description

This property returns the datetime worklist was worked as a datetime value.

This property is read-write.

IsCreatedViaWebService

Description

This property returns a boolean value: true if the WorklistEntry object was created from a web service, false otherwise.

This property is read-only.

oprid

Description

This property returns the UserID that is assigned the worklist as a string.

This property is read-write.

originatorid

Description

This property returns the UserID that caused the worklist to be created as a string.

This property is read-only.

prevoprid

Description

This property returns the previous UserID who used this worklist entry as a string.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

requestmessageid

Description

This property is used with worklist entries that are created by web services. It contains the original message ID from the requesting party.

This property is read-write.

ResponseStatus

Description

This property returns the status of the response sent to the originating web service.

Valid values are:

Value	Description
0	CreateWorklistEntry message not received yet or this entry was not created by any web service.
1	Message received but not processed yet.
2	Message processed and sent to message queue.
3	Error while sending message to the caller of create worklist entry.
4	Message was sent to the originating web server and was received successfully.

This property is read-write.

syncid

Description

This property is reserved for future use.

This property is read-write.

timedout

Description

This property returns the status flag if the worklist has timed out. Values are:

Value	Description
Y	Worklist has timed out
N	Worklist has not timed out

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

transactionid

Description

This property returns the transaction Id as a number, which represents the unit of work. This can be useful for pooled worklists.

This property is read-write.

url

Description

This property returns the URL to work for the worklist as a string. This is used only for replicating worklist entries across PeopleSoft databases.

This property is read-write.

wl_priority

Description

This property returns the priority of the worklist as a string. The values are:

Value	Description
low	Worklist has low priority.
medium	Worklist has medium priority.
high	Worklist has high priority.

This property is read-write.

wldaystoselect

Description

This property returns the number of days to select worklist as a number.

This property is read-write.

wldaystowork

Description

This property returns the number of days to work the worklist as a number.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

worklistname

Description

This property returns the source worklist name for the source event string as a string.

This property is read-write.

WSWorklistEntry Class

This class is used by web services to create Worklist entries.

This class implements the Integration Broker class IRequestHandler.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

WSWorklistEntry Class Methods

The following are the WSWorklistEntry class methods, in alphabetical order.

OnError

Syntax

```
OnError(&Message)
```

Description

Use the OnError method to do error handling. This method implements the Integration Broker IRequestHandler interface OnError method.

Parameters

Parameter	Description
&Message	Specify an already instantiated and populated message object. This would normally be the message that was used to generate the worklist entry from the web service.

Returns

Depends on implementation.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 22, “Message Classes,” page 729

Chapter 37, “SOAPDoc Class,” page 1539

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Sending and Receiving Messages”

OnNotify

Syntax

```
OnNotify(&Message)
```

Description

Use the OnNotify method to creates the worklist entry and the application worklist record.

This method implements the Integration Broker INotificationHandler interface OnNotify method.

Parameters

Parameter	Description
&Message	Specify an already instantiated and populated message object. Generally this would be the originating message that created the worklist entry.

Returns

None.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Sending and Receiving Messages”

WSWorklistEntry Class Properties

The following describes the WSWorklistEntry class properties, in alphabetical order.

InstanceID

Description

This property returns the instance ID for the worklist entry, as a string.

This property is read-only.

TransactionID

Description

This property returns the transaction ID for the worklist entry, as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Notification Classes Examples

The following are example of typical actions you perform using the notification classes.

See *Enterprise PeopleTools 8.49 PeopleBook: Workflow Technology*, "Using Notification Templates".

Creating a WorklistEntry

The following example creates a WorklistEntry.

The following is the complete code sample: the steps explain each line.

```
import PT_WF_WORKLIST:*;

Local WorklistEntry &worklist;

&worklist = create WorklistEntry();

&worklist.busprocname  = "Administer Workflow";
&worklist.busactivity  = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";

If (&worklist.Create() <> 0) Then
   &worklist.oprid = %UserId;
Else
   /* handle error */
End-If;

If (&worklist.Save() <> 0) Then
   /* success, create a corresponding row in */
   /* application worklist record... */
Else
   /* handle error */
End-If;
```

To create a WorklistEntry:

1. Import the WorklistEntry class.

   You must import the WorklistEntry class before you can use it in your PeopleCode program. In addition, the second line declares the variable &worklist as type WorklistEntry.

   ```
   import PT_WF_WORKLIST:*;

   Local WorklistEntry &worklist;
   ```

2. Instantiate a WorklistEntry object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Using the constructor, create an instance of the WorklistEntry class. This does *not* create a WorklistEntry. This creates a WorklistEntry object only. You must use the Create method to actually create the WorklistEntry in the system.

```
&worklist = create WorklistEntry();
```

3. Set the required parameters.

   Some of the WorklistEntry class properties are required for every WorklistEntry. You must set a value for these properties before you can update or save the WorklistEntry.

```
&worklist.busprocname  = "Administer Workflow";
&worklist.busactivity  = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";
```

4. Create the WorklistEntry.

   After you've set the required values, you need to create the WorklistEntry in the system. This does *not* update the database. This does, however, add values to the notification tables.

   If this method completes successfully, the following properties are populated: instanceid and transactionid.

   - instanceid
   - transactionid

```
If (&worklist.Create() <> 0) Then
   &worklist.oprid = %Userid;
Else
   /* handle error */
End-If;
```

5. Save the WorklistEntry.

   After you've created the WorklistEntry, save it. The Save method actually updates the database, and so can be used only in events that allow database updates.

```
If (&worklist.Save() <> 0) Then
   /* success, create a corresponding row in */
   /* application worklist record... */
Else
   /* handle error */
End-If;
```

Updating a WorklistEntry

The following example updates a WorklistEntry so that it is considered worked.

The following is the complete code sample: the steps explain each line.

```
import PT_WF_WORKLIST:*;

Local WorklistEntry &worklist;

&worklist = create WorklistEntry();
&worklist.busprocname = "Administer Workflow";
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&worklist.busactivity = "Send Note";
&worklist.buseventname = "Worklist Note";
&worklist.worklistname = "Worklist Note";
&worklist.instanceid = 2;

&ret = &worklist.SelectByKey();

&worklist.inststatus = "2"; /* mark entry worked */
If (&worklist.Update() <> 0) Then
   If (&worklist.Save() <> 0) Then
      /* success */
   Else
      /* handle error */
   End-If;
Else
   /* handle error */
End-If;
```

To update a WorklistEntry:

1. Import the WorklistEntry class.

   You must import the WorklistEntry class before you can use it in your PeopleCode program. In addition, the second line declares the variable &worklist as type WorklistEntry.

   ```
   import PT_WF_WORKLIST:*;

   Local WorklistEntry &worklist;
   ```

2. Instantiate a WorklistEntry object.

   Using the constructor, create an instance of the WorklistEntry class. This does *not* create a WorklistEntry. This creates a WorklistEntry object only. You must use the Create method to actually create the WorklistEntry in the database tables, and the Save method to update the tables.

   ```
   &worklist = create WorklistEntry();
   ```

3. Set the required parameters.

   Some of the WorklistEntry class properties are required for every WorklistEntry. You must set a value for these properties before you can update or save the WorklistEntry. The instanceid property, combined with the other properties, indicates a unique WorklistEntry. This value is populated after a WorklistEntry has already been created.

   ```
   &worklist.busprocname  = "Administer Workflow";
   &worklist.busactivity  = "Send Note";
   &worklist.buseventname = "Worklist Note";
   &worklist.worklistname = "Worklist Note";
   &worklist.instanceid = 2;
   ```

4. Populate the worklist entry with data.

   After you've set the keys, populate the worklist entry based on those keys using the SelectByKey method.

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&ret = &worklist.SelectByKey();
```

5. Update and save the WorklistEntry.

   After you've set the required parameters, try to update the WorklistEntry. If that is successful, mark the WorklistEntry as 'worked', and try to save the WorklistEntry. The Save method actually updates the database, and so can be used only in events that allow database updates.

```
If (&worklist.Update() <> 0) Then
   &worklist.inststatus = "2"  /* mark entry worked */
   If (&worklist.Save() <> 0) Then
      /* success */
   Else
      /* handle error */
   End-If;
Else
      /* handle error */
End-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 25

Optimization PeopleCode

This chapter discusses how to:

- Use optimization PeopleCode on the application server.
- Use optimization PeopleCode in an Application Engine program.
- Perform optimization in PeopleCode.
- Use lights-out mode with optimization.
- Use optimization built-in functions.
- Use OptEngine class methods.
- Use OptEngine class properties.
- Use the OptBase application class.
- Use OptBase class methods.
- Use OptInterface class methods.

Using Optimization PeopleCode on the Application Server

While running optimization PeopleCode on the application server, ensure that changed data is committed to the database before calling the CreateOptEngine optimization function and the following OptEngine class methods:

- RunSynch
- RunAsynch
- CheckOptEngineStatus
- ShutDown
- SetTraceLevel
- GetTraceLevel
- InsertOptProbInst
- DeleteOptProbInst

Note. The PeopleCode functions CommitWork and DoSaveNow can be called within a step to save uncommitted data to the database before calling the listed functions and methods. Keep in mind that forcing a commit on pending database updates is a serious step; it prevents roll-back on error. CreateOptEngine, ShutDown, InsertOptProbInst and DeleteOptProbInst modify the database, so take care when terminating the Application Engine program without committing the changes made by those calls.

Copyright © 1988-2007, Oracle. All rights reserved.

Using Optimization PeopleCode in an Application Engine Program

When you write an optimization PeopleCode program in an Application Engine program and you schedule it in PeopleSoft Process Scheduler, you must set the process definition with a process type of *Optimization Engine*. Other process types do not allow optimization PeopleCode in Application Engine programs.

While using optimization PeopleCode in Application Engine programs, make sure data is committed before calling the CreateOptEngine optimization function and the following OptEngine class methods:

- RunSynch
- RunAsynch
- CheckOptEngineStatus
- ShutDown
- SetTraceLevel
- GetTraceLevel
- InsertOptProbInst
- DeleteOptProbInst

Note. You can call the PeopleCode functions CommitWork and DoSaveNow within a step to save uncommitted data to the database before calling the listed functions and class methods. Keep in mind that forcing a commit on pending database updates is a serious step; it prevents roll-back on error. CreateOptEngine, ShutDown, InsertOptProbInst and DeleteOptProbInst modify the database, so take care when terminating the Application Engine program without committing the changes made by those calls.

Performing Optimization in PeopleCode

This section discusses how to:

- Create new analytic instances.
- Load analytic instances into an analytic server.
- Run optimization transactions.
- Invoke the Optimization PeopleCode plug-in.
- Shut down optimization engines.
- Delete existing analytic instances.
- Program for database updates.

Creating New Analytic Instances

To create a new analytic instance for an analytic type:

1. Call the function InsertOptProbInst with the analytic type and analytic instance as parameters to create an analytic instance ID.
2. Use PeopleSoft Application Engine or a similar mechanism to load the optimization application tables with data.

 Copyright © 1988-2007, Oracle. All rights reserved.

Use the analytic instance ID as the key value in scenario-managed optimization application tables.

The analytic instance is now ready to be loaded into an analytic server.

Note. You can load multiple copies of the same analytic instance into multiple instances of an analytic server, provided that each instance of the analytic server resides in a different application server domain. Each analytic instance loaded into a given domain must be unique. Within a given domain, you cannot have the same analytic instance in more than one analytic server. The analytic server maintains data integrity by checking to see if the data has been altered by another user (refer to the steps in the optimization system architecture description). Try to maintain data consistency when the same analytic instance uses the same database in different domains.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Understanding PeopleSoft Optimization Framework," PeopleSoft Optimization Framework System Architecture

Chapter 25, "Optimization PeopleCode," InsertOptProbInst, page 939

Loading Analytic Instances Into an Analytic Server

Use the CreateOptEngine function to load an analytic server with an analytic instance. It takes analytic instance ID and a mode parameter with %Synch and %Asynch as possible values and returns a PeopleCode object of type OptEngine.

You can run the PeopleCode on the application server or from PeopleSoft Application Engine.

Loading Analytic Instances by Running PeopleCode on the Application Server

To block PeopleCode from running on the application server until the load is done (synchronous mode), use the %Synch value for the mode parameter. An error is generated if the load isn't successful. The application server imposes a timeout beyond which the PeopleCode and optimization engine load are terminated. Here is a code example:

```
Local OptEngine &myopt;
&myopt = CreateOptEngine("PATSMITH", %Synch);
```

To load the analytic server without blocking the PeopleCode from running (asynchronous mode) on the application server, use the %Asynch value for the mode parameter. The analytic server performs a preliminary check of the load request and returns the OptEngine object if it is successful or an error if it is unsuccessful. A successful return does not mean that the load was successful. You must then use repeated CheckOptEngineStatus methods on the returned OptEngine object to determine whether the analytic engine is done with the load and whether it was successful. Here is a code example:

```
Local OptEngine &myopt;
&myopt = CreateOptEngine("PATSMITH", %Asynch);
```

Loading Analytic Instances by Running PeopleCode in PeopleSoft Application Engine

Both synchronous (%Synch) and asynchronous (%Asynch) modes block the PeopleCode from running on PeopleSoft Application Engine until the load is done. Use only %Asynch while loading an optimization engine.

The absolute number of optimization engine instances that may be loaded in a given domain is governed by a configuration file loaded by Tuxedo during its domain startup.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 25, "Optimization PeopleCode," CheckOptEngineStatus, page 942

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Administering Optimization Server Components"

Running Optimization Transactions

You send an optimization transaction to the optimization engine using the RunSynch and RunAsynch methods. Both are methods on an OptEngine object. The OptEngine object can be created either by calling CreateOptEngine (if the optimization engine is not loaded already) or by calling GetOptEngine (if the optimization engine is already loaded). Both RunSynch and RunAsynch have the same signature, except that RunSynch runs the optimization transaction in synchronous mode and RunAsynch runs it in asynchronous mode. Both return an integer status code. You can run transactions either on the application server or with PeopleSoft Application Engine.

To invoke an optimization transaction:

1. Use the GetOptEngine function to get the OptEngine object as a handle for the optimization engine that has loaded an analytic instance ID.

   Use the CreateOptEngine function to create the OptEngine object for a new optimization engine if the analytic instance has not been loaded.
2. Call RunSynch or RunAsynch to send an optimization transaction to the optimization engine to be run in synchronous or asynchronous mode.
3. If the transaction is run in synchronous mode (RunSynch), use the OptEngine methods GetString, GetNumber, and so on, to retrieve the output result from the optimization transaction.

   The transaction names, parameter names, and data types are viewable in the analytic type in PeopleSoft Application Designer.
4. If the transaction is run in asynchronous mode, use the OptEngine method CheckOptEngineStatus to check the status of the optimization transaction in the optimization engine.

   After the transaction is done, result data is available in the database for retrieval using standard PeopleCode mechanisms.

Running Optimization Transactions from the Application Server

To block the PeopleCode from running on the application server until the optimization transaction is done (synchronous mode) and receives the results, use RunSynch to send an optimization transaction. An error status code is returned if the transaction isn't successful. If successful, you can use other methods to retrieve the results from the transaction call. The application server imposes a timeout beyond which the PeopleCode and optimization engine transaction are terminated.

To run a transaction without blocking PeopleCode from running (asynchronous mode) on the application server, use RunAsynch to send an optimization transaction. In this mode, the optimization engine performs a preliminary check of the transaction request and returns a success or failure status code. A successful return does not mean that the transaction is successful; it means only that the syntax is correct. You must then use repeated calls to the CheckOptEngineStatus method on the OptEngine object to determine whether the optimization engine is done with the transaction and whether it is successful.

RunAsynch does not allow transaction output results to be returned. Use this method for long-running transactions that return results entirely through the database.

 Copyright © 1988-2007, Oracle. All rights reserved.

Running Optimization Transactions from PeopleSoft Application Engine

Both synchronous (RunSynch) and asynchronous (RunAsynch) methods block the PeopleCode from running on PeopleSoft Application Engine until the optimization transaction is done. RunSynch allows output results to be returned, but it should be used for transactions that are fast (less than 10 seconds). RunAsynch does not have a time limit, but it does not return output results.

See Also

Chapter 25, "Optimization PeopleCode," RunAsynch, page 954

Invoking the Optimization PeopleCode Plug-In

If you're developing an optimization application that uses the Optimization PeopleCode plug-in, you must do the following to invoke the plug-in:

- Develop a PeopleCode application class that extends the PT_OPT_BASE:OptBase class.
- Define methods in your application class that use the PeopleCode OptInterface class to perform your optimization functions.
- Define an analytic type that specifies the Optimization PeopleCode plug-in, by selecting the PeopleCode Plugin check box in the analytic type properties.
- Also in the analytic type properties, specify the application package and application class that you developed.
- Define transactions in your analytic type definition that correspond to the methods you developed in your application class, with corresponding parameters.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework, "Designing Analytic Type Definitions," Creating Analytic Type Definitions

Chapter 25, "Optimization PeopleCode," CreateOptInterface, page 935

Chapter 25, "Optimization PeopleCode," OptBase Application Class, page 962

Chapter 25, "Optimization PeopleCode," OptInterface Class Methods, page 978

Shutting Down Optimization Engines

Use the GetOptEngine function to get the OptEngine object as a handle for the optimization engine that loaded an analytic instance ID.

Use the OptEngine method named ShutDown to shut down the optimization engine. This ends the optimization engine process with the current analytic instance ID. Based on application server settings, the system restarts a new, unloaded optimization engine process that can be loaded with any other analytic instance.

See Also

Chapter 25, "Optimization PeopleCode," ShutDown, page 959

Deleting Existing Analytic Instances

To delete an existing analytic instance for an analytic type:

1. Shut down any optimization engines that have this analytic instance currently loaded.

Copyright © 1988-2007, Oracle. All rights reserved.

2. Using PeopleSoft Application Engine or a similar mechanism, delete the data in the optimization application tables pertaining to that analytic instance.

   Use the analytic instance ID as the key value to find and delete analytic instance rows from scenario-managed optimization application tables.

3. Use the function DeleteOptProbInst with the analytic type and analytic instance as arguments to delete the analytic instance ID from PeopleTools metadata.

Note. If you try to delete an existing analytic instance that is loaded in a running optimization engine, DeleteOptProbInst returns %OptEng_Fail, and the optional status reference parameter is set to %OptEng_Exists.

See Also

Chapter 25, "Optimization PeopleCode," DeleteOptProbInst, page 936

Programming for Database Updates

You must plan for uncommitted database changes in your optimization PeopleCode. The Optimization Framework detects pending database updates, and generates a failure status if data is not committed to the database before certain optimization methods are called.

This checking for database updates happens in runtime for the CreateOptEngine function and the following methods: RunSync, RunAsync, Shutdown, GetTraceLevel, and SetTraceLevel. Ensure that your PeopleCode performs proper database updates and commits before you execute these methods. If you use the optional parameter for detailed status that is available for these functions, or the DetailedStatus property that is available for the methods, you can check for the status of %OptEng_DB_Updates_Pending to see if there is a pending database update.

Note. The pending database update may have happened considerably earlier in the code. Forcing a commit within your PeopleCode to avoid this problem prevents roll-back on database error. Forcing a commit should be used with care.

The InsertOptProbInst and DeleteOptProbInst functions can be called only inside FieldChange, PreSaveChange and PostSaveChange PeopleCode events, and in Workflow.

This database update checking happens in compile time for the following functions: InsertOptProbInst and DeleteOptProbInst. Make sure that there are no pending database updates before you execute these methods.

Using Lights-Out Mode with Optimization

This section provides an overview of lights-out mode, and discusses how to:

- Create a request message.
- Create a response message.
- Edit the request PeopleCode.
- Edit the response PeopleCode.

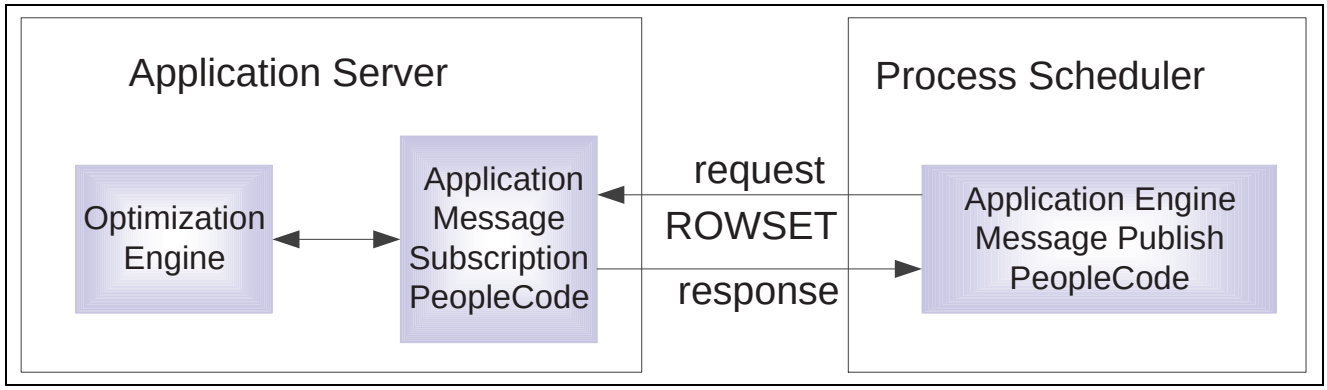
 Copyright © 1988-2007, Oracle. All rights reserved.

Understanding Lights-out Mode

Some optimization applications can take several hours to run. These are typically run as overnight batch jobs every night when the work load is small to regenerate the optimization solution and have it ready for end users to use in the morning — hence the term *lights-out mode.*

In the current release, application messages communicate between the Application Engine batch job and the online optimization engine. After the Application Engine job completes and the optimization solution has been written to the database, an application message initiates the download of the data from the database batch job to the online optimization engine.

Lights-out mode uses an Application Engine PeopleCode program within PeopleSoft Process Scheduler to send requests to an application server and receive responses from it. Within the application server, the OnRequest PeopleCode runs an optimization engine process.



Lights-out process

This request and response is in the form of a rowset as shown by the example supplied with optimization, the OPT_CALL message. Also supplied as an example is an Application Engine message publish PeopleCode program called PT_OPTCALL.

Important! PeopleSoft Application Engine includes an action of type *Log Message*, which PeopleSoft Process Scheduler uses to record its activity in the PS_MESSAGE_LOG table. The PeopleCode MessageBox and WinMessage built-in functions also record their activity in the PS_MESSAGE_LOG table.

During lights-out optimization, these processes can conflict with each other or with the optimization engine when one process locks a row of the table, and another process tries to access the same row.

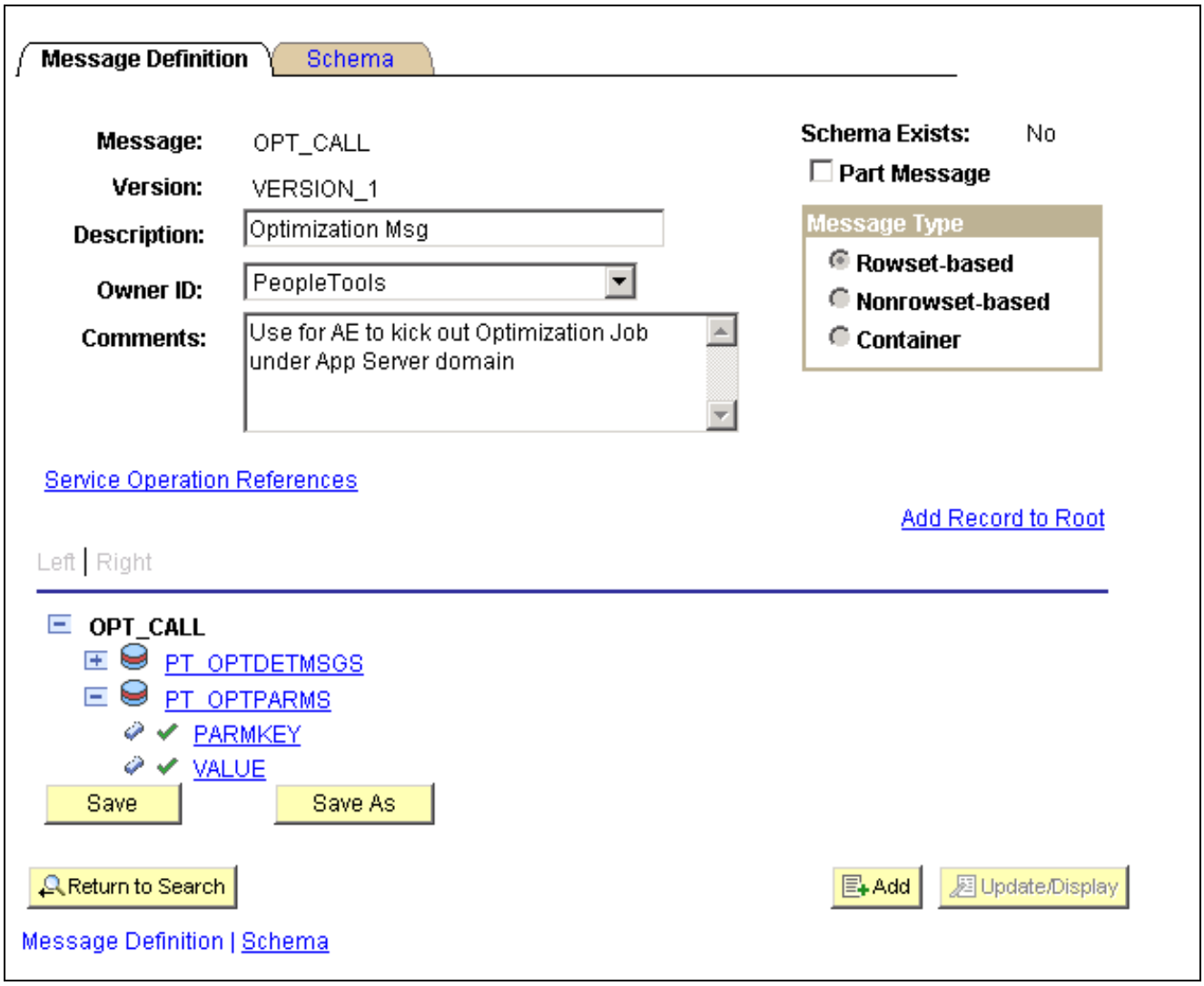
To prevent this conflict, pay close attention to where the MessageBox or WinMessage built-in functions are used in your Application Engine PeopleCode. In general, there can't be any outstanding database updates pending when communicating with the optimization engine using application messages.

The OPT_CALL Message

The OPT_CALL message is an example of what the lights-out process uses as the message for optimization. The OPT_CALL message has a structure using a record, PT_OPTPARMS, having the fields PARMKEY and VALUE which represent a name/value pair. These send requests and responses from the Application Engine PeopleCode in PeopleSoft Process Scheduler to and from the message OnRequest PeopleCode in the application server.

The OPT_CALL message also uses a record, PT_OPTDETMSGS, which contains the information needed for processing a detailed message.

Copyright © 1988-2007, Oracle. All rights reserved.



OPT_CALL message definition

The OPT_CALL message is associated with the OPT_CALL service operation. The OPT_CALL service operation defines the OPT_CALL application package as a handler. This application package implements the Integration Broker methods needed to handle any messaging PeopleCode.

Creating a Request Message

This section provides an overview of the request message and describes how to create messages that:

- Create an optimization engine.
- Check optimization engine status.
- Run an optimization engine transaction.
- Set the trace level.
- Get the trace level.
- Shutdown an optimization engine.

Copyright © 1988-2007, Oracle. All rights reserved.

Understanding the Request Message

For optimization, the Application Engine PeopleCode in PeopleSoft Process Scheduler sends a request OPT_CALL message. The message uses rowsets built from PT_OPTPARMS records as the request. You can use the following rowset structures as an example of how to perform certain optimization actions, by sending them as requests from the application engine program in the process scheduler to the message notification PeopleCode in the application server.

Creating an Optimization Engine

To create an optimization engine, structure the rowset as follows, using the PT_OPTPARMS record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	VALUE Field
OPTCMD	CREATE Causes the PeopleCode program implementing the Integration Broker OnRequest method to load an optimization engine. The OPT_CALL example executes the CreateOptEngine function.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	Y if this optimization engine load is to occur synchronously, N if asynchronously.

Checking Optimization Engine Status

To check optimization engine status (for example, to see when it finishes loading), structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	CHECK_STATUS Causes the PeopleCode program implementing the Integration Broker OnRequest method to check the status of an optimization engine. The OPT_CALL example executes the CheckOptEngineStatus function.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Running a Transaction

To run a transaction, structure the rowset as follows, using the PT_OPTPARMS record.

Copyright © 1988-2007, Oracle. All rights reserved.

PARMKEY Field	VALUE Field
OPTCMD	RUN Causes the PeopleCode program implementing the Integration Broker OnRequest method to run an optimization transaction. The OPT_CALL example executes the GetOptEngine method and either the RunSynch or RunAsynch method.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.
SYNCH	Y for a synchronous transaction, N for asynchronous.
TRANSACTION	The name of the transaction to run.
The names of one or more transaction parameters.	The value of each named transaction parameter.

Setting the Trace Level

To set a trace level, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	SET_TRACE_LEVEL Causes the PeopleCode program implementing the OnRequest Integration Broker method to set the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the SetTraceLevel method.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Copyright © 1988-2007, Oracle. All rights reserved.

PARMKEY Field	VALUE Field
COMPONENT	One of the following values: • %Opt_Engine — server activity of the running optimization engine. • %Opt_Utility — low level elements that support the running optimization engine. • %Opt_Datacache — the in-memory database cache. • %Opt_Plugin — the plugin being used for the running optimization engine.
SEVERITY_LEVEL	The severity level to log. The following list starts with the most severe level; the level you specify includes all higher levels. For example, if you specify %Severity_Error, it logs %Severity_Fatal, %Severity_Status, and %Severity_Error messages and filters out the others. • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2

Getting the Trace Level

To get a trace level, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	GET_TRACE_LEVEL Causes the PeopleCode program implementing the OnRequest Integration Broker method to get the severity level at which events will be logged for an optimization engine. The OPT_CALL example executes the GetTraceLevel method.
PROBINST	Set to the name of the analytic instance.
PROCINSTANCE	Set to the name of the process instance for this process scheduler job.
COMPONENT	One of the following values: • %Opt_Engine — server activity of the running optimization engine. • %Opt_Utility — low level elements that support the running optimization engine. • %Opt_Datacache — the in-memory database cache. • %Opt_Plugin — the plugin being used for the current opt engine.

Copyright © 1988-2007, Oracle. All rights reserved.

Shutting Down an Optimization Engine

To shut down an optimization engine, structure the rowset as follows, using the PT_OPTPARMS record.

PARMKEY Field	VALUE Field
OPTCMD	SHUTDOWN Causes the PeopleCode program implementing the OnRequest Integration Broker method to shut down an optimization engine. The OPT_CALL example executes the Shutdown method.
PROBINST	The name of the analytic instance.
PROCINSTANCE	The name of the process instance for this process scheduler job.

Creating a Response Message

This section provides an overview of the response message and describes how to create messages that:

- Send optimization status.
- Send a detailed message.

Understanding the Response Message

For optimization, the message PeopleCode in application server receives the request messages, performs an optimization actions, and sends response OPT_CALL messages. One message uses rowsets built from PT_OPTPARMS records, the other uses rowsets from PT_DETMSGS records. You can use the following rowset structures as an example of how to send responses from the message notification PeopleCode in the application server to the application engine program in the process scheduler.

Sending Optimization Status

To send the status of the optimization functions and methods called within the PeopleCode program implementing the OnRequest Integration Broker method, structure the rowset as follows using the PT_OPTPARMS record. The optimization functions and messages are called in response to the request input message. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

PARMKEY Field	VALUE Field
STATUS	The return status of the optimization function or method that is called in the message PeopleCode.
DETAILED_STATUS	The optional detailed status returned by many of the optimization functions and methods.

Sending a Detailed Message

To send a detailed message, structure the rowset as follows, using the PT_DETMSGS record. You set key values using the PARMKEY field, and then set a value for that key field in the VALUE field.

Copyright © 1988-2007, Oracle. All rights reserved.

PARMKEY Field	VALUE Field
MSGSET	The message set number. In the case of optimization, the message set number is 148.
MSGNUM	The name of the detailed message.
PARMCOUNT	The number of message parameters for the detailed message. There can be up to five parameters.
MSGPARM1	The first parameter value.
MSGPARM2	The second parameter value.
MSGPARM3	The third parameter value.
MSGPARM4	The fourth parameter value.
MSGPARM5	The fifth parameter value.

Editing the Request PeopleCode

The PT_OPTCALL Application Engine program serves as a template. It is delivered with all the sections marked as inactive. You can edit the program to suit your needs, then mark the appropriate sections active before running it. You can also use the program as a guide to creating your own Application Engine program.

The program uses these steps to send request messages to perform the following tasks:

1. Load the optimization engine.
2. Wait for the optimization engine load to finish.
3. Run an optimization transaction against the loaded optimization engine.
4. Wait for the optimization transaction to finish running.
5. Set the trace level.
6. Get the trace level.
7. Shut down the optimization engine.

You can edit steps 1 and 3 to run an optimization transaction. You can also use the entire program as a template to create your own Application Engine program.

Loading an Optimization Engine

In step 1, enter the name of your analytic instance. In this example, the name of the analytic instance is FEMALE1.

If you have multiple domains, enter the local node name and the machine name and port number for your application server. In this case, the local node name is %LocalNode and the machine name and port number are foo111111:9000.

```
Local Message &MSG;
Local Message &response;

Component string &probid;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Component string &isSync;
Component string &procinst;
Local integer &nInst;
Local string &url;

Local Rowset &rs;
Local Row &row;
Local Record &rec;

Local string &stName;
Local integer &stVal;

&MSG = CreateMessage(OPERATION.OPT_CALL);
&rs = &MSG.GetRowset();

&row = &rs.GetRow(1);
&rec = &row.GetRecord(Record.PT_OPTPARMS);
&rec.PARMKEY.Value = "OPTCMD";
&rec.VALUE.Value = "CREATE";

&rs.InsertRow(1);
&rec = &rs.GetRow(2).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROBINST";
&rec.VALUE.Value = "FEMALE1";
&probid = "FEMALE1";

&rs.InsertRow(2);
&rec = &rs.GetRow(3).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROCINSTANCE";
&nInst = Record.PT_OPT_AET.PROCESS_INSTANCE.Value;
&rec.VALUE.Value = String(&nInst);
&procinst = String(&nInst);

&rs.InsertRow(3);
&rec = &rs.GetRow(4).PT_OPTPARMS;
&rec.PARMKEY.Value = "SYNCH";
&rec.VALUE.Value = "N";
&isSync = "N";


/* Specify the Application Server domain URL (foo111111:9000 in this example)
*/
&response = %IntBroker.SyncRequest(%LocalNode, "//foo111111:9000 e");

If &response.ResponseStatus = 0 Then
   &stName = &response.GetRowset().GetRow(1).GetRecord(Record.PT_OPTPARMS).Get⇒
Field(Field.PARMKEY).Value;
   &stVal = Value(&response.GetRowset().GetRow(1).GetRecord(Record.PT_⇒
OPTPARMS).GetField(Field.VALUE).Value);
   If &stName = "STATUS" And
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
         &stVal = %OptEng_Fail Then
      /* Check detailed message here */
      throw CreateException(148, 2, "Can not send to OptEngine");
   End-If;
End-If;
```

Running An Optimization Transaction

In step 3, enter the name of your optimization transaction and its parameter name/value pairs. In this example, the transaction name is TEST_LONG_TRANS, the first parameter name/value pair is Delay_in_Secs and 30, and the second parameter name/value pair is Sleep0_Work1 and 0.

The parameter values are stored as strings. You may need to convert them in the OnRequest PeopleCode.

```
Local Message &MSG;
Local Message &response;

Local Rowset &rs, &respRS;
Local Row &row;
Local Record &rec, &msgRec;

Component string &probid;
Component string &procinst;
Component string &isSync;
Local string &url = "";
Local integer &parmCount, &msgSet, &msgNum;

&MSG = CreateMessage(OPERATION.OPT_CALL);
&rs = &MSG.GetRowset();

&row = &rs.GetRow(1);
&rec = &row.GetRecord(Record.PT_OPTPARMS);
&rec.PARMKEY.Value = "OPTCMD";
&rec.VALUE.Value = "RUN";

&rs.InsertRow(1);
&rec = &rs.GetRow(2).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROBINST";
&rec.VALUE.Value = &probid;

&rs.InsertRow(2);
&rec = &rs.GetRow(3).PT_OPTPARMS;
&rec.PARMKEY.Value = "PROCINSTANCE";
&rec.VALUE.Value = &procinst;

&rs.InsertRow(3);
&rec = &rs.GetRow(4).PT_OPTPARMS;
&rec.PARMKEY.Value = "SYNCH";
&rec.VALUE.Value = &isSync;

&rs.InsertRow(4);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&rec = &rs.GetRow(5).PT_OPTPARMS;
&rec.PARMKEY.Value = "TRANSACTION";
&rec.VALUE.Value = "TEST_LONG_TRANS";

&rs.InsertRow(5);
&rec = &rs.GetRow(6).PT_OPTPARMS;
&rec.PARMKEY.Value = "Delay_in_Secs";
&rec.VALUE.Value = "30";

&rs.InsertRow(6);
&rec = &rs.GetRow(7).PT_OPTPARMS;
&rec.PARMKEY.Value = "Sleep0_Work1";
&rec.VALUE.Value = "0";

/* SyncRequest will carry a url */
SQLExec("select URL from PSOPTSTATUS where PROBINST=:1 AND URL NOT LIKE '%:0';",⇒
 &probid, &url);
If &url = "" Then
   throw CreateException(148, 2, "Can not send to OptEngine");
End-If;

/* Specify the Application Server domain URL.
   (This was specified in Step 1 in this example.)
*/
&response = %IntBroker.SyncRequest(%LocalNode, &url);

If &response.ResponseStatus = 0 Then
   &stName = &response.GetRowset().GetRow(1).GetRecord(Record.PT_OPTPARMS).Get⇒
Field(Field.PARMKEY).Value;
   &stVal = Value(&response.GetRowset().GetRow(1).GetRecord(Record.PT_⇒
OPTPARMS).GetField(Field.VALUE).Value);

   If &stName = "STATUS" And
         &stVal = %OptEng_Fail Then
      throw CreateException(148, 2, "Can not send to OptEngine");
   End-If;

   /* Check Detailed msg here */
   If &isSync = "Y" And
         &stVal = %OptEng_Success Then

      &respRS = &response.GetRowset();
      &rowNum = &respRS.ActiveRowCount;
      For &iloop = 1 To &rowNum
         &msgRec = &respRS.GetRow(&iloop).GetRecord(Record.PT_OPTDETMSGS);
         If (&msgRec.GetField(Field.MSGSET).Value <> 0) Then
            &msgSet = Value(&msgRec.GetField(Field.MSGSET).Value);
            &msgNum = Value(&msgRec.GetField(Field.MSGNUM).Value);
            &parm1 = &msgRec.GetField(Field.MSGPARM1).Value;
            &parm2 = &msgRec.GetField(Field.MSGPARM2).Value;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
            &parm3 = &msgRec.GetField(Field.MSGPARM3).Value;
            &parm4 = &msgRec.GetField(Field.MSGPARM4).Value;
            &parm5 = &msgRec.GetField(Field.MSGPARM5).Value;
            &string = MsgGetText(&msgSet, &msgNum, "Message Not Found", &parm1,⇒
 &parm2, &parm3, &parm4, &parm5);

         End-If;
      End-For;

   End-If;

End-If;
```

Editing the Response PeopleCode

The OPT_CALL message definition serves as a template. It is delivered to work with the PT_OPTCALL Application Engine program. You can edit the program to suit your needs, or use it as a guide when creating your own response message program.

OPT_CALL Message Program

The OPT_CALL application package implements the Integration Broker method OnRequest. The PeopleCode in this method shows application messages for lights-out mode.

Depending upon the request message, the OnRequest method PeopleCode calls appropriate optimization functions and methods to perform these tasks, and sends a response message containing the returned status and detailed messages from the optimization functions and methods.

You can use the OnRequest method PeopleCode as a template to create your own response message PeopleCode program. For example, you can edit it to run an optimization transaction, which is shown below as an example. This example is edited to match the examples for step 1 and step 3 in the PT_OPTCALL program.

Processing the Transaction Parameters

Edit the OPT_CALL application program OnRequest method to enter the name of your optimization transaction and the name/value pairs for its parameters. In this example, the transaction name is TEST_LONG_TRANS, the first parameter name/value pair is &delayParm and &delay (maps to Delay_in_Secs from the request message), and the second parameter name/value pair is &sleepParm and &isSleep (maps to Sleep0_Work1 from the request message).

The parameter values are stored as strings in step 3 of the Application Engine program. You may need to convert them here to your desired format. Here is a section of the application program showing the places to edit.

```
If &trans = "TEST_LONG_TRANS" Then
   &REC = &rs.GetRow(6).PT_OPTPARMS;
   &delayParm = &REC.PARMKEY.Value;
   &delay = Value(&REC.VALUE.Value);

   &REC = &rs.GetRow(7).PT_OPTPARMS;
   &sleepParm = &REC.PARMKEY.Value;
   &isSleep = Value(&REC.VALUE.Value);

   &myopt = GetOptEngine(&inst, &detStatus);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   If (&myopt = Null) Then
      &optstatus = %OptEng_Fail;
   End-If;

   If &myopt <> Null And &isSync = "Y" Then
      &optstatus = &myopt.RunSynch(&trans, &delayParm, &delay, &sleepParm, &isSleep⇒
);
      &detStatus = &myopt.DetailedStatus;
      End-If;

   If &myopt <> Null And &isSync = "N" Then
      &myopt.ProcessInstance = &procInst;
      &optstatus = &myopt.RunASynch(&trans, &delayParm, &delay, &sleepParm, &is⇒
Sleep);
      &detStatus = &myopt.DetailedStatus;
   End-If; /* iif myopt=null */
End-If;
```

Building a Status Response Message

This section shows the a response message to send a status message for the OPT_CALL message in the application server.

```
/* Insert detailed status and detailed msgs into Response msg rowset */
&respRS = &response.GetRowset();
&respRS.GetRow(1).GetRecord(Record.PT_OPTPARMS).GetField(Field.PARMKEY).Value =⇒
 "STATUS";
&respRS.GetRow(1).GetRecord(Record.PT_OPTPARMS).GetField(Field.VALUE).Value =⇒
 String(&optstatus);

&respRS.InsertRow(1);
&respRS.GetRow(2).GetRecord(Record.PT_OPTPARMS).GetField(Field.PARMKEY).Value =⇒
 "DETAILED_STATUS";
&respRS.GetRow(2).GetRecord(Record.PT_OPTPARMS).GetField(Field.VALUE).Value =⇒
 String(&detStatus);
```

Building a Detailed Response Message

This section shows a response message to send a detailed message for the OPT_CALL message on the application server.

```
/*Either optcmd or inst is not passed in correctly, or optengine is not loaded⇒
/created correctly */
If &myopt = Null Then
   &msgRec = &respRS.GetRow(1).GetRecord(Record.PT_OPTDETMSGS);
   If &isParmBad = True Then
      &msgRec.GetField(Field.MSGSET).Value = 148;
      &msgRec.GetField(Field.MSGNUM).Value = 505;
   End-If;
End-If;

/* If it is sync transaction, insert DetailMsg to response msg */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
If &myopt <> Null And
      &isSync = "Y" And
      &optcmd = "RUN" And
      &optstatus = %OptEng_Success Then
   &arrArray = &myopt.DetailMsgs;
   For &iloop = 1 To &arrArray.Len
      /* First two rows have been inserted because of PT_OPTPARMS for two status⇒
 codes */
      If &iloop > 2 Then
         &respRS.InsertRow(&iloop - 1);
      End-If;

      &msgRec = &respRS.GetRow(&iloop).GetRecord(Record.PT_OPTDETMSGS);
      &msgRec.GetField(Field.MSGSET).Value = String(&arrArray [&iloop][1]);
      &msgRec.GetField(Field.MSGNUM).Value = String(&arrArray [&iloop][2]);
      &msgRec.GetField(Field.PARMCOUNT).Value = String(&arrArray [&iloop][3]);
      &msgRec.GetField(Field.MSGPARM1).Value = String(&arrArray [&iloop][4]);
      &msgRec.GetField(Field.MSGPARM2).Value = String(&arrArray [&iloop][5]);
      &msgRec.GetField(Field.MSGPARM3).Value = String(&arrArray [&iloop][6]);
      &msgRec.GetField(Field.MSGPARM4).Value = String(&arrArray [&iloop][7]);
      &msgRec.GetField(Field.MSGPARM5).Value = String(&arrArray [&iloop][8]);
   End-For;

End-If;
```

Optimization Built-in Functions

This section discusses the optimization functions. The functions are discussed in alphabetical order.

CreateOptEngine

Syntax

CreateOptEngine(*analytic_inst*, {%Synch | %ASynch}[, *&detailedstatus*] [, *processinstance*])

Description

The CreateOptEngine function instantiates an OptEngine object, loads an optimization engine with an analytic instance and returns a reference to it.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Analytic_inst	Specify the analytic instance ID, which is a unique ID for this analytic instance in this optimization engine. This is supplied by users when they request that an optimization be run.
%Synch | %Asynch	Specify whether the optimization engine is synchronous or asynchronous. The values are: • %Synch: run the optimization engine synchronously. • %Asynch: run the optimization engine asynchronously.
&detailedstatus	Specify a variable that the engine uses to give further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Aiid: The analytic instance ID passed to the function is invalid. • %OptEng_Exists: An optimization engine instance already exists and is loaded. • %OptEng_Method_Disabled: A method is disabled or not valid. • %OptEng_DB_Updates_Pending: indicates that database updates are pending.
processinstance	Enter the process instance id. You use this parameter only with lights-out processing, most likely with the subscription PeopleCode for application message. **Note.** This optional parameter is positional. If you use it, you must also use the *&detailedstatus* parameter. See Chapter 25, "Optimization PeopleCode," Using Lights-Out Mode with Optimization, page 920. The state record that you use with Application Engine contains the process instance id. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Engine*, "Developing Efficient Programs," Using State Records.

Returns

If successful, CreateOptEngine returns an OptEngine PeopleCode object. If the function fails, it returns a null value. Examine the optional status reference parameter in case of a Null return for additional information regarding the failure.

Example

An OptEngine object variable can be scoped as Local, Component, or Global.

You declare OptEngine objects as type OptEngine. For example:

```
Local OptEngine &MyOptEngine;
Component OptEngine &MyOpt;
Global OptEngine &MyOptEng;
```

The following example loads an optimization engine with the analytic instance:

```
Local OptEngine &myopt;
Local string &probinst;
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
Local string &transaction;
Local integer &detailedstatus;

&probinst = GetRecord(Record.PSOPTPRBINST).GetField(Field.PROBINST).Value;
&myopt = CreateOptEngine(&probinst, %Synch);
```

The following example shows the use of the optional status parameter:

```
&myopt = CreateOptEngine(&probinst, %Synch, &detailedstatus);
if &myopt = Null then
    if &detailedstatus = %OptEng_Invalid_Piid then
        /*perform some action */
    end_if;
end_if;
```

CreateOptInterface

Syntax

CreateOptInterface()

Description

The CreateOptInterface function instantiates an OptInterface object.

Note. You can use this function — and the OptInterface methods — only within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class. This ensures that the OptInterface PeopleCode runs only on the optimization engine.

Parameters

None.

Returns

If successful, CreateOptInterface returns an OptInterface PeopleCode object. If the function fails, it returns a null value.

Example

You declare OptInterface objects as type OptInterface. For example:

```
Local OptInterface &MyOptInterface;
Component OptInterface &MyOptInt;
Global OptInterface &MyOptInt;
```

The following example instantiates an OptInterface object:

```
Local OptInterface &myInterface;
Int &status;

&myInterface = CreateOptInterface(&addtionalStatus);
if (&myInterface != NULL) then
    &status = &myInterface.ActivateModel("RMO_TEST");
        if (&status = %OptInter_Fail) then
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
            /* examine &myInterface.DetailedStatus for reason */
            ...
        end-if;
else
    /* CreateOptInterface has returned NULL */
    /* take some corrective action here */
        ...
end_if;
```

DeleteOptProbInst

Syntax

DeleteOptProbInst(*probinst*[, *&detailedstatus*])

Description

The DeleteOptProbInst function deletes the analytic instance ID from PeopleTools metadata. This function can be called only inside FieldChange, PreSaveChange and PostSaveChange PeopleCode events, and in Workflow.

Note. Use this function to delete the analytic instance ID after deleting data in optimization application tables for this analytic instance.

Parameters

Parameter	Description
probinst	Enter the analytic instance ID to delete.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid. • %OptEng_Sql_Exception: A SQLerror is encountered when access database. • %OptEng_Exists: An analytic server loaded with this analytic instance still exists.

Returns

Returns %OptEng_Success if successful; otherwise returns %OptEng_Fail.

Example

The following example deletes the instance for an analytic type:

Note. Whenever you add records to an analytic type, you must call DeleteOptProbInst to delete the old analytic type instances and then call InsertOptProbInst to recreate them.

```
Local string &probinst;
Local string &probtype;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local integer &ret;
&probinst = "PATSMITH";
&probtype = "QEOPT";
&ret = DeleteOptProbInst(&probinst, &probtype);
If &ret <> %OptEng_Success  Then
   QEOPT_WRK.MESSAGE_TEXT = "Delete of analytic instance " | &probinst | "⇒
 failed.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance " | &probinst | " deleted.";
End-If;
```

The following example shows the use of the optional status parameter:

```
Local integer &detailedstatus;
&ret = DeleteOptProbInst(&probinst, &probtype, &detailedstatus);
If &ret <> %OptEng_Success  AND &detailedstatus=%OptEng_Invalid_Piid then
   QEOPT_WRK.MESSAGE_TEXT = "Delete of analytic instance " | &probinst | " failed
for bad piid.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance " | &probinst | deleted.";
End-If;
```

GetOptEngine

Syntax

GetOptEngine(*probinst*[,*&detailedstatus*])

Description

The GetOptEngine function returns a handle to an optimization engine that is already loaded with the analytic instance.

Note. You cannot call GetOptEngine from a domain other than the application server.

Parameters

Parameter	Description
probinst	Enter the analytic instance ID, which is unique ID for this analytic instance in this optimization engine.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid.

Returns

Returns an OptEngine PeopleCode object if successful, a null value otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example causes an optimization engine to shut down its analytic instance:

```
Global string &probinst;
Local OptEngine &myopt;
Local integer &status;

&myopt = GetOptEngine(&probinst);
If &myopt <> NULL then
&status = &myopt.ShutDown();
QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance ID " | &probinst
 | " has been shutdown successfully.";
End-if;
```

Or, you can use the optional status parameter:

```
&myopt = GetOptEngine(&probinst, &detailedstatus);
if &myopt=NULL and &detailedstatus=%OptEng_Invalid_Piid then
     /* perform some action */
End-if;
```

GetOptProbInstList

Syntax

GetOptProbInstList(*ProblemType* , *OutputErrorCode* [, *Prefix*] [, *&detailedstatus*])

Description

The GetOptProbInstList function gets the list of all analytic instance IDs in an analytic type.

Parameters

Parameter	Description
ProblemType	Enter the name of the analytic type that you created in PeopleSoft Application Designer.
OutputErrorCode	Future use. Always returns zero.
Prefix	(Optional) Enter a string. If used, this prefix causes the returned list to include only the analytic instance IDs that start with this prefix. If not used, all the analytic instance IDs in the analytic type are returned.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic type name passed to the function is invalid.

Returns

Returns an array of strings containing the optimization analytic instance list.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example shows the usage of GetOptProbInstList to fill the display field on a page:

```
Global string &probinst;
Local integer &detailedstatus;
Local integer &iloop;
Local array of string &instarray;

QEOPT.OPERATOR = %UserId;

&instarray = GetOptProbInstList(QEOPT.PROBTYPE, &ret, &detailedstatus);

If &ret <> %OptEng_Success Then
   QEOPT_WRK.MESSAGE_TEXT = "Could not get analytic instances
 for analytic type " | QEOPT.PROBTYPE ;
Else
   For &iloop = 1 To &instarray.Len
      QEOPT_WRK.MESSAGE_TEXT = QEOPT_WRK.MESSAGE_TEXT | &instarray[&iloop] | " ";
   End-For;
End-If;
```

The following example shows the use of the optional status parameter:

```
&instarray = GetOptProbInstList(QEOPT.PROBTYPE, &ret, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   QEOPT_WRK.MESSAGE_TEXT = "Could not get analytic instances for analytic type "
| QEOPT.PROBTYPE | "because bad piid" ;
End-If;
```

InsertOptProbInst

Syntax

InsertOptProbInst(*probinst, ProblemType*[, *&detailedstatus*] [,*Description*])

Description

The InsertOptProbInst function inserts a new analytic instance ID into the PeopleTools metadata.

The InsertOptProbInst function can be called only inside FieldChange, PreSave and PostSave PeopleCode events, and in Workflow.

Note. You must use this function to create the analytic instance ID before inserting data into optimization application tables for this analytic instance.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
probinst	Enter the analytic instance ID to be inserted into the analytic type.
ProblemType	Enter the name of the analytic type that you created in PeopleSoft Application Designer.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Fail: The function failed. • %OptEng_Invalid_Piid: The analytic instance ID passed to the function is invalid.
Description	(Optional) Specify a description for the analytic instance. This parameter takes a string value.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local string &probinst;
Local string &probtype;
Local integer &ret;
Local integer &detailedstatus;

&probinst = "PATSMITH";
&probtype = "QEOPT";
&probDescr = "New QEOPT instance";
&ret = InsertOptProbInst(&probinst, &probtype, &probDescr);
If &ret <> %OptEng_Success Then
   QEOPT_WRK.MESSAGE_TEXT = "Insert of analytic instance "
 | &probinst | " failed.";
Else
   QEOPT_WRK.MESSAGE_TEXT = "Analytic Instance " | &probinst | " created.";
End-If;
```

The following example shows the use of the optional status parameter:

```
&ret = InsertOptProbInst(&probinst, &probtype, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   QEOPT_WRK.MESSAGE_TEXT = "Insert of analytic instance "
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
 | &probinst | " failed for bad piid.";
End-if;
```

IsValidOptProbInst

Syntax

IsValidOptProbInst(*probinst* [, &*detailedstatus*])

Description

IsValidOptProbInst determines if a given analytic instance exists in the optimization metadata.

Parameters

Parameter	Description
probinst	Enter the analytic instance ID to be validated.
&detailedstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • %OptEng_Success: The function completed successfully. • %OptEng_Invalid_Piid: The analytic type name passed to the function is invalid.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local string &probinst;
Local integer &detailedstatus;
Local integer &ret;

&probinst = "PATSMITH";
&ret = IsValidOptProbInst(&probinst, &detailedstatus);
If &ret <> %OptEng_Success and &detailedstatus=%OptEng_Invalid_Piid Then
   <perform some action>
End-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

OptEngine Class Methods

This section discusses the optimization methods for the OptEngine PeopleCode class. The methods are listed in alphabetical order.

CheckOptEngineStatus

Syntax

```
CheckOptEngineStatus()
```

Description

The CheckOptEngineStatus method returns the status of the optimization engine, using a combination of its return value and the DetailedStatus OptEngine class property. Keep the following in mind:

- The value returned by CheckOptEngineStatus is the operational status of the optimization engine.
- The DetailedStatus property indicates the completion status of the OptEngine method call CheckOptEngineStatus.

For example, CheckOptEngineStatus can return %OptEng_Idle and DetailedStatus is %OptEng_Success. For CheckOptEngineStatus, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Not_Available

Note. Before this method is called, the CreateOptEngine or GetOptEngine must be called.

Returns

Returns an integer for the status of the optimization engine. These numbers are message IDs belonging to message set 148 in the message catalog.

Numeric Value	Constant Value	Description
21	%OptEng_Not_Loaded	The optimization engine process is running, but is not currently loaded with an application problem.
22	%OptEng_Busy_Loading	The optimization engine is busy loading an application problem. It will not accept transaction requests until loading completes.
23	%OptEng_Idle	The optimization engine is loaded with an application problem and waiting for a transaction request.
24	%OptEng_Busy	The optimization engine is busy processing a transaction request for the loaded application problem. It will not accept additional transaction requests until the current one completes.
26	%OptEng_Unknown	An error has occurred. The optimization engine status cannot be determined.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

This PeopleCode example shows optimization engine status being checked:

```
Local OptEngine &myopt;
Local string &probinst;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Initialize the DESCRLONG field in the QE_FUNCLIB_OPT record to null.  */
GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = "";
&status = &myopt.CheckOptEngineStatus();
GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = "Opt⇒
Engine status = " | MsgGet(148, &status, "Could not send to the OptEngine.");
```

You can also retrieve the detailed status:

```
Local integer &detailedstatus
&status = &myopt.CheckOptEngineStatus();
&detailedstatus = &myopt.DetailedStatus;
```

FillRowset

Syntax

FillRowset(*PARAM_NAME, &Rowset*[, *&functionstatus*])

Description

This method gets the value of a transaction output parameter that is a rowset. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

When using the OptEngine DetailedStatus property, keep the following in mind:

- The value returned by FillRowset is the operational status of the optimization engine.
- The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call FillRowset.

For example, FillRowset returns %OptEng_Fail, and DetailedStatus is %OptEng_Method_Disabled.

For FillRowset, the DetailedStatus property can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled.

  This indicates that the method is disabled or not valid.
- %OptEng_Wrong_Parm_Type

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.
&Rowset	Enter the rowset containing the values. This rowset must be a single record rowset, and the record must match the record name associated with the transaction parameter in the analytic type definition.
&functionstatus	(Optional) This status reference parameter returns an integer value giving further information about the evaluation of this function. The value returned is one of the following: • OptEng_Success: The function completed successfully. • OptEng_Fail: The function failed. • OptEng_Method_Disabled: A method is disabled or not valid.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

The following PeopleCode example runs a synchronous optimization transaction named RETURN_MACHINE_UNAVAILABLE. It has these parameters:

- Input: MACHINE_NAME to specify the machine for which we need unavailable times.
- Output: RETURN_TIMES to specify a rowset and MACHINE_WRK record containing the BEGIN_DATE and END_DATE fields.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local Rowset &rs;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
/* Run the RETURN_MACHINE_UNAVAILABLE transaction synchronously with input values.⇒
  */
&status = &myopt.RunSynch("RETURN_MACHINE_UNAVAILABLE", "MACHINE_NAME", &machname);
If Not &status Then
   QEOPT_WRK.MESSAGE_TEXT = " RETURN_MACHINE_UNAVAILABLE transaction failed.";
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
   Return;
End-If;
/* Get output value from the RETURN_MACHINE_UNAVAILABLE transaction.  */
&rs = CreateRowset(Record.MACHINE_WRK);
&status = &myopt.FillRowset("RETURN_TIMES", &rs);
```

You can also use the [new->] DetailedStatus property as follows:

```
&status = &myopt.FillRowset("RETURN_TIMES", &rs);
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
    /* perform some action */
End-if;
```

GetDate

Syntax

GetDate(*PARAM_NAME*[, *&status*])

Description

This method gets the value of a transaction output parameter with a data type of Date. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call GetDate. For GetDate, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a Date object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetNumber, page 948.

GetDateArray

Syntax

GetDateArray(*PARAM_NAME*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

This method gets the value of a transaction output parameter with a data type Array of Date. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The OptEngine DetailedStatus property indicates the completion status of the OptEngine method call GetDateArray. For GetDateArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of Date object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetStringArray, page 950.

GetDateTime

Syntax

GetDateTime(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of DateTime. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetDateTime. For GetDateTime, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a DateTime object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetNumber, page 948.

GetDateTimeArray

Syntax

GetDateTimeArray(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type Array of DateTime. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetDateTimeArray. For GetDateTimeArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of DateTime object; use this method when that is the data type of the transaction output parameter value.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

See Chapter 25, "Optimization PeopleCode," GetStringArray, page 950.

GetNumber

Syntax

GetNumber(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of Number. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetNumber. For GetNumber, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a Number object; use this method when that is the data type of the transaction output parameter value.

Example

The following PeopleCode example runs a synchronous optimization transaction named IS_MACHINE_AVAILABLE. It has these parameters:

- Input MACHINE_NAME to specify the machine.
- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output AVAILABLE_FLAG to specify whether the machine is available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&machname = QEOPT_WRK.MACHINE_NAME.Value;
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the IS_MACHINE_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("IS_MACHINE_AVAILABLE", "MACHINE_NAME",
 &machname, "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If Not &status Then
   QEOPT_WRK.MESSAGE_TEXT = "IS_MACHINE_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the IS_MACHINE_AVAILABLE transaction.  */
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
```

You can use the DetailedStatus property as follows:

```
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
if &myopt.DetailedStatus=%OptEng_Fail then
   /* perform some action */
End-if;
```

GetNumberArray

Syntax

GetNumberArray(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type Array of Number. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetNumberArray. For GetNumberArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled

  This indicates that the method is disabled or not valid.

Note. Do not pass an array of type Integer as a transaction parameter. Use an array of type Number instead.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

Returns an Array of Number object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetStringArray, page 950.

GetString

Syntax

GetString(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of String. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetString. For GetString, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a String object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetNumber, page 948.

GetStringArray

Syntax

GetStringArray(*PARAM_NAME*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

This method gets the value of a transaction output parameter with a data type Array of String. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetStringArray. For GetStringArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of String object; use this method when that is the data type of the transaction output parameter value.

Example

The following PeopleCode example runs a synchronous optimization transaction named ARE_MACHINES_AVAILABLE. It has these parameters:

- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output MACHINE_NAMES to specify the machines available in that time slot.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local array of string &machnames;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the ARE_MACHINES_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("ARE_MACHINES_AVAILABLE",
 "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "ARE_MACHINES_AVAILABLE transaction failed.";
   Return;
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Get output value from the ARE_MACHINES_AVAILABLE transaction.  */
&machnames = &myopt.GetStringArray("MACHINE_NAMES");
```

The following example shows the use of the DetailedStatus property:

```
Local array of string &machnames;
&machnames = &myopt.GetStringArray("MACHINE_NAMES");
if &myopt.DetailedStatus=%OptEng_Fail then
   /* perform some action */
End-if;
```

GetTime

Syntax

GetTime(*PARAM_NAME*)

Description

This method gets the value of a transaction output parameter with a data type of Time. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTime. For GetTime, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns a Time object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetNumber, page 948.

GetTimeArray

Syntax

GetTimeArray(*PARAM_NAME*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

This method gets the value of a transaction output parameter with a data type Array of Time. This cannot be used with the RunAsynch method; RunSynch is needed to make the transaction output parameter values immediately available.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTimeArray. For GetTimeArray, DetailedStatus can have the value:

- %OptEng_Success.
- %OptEng_Fail.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.

Parameters

Parameter	Description
PARAM_NAME	Enter a string for the name of the output parameter to get from the transaction that was just performed with RunSynch. This parameter must be defined as an output or both (input and output) in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

Returns an Array of Time object; use this method when that is the data type of the transaction output parameter value.

Example

See Chapter 25, "Optimization PeopleCode," GetStringArray, page 950.

GetTraceLevel

Syntax

GetTraceLevel(*component*)

Description

GetTraceLevel gets the severity level at which events are logged for a given component.

The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call GetTraceLevel. For GetTraceLevel, DetailedStatus can have the value:

- %OptEng_Success.

  This indicates that the function completed successfully.
- %OptEng_Fail.

  This indicates that the function failed.
- %OptEng_Method_Disabled.

  This indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending.

Copyright © 1988-2007, Oracle. All rights reserved.

This indicates that database updates are pending.

Parameters

Parameter	Description
component	Enter one of the following PeopleCode constants: Opt_Engine, Opt_Utility, Opt_Datacache, or Opt_Plugin.

Returns

Returns one of the following.

- %Severity_Fatal
- %Severity_Status
- %Severity_Error
- %Severity_Warn
- %Severity_Info
- %Severity_Trace1
- %Severity_Trace2

Example

```
Local OptEngine &myopt;
Local integer &tracelevel;

&myopt = GetOptEngine("PATSMITH");

&tracelevel = &myopt.GetTraceLevel(%Opt_Engine);
if &myopt.DetailedStatus = %OptEng_Success then

   if (&tracelevel = %Severity_Info_ then
      winmessage("Severity level for the OptEngine is 'Info'");
   End-if;
End-if;
```

RunAsynch

Syntax

RunAsynch(*TRANSACTION, PARM_PAIRS*)

Description

The RunAsynch method requests the optimization engine to run the transaction in asynchronous mode.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by RunASynch is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call RunASynch.

 Copyright © 1988-2007, Oracle. All rights reserved.

For example, RunASynch can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For RunASynch, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	Description
TRANSACTION	Enter a string for the name of the transaction to run.
PARAM_PAIRS	Enter the name and value pairs (string name and value) for this transaction. Not used if the transaction has no parameters. Parameters are defined in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

This PeopleCode example runs an asynchronous optimization transaction named SOLVE. It has no input or output parameters. The SOLVE transaction solves the exercise scheduling problem and puts the results into the QE_RWSM_EXERSCH table.

```
Local OptEngine &myopt;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Run the SOLVE transaction asynchronously with input values.  */
&status = &myopt.RunAsynch("SOLVE");
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "SOLVE transaction failed.";
   Return;
End-If;
```

The following example shows the use of the DetailedStatus property.

```
Local integer &status;
&status = myopt.RunAsynch("SOLVE");
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

RunSynch

Syntax

RunSynch(*TRANSACTION, PARM_PAIRS*)

Description

The RunSynch method requests the optimization engine to run the transaction in synchronous mode.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by RunSynch is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call RunSynch.

For example, RunSynch can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For RunSynch, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Parameters

Parameter	Description
TRANSACTION	Enter a string for the name of the transaction to run.
PARAM_PAIRS	Enter the name and value pairs (string name and value) for this transaction. Not used if the transaction has no parameters. Parameters are defined in the analytic type definition. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Transactions.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

The following PeopleCode example runs a synchronous optimization transaction named IS_MACHINE_AVAILABLE. It has these parameters:

- Input MACHINE_NAME to specify the machine.
- Inputs BEGIN_DATE and END_DATE to specify the time slot.
- Output AVAILABLE_FLAG to specify whether the machine is available in that time slot.

Copyright © 1988-2007, Oracle. All rights reserved.

This PeopleCode example sets input parameter values and gets an output parameter value:

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;
&myopt = GetOptEngine("PATSMITH");
&machname = QEOPT_WRK.MACHINE_NAME.Value;
&begindate = QEOPT_WRK.BEGIN_DATE.Value;
&enddate = QEOPT_WRK.END_DATE.Value;
/* Run the IS_MACHINE_AVAILABLE transaction synchronously with input values.  */
&status = &myopt.RunSynch("IS_MACHINE_AVAILABLE",
 "MACHINE_NAME", &machname, "BEGIN_DATE", &begindate, "END_DATE", &enddate);
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "IS_MACHINE_AVAILABLE transaction failed.";
   Return;
End-If;
/* Get output value from the IS_MACHINE_AVAILABLE transaction.  */
QEOPT_WRK.AVAILABLE_FLAG = &myopt.GetNumber("AVAILABLE_FLAG");
```

Or, the following example shows the use of the DetailedStatus property.

```
Local integer &status;
&status = myopt.RunSynch("SOLVE");
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

SetTraceLevel

Syntax

SetTraceLevel(*component*, *severity*)

Description

SetTraceLevel sets the severity level at which events are logged for a given component.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by SetTraceLevel is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call SetTraceLevel.

For example, SetTraceLevel can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For SetTraceLevel, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
component	Use one of the following PeopleCode constants: Opt_Engine, Opt_Utility, Opt_Datacache, or Opt_Plugin.
severity	Use one of the following PeopleCode constants. These options set the degree to which errors are logged. You can set the tracing levels differently for various parts of your program. This enables you to control the amount of trace information that your program generates. The following list shows the order of the severity, starting with the highest level. For example, %Severity_Error logs %Severity_Fatal, %Severity_Status, and %Severity_Error messages, while the system filters out other messages. Keep in mind that the higher the severity, the greater the performance overhead. • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

```
Local OptEngine &myopt;
Local integer &status;
Local string &machname;
Local datetime &begindate;
Local datetime &enddate;

&myopt = GetOptEngine("PATSMITH");

&status = &myopt.SetTraceLevel(%Opt_Engine, %Severity_Warn);
if &status = %OptEng_Fail then
   <example: notify user that set trace action has failed>
End-if;
```


Copyright © 1988-2007, Oracle. All rights reserved.

ShutDown

Syntax

ShutDown()

Description

The ShutDown method requests the optimization engine to shut down.

If the optimization engine cannot be contacted for shutdown, the return status is %OptEng_Fail and the DetailedStatus property is OptEng_Not_Available.

When using the DetailedStatus OptEngine property, keep the following in mind:

- The value returned by Shutdown is the operational status of the optimization engine.
- The DetailedStatus OptEngine property indicates the completion status of the OptEngine method call Shutdown.

For example, Shutdown can return %OptEng_Fail and DetailedStatus is %OptEng_DB_Updates_Pending. For Shutdown, DetailedStatus can have the value:

- %OptEng_Success: indicates that the function completed successfully.
- %OptEng_Fail: indicates that the function failed.
- %OptEng_Method_Disabled: indicates that the method is disabled or not valid.
- %OptEng_DB_Updates_Pending: indicates that database updates are pending.

Note. Before this method is called, CreateOptEngine or GetOptEngine must be called. Call ShutDown to shut down optimization engines even when running in PeopleSoft Application Engine.

Parameters

None.

Returns

This method returns a constant. Valid values are:

Value	Description
%OptEng_Success	Returned if method succeeds.
%OptEng_Fail	Returned if the method fails.

Example

This PeopleCode example shows an optimization engine being shut down:

```
Local OptEngine &myopt;
Local integer &status;
&myopt = GetOptEngine("PATSMITH");
/* Shut down the optimization engine */
&status = &myopt.ShutDown();
If &status=%OptEng_Fail Then
   QEOPT_WRK.MESSAGE_TEXT = "PATSMITH optimization engine shutdown failed.";
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   Return;
Else
   QEOPT_WRK.MESSAGE_TEXT = "PATSMITH optimization engine shutdown successful.";
   Return;
End-If;
```

The following example shows the use of the DetailedStatus property:

```
Local integer &status;
&status = myopt.ShutDown();
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

OptEngine Class Properties

This section lists the optimization properties for the OptEngine PeopleCode class. The properties are listed in alphabetical order.

DetailMsgs

Description

The DetailMsgs property returns a list of messages generated by an optimization engine. Use DetailMsgs after you use the RunAsynch and RunSynch methods to check the status messages for an optimization transaction.

If the transaction fails, detailed messages are automatically shown to the user. If the transaction succeeds, warnings and informational messages may be generated by the transaction. Use this property to retrieve those messages and make them available to the user.

DetailMsgs provides a two-dimensional array containing the message set ID, the message number in the message catalog, and any arguments. Each row in the two-dimensional array has the following structure:

1. Message set ID.
2. Message number.
3. Number of message arguments.
4. Argument1.
5. Argument2.
6. Argument3.
7. Argument4.
8. Argument5.

A maximum of five arguments is supported for each message.

Note. To hold the property value returned, you need to declare an array of array of type Any.

Note. Before this method is called, you must call CreateOptEngine or GetOptEngine.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local OptEngine &myopt;
Local integer &status;
Local string &piid;

Local string &string;
Local array of array of any &arrArray;

&NEWLINE = Char(10);
&string = "";

&piid = GetRecord(Record.PSOPTPRBINST).GetField(Field.PROBINST).Value;
&myopt = GetOptEngine(&piid);

&status = &myopt.RunSynch("TEST_TRANSACTION");

If (&status = %OptEng_Success) then

&arrArray = &myopt.DetailMsgs;
For &iloop = 1 To &arrArray.Len

   &string = &string | &NEWLINE | MsgGetText(&arrArray [&iloop][1] /*message set*⇒
/,
   &arrArray [&iloop][2] /*message id*/, "Message Not Found",&arrArray[&iloop][4],
   &arrArray [&iloop][5],&arrArray [&iloop][6],
&arrArray [&iloop][7],&arrArray[&iloop][8]);

End-For;

GetLevel0().GetRow(1).GetRecord(Record.QE_FUNCLIB_OPT).DESCRLONG.Value = &string;
End-If;
```

DetailedStatus

Description

The DetailedStatus property contains the detailed execution status of an OptEngine method after the method is executed.

Example

```
Local integer &status;
&status = myopt.ShutDown();
if &status=%OptEng_Fail and &myopt.DetailedStatus=%OptEng_Method_Disabled then
   <perform some action>
End-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

OptBase Application Class

This PeopleCode application class is part of the PT_OPT_BASE application package. It establishes the basic framework for developing PeopleCode that invokes the Optimization PeopleCode plug-in. To use the plug-in, you develop a application class that extends the OptBase application class. OptBase contains the following types of methods:

- A set of base methods that you can extend for the purpose of handling input and output parameters.

  You can use them within any method you develop that corresponds by name to a transaction in an analytic type definition. These methods apply to the parameters that are defined for the transaction in the analytic type.

- A set of abstract placeholder methods that you can use to implement callback capability.

  You must extend these if you designate one or more records as callback records in your analytic type definition, even if you don't add any functionality to the methods.

- An abstract placeholder method, Init, that you can extend if you want to do any preprocessing before your first Optimization PeopleCode plug-in transaction runs.

Note. The analytic type definition to which these methods apply is the one that specifies this derived application class.

The CreateOptInterface function is the only optimization built-in function that you can use within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class.

Optbase Callback Methods

PeopleSoft Optimization Framework has a built-in callback functionality — when the OptInterface PeopleCode calls an Optimization PeopleCode plug-in transaction, it first determines whether you designated one or more records in your analytic type definition as callback records. For each callback record, the framework determines if any the record's database rows have been inserted, deleted, or updated since the optimization datacache was populated. If any changes have occurred, the framework propagates those changes to the datacache before invoking the transaction.

PeopleSoft provides methods that the framework uses to apply its callback functionality. In combination with the framework's callback changes, you might want to perform additional processing for your own purposes, including updating any derived data structures that are used by your optimization application. You can accomplish this by extending the callback methods and adding your own PeopleCode. Each callback method launches under different circumstances.

Note. Don't call any of these methods in your own PeopleCode. They're called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run within each method.

Following is a list of the abstract callback placeholder methods documented as part of the PT_OPT_BASE:OptBase application class:

- OptInsertCallback

  This method launches when the framework propagates to the datacache any database insertions encountered for a callback record.

- OptDeleteCallback

 Copyright © 1988-2007, Oracle. All rights reserved.

This method launches when the framework propagates to the datacache any database deletions encountered for a callback record.

- OptPreUpdateCallback

  This method launches before the framework propagates each database update encountered for a callback record.

- OptPostUpdateCallback

  This method launches after the framework propagates each database update encountered for a callback record.

- OptRefreshCallback

  This method launches after the framework propagates all database deletions, insertions, and updates encountered for all callback records.

Important! If any record in your analytic type definition is designated a callback record, you must ensure that you extend all of the callback methods in your extended class, even if each extended method contains only a Return statement. Otherwise your Optimization PeopleCode plug-in will fail.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Analytic Type Records.

OptBase Class Methods

This section discusses the abstract base class placeholder methods for the PT_OPT_BASE:OptBase application class. The methods are listed in alphabetical order.

GetParmDate

Syntax

GetParmDate(*parmName*, *&parmVal*)

Description

The GetParmDate method retrieves a Date parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

GetParmDateArray

Syntax

GetParmDateArray(*parmName, &parmVal*)

Description

The GetParmDateArray method retrieves a Date array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateTime

Syntax

GetParmDateTime(*parmName, &parmVal*)

Description

The GetParmDateTime method retrieves a DateTime parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmDateTimeArray

Syntax

GetParmDateTimeArray(*parmName, &parmVal*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The GetParmDateTimeArray method retrieves a DateTime array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmNumber

Syntax

```
GetParmNumber(parmName, &parmVal)
```

Description

The GetParmNumber method retrieves a Number parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmNumberArray

Syntax

```
GetParmNumberArray(parmName, &parmVal)
```

Description

The GetParmNumberArray method retrieves a Number array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmInt

Syntax

GetParmInt(*parmName, &parmVal*)

Description

The GetParmInt method retrieves an Integer parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify an Integer variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmIntArray

Syntax

GetParmIntArray(*parmName, &parmVal*)

Description

The GetParmIntArray method retrieves a Number array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmString

Syntax

GetParmString(*parmName, &parmVal*)

Description

The GetParmString method retrieves a String parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmStringArray

Syntax

GetParmStringArray(*parmName, &parmVal*)

Description

The GetParmStringArray method retrieves a String array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmTime

Syntax

GetParmTime(*parmName, &parmVal*)

Description

The GetParmTime method retrieves a Time parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

GetParmTimeArray

Syntax

GetParmTimeArray(*parmName, &parmVal*)

Description

The GetParmTimeArray method retrieves a Time array parameter value that's passed as input by any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time array variable to contain the value passed as input by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

Init

Syntax

```
Init()
```

Description

The Init method launches when the CreateOptEngine built-in function loads an analytic instance that uses the Optimization PeopleCode plug-in.

Use this method to perform additional processing for your own purposes, including checking table data, or any functionality you want to apply before any plug-in transactions run. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run before any other code in your extended class.

Note. If you don't extend this method, PeopleSoft Optimization Framework calls its base version from the OptBase application class.

Parameters

None.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptDeleteCallback

Syntax

```
OptDeleteCallback(&Record)
```

Description

The OptDeleteCallback method launches when PeopleSoft Optimization Framework propagates to the datacache any database deletions that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the deletion. You accomplish this by adding your own PeopleCode to the extended method.

Copyright © 1988-2007, Oracle. All rights reserved.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&Record	Specifies a record variable that contains the keys of the data row to be deleted.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptInsertCallback

Syntax

```
OptInsertCallback(&Record)
```

Description

The OptInsertCallback method launches when PeopleSoft Optimization Framework propagates to the datacache any database insertion that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the insertion. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&Record	Specifies a record variable that contains the new data row to be inserted.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptPostUpdateCallback

Syntax

```
OptPostUpdateCallback(&OldRecord, &NewRecord)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The OptPostUpdateCallback method launches after PeopleSoft Optimization Framework propagates to the datacache any database update that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might have been affected by the update. You accomplish this by adding your own PeopleCode to the extended method. The parameters provide the previous and current content of the row.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

Parameter	Description
&OldRecord	Specifies a record variable that contains the pre-update content of the data row that was updated.
&NewRecord	Specifies a record variable that contains the post-update content of the data row that was updated.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptPreUpdateCallback

Syntax

```
OptPreUpdateCallback(&OldRecord, &NewRecord)
```

Description

The OptPreUpdateCallback method launches before PeopleSoft Optimization Framework propagates to the datacache any database update that it encounters for a callback record.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the update. You accomplish this by adding your own PeopleCode to the extended method. The parameters provide the current and future content of the row.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&OldRecord	Specifies a record variable that contains the pre-update content of the data row to be updated.
&NewRecord	Specifies a record variable that contains the post-update content of the data row to be updated.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptRefreshCallback

Syntax

```
OptRefreshCallback()
```

Description

The OptRefreshCallback method launches after PeopleSoft Optimization Framework propagates to the datacache all database insertions, deletions, and updates that it encounters for all callback records.

Use this method to perform additional processing for your own purposes, including modifying any derived data structures that might be affected by the modifications. You accomplish this by adding your own PeopleCode to the extended method.

Don't call this method in your own PeopleCode. It's called automatically at the appropriate moment by PeopleSoft Optimization Framework, which enables your added functionality to run.

Important! If you designate any record in the analytic type definition as a callback record, you must ensure that you extend this callback method in your derived class, even if the extended method contains only a Return statement. Otherwise the Optimization PeopleCode plug-in will fail.

Parameters

None.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDate

Syntax

```
SetOutputParmDate(parmName, &parmVal)
```

Description

Use the SetOutputParmDate method to pass a Date parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateArray

Syntax

```
SetOutputParmDateArray(parmName, &parmVal)
```

Description

Use the SetOutputParmDateArray method to pass a Date array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Date array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateTime

Syntax

```
SetOutputParmDateTime(parmName, &parmVal)
```

Description

Use the SetOutputParmDateTime method to pass a DateTime parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmDateTimeArray

Syntax

SetOutputParmDateTimeArray(*parmName, &parmVal*)

Description

Use the SetOutputParmDateTimeArray method to pass a DateTime array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a DateTime array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmNumber

Syntax

SetOutputParmNumber(*parmName, &parmVal*)

Description

Use the SetOutputParmNumber method to pass a Number parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmNumberArray

Syntax

```
SetOutputParmNumberArray(parmName, &parmVal)
```

Description

Use the SetOutputParmNumberArray method to pass a Number array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmInt

Syntax

```
SetOutputParmInt(parmName, &parmVal)
```

Description

Use the SetOutputParmInt method to pass an Integer parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify an Integer variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmIntArray

Syntax

```
SetOutputParmIntArray(parmName, &parmVal)
```

Description

Use the SetOutputParmIntArray method to pass a Number array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Number array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmString

Syntax

```
SetOutputParmString(parmName, &parmVal)
```

Description

Use the SetOutputParmString method to pass a String parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmStringArray

Syntax

```
SetOutputParmStringArray(parmName, &parmVal)
```

Description

Use the SetOutputParmStringArray method to pass a String array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a String array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmTime

Syntax

```
SetOutputParmTime(parmName, &parmVal)
```

Description

Use the SetOutputParmTime method to pass a Time parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

SetOutputParmTimeArray

Syntax

SetOutputParmTimeArray(*parmName, &parmVal*)

Description

Use the SetOutputParmTimeArray method to pass a Time array parameter value as output from any method you develop that corresponds to an Optimization PeopleCode plug-in transaction. You develop the transaction method in an application class that you derive from the OptBase application class.

Parameters

Parameter	Description
parmName	Specify the name of the parameter as it's defined for the Optimization PeopleCode plug-in transaction.
&parmVal	Specify a Time array variable that contains a value to be passed as output by the parameter.

Returns

A Boolean value: True if the method is successful, False otherwise.

OptInterface Class Methods

This section discusses the optimization methods for the OptInterface PeopleCode class. The methods are listed in alphabetical order.

Note. You can use the OptInterface class methods only within an application class that you extend from the OptBase application class, or within PeopleCode that you call from that application class. This ensures that the OptInterface PeopleCode runs only on the optimization engine.

Copyright © 1988-2007, Oracle. All rights reserved.

ActivateModel

Syntax

ActivateModel(*ModelID*, *SolverSettingID*)

Description

The ActivateModel method designates the specified model and solver setting as active. The model and the solver are initialized and populated with data from the current analytic instance.

Note. This method fails if the specified model (and by extension, one of its solver settings) is already active. If you want to activate a different solver setting for the same model, you must first deactivate the model.

See Chapter 25, "Optimization PeopleCode," DeactivateModel, page 980.

Parameters

Parameter	Description
ModelID	Specify the name of the optimization model you want to activate. This must be the name of one of the models associated with the analytic type definition.
SolverSettingID	Specify the name of the solver setting you want to activate. This is the name you specified for the solver setting in the analytic type definition.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

Example

```
Local integer &result;
Local OptInterface &oi = CreateOptInterface();

&result = &oi.ActivateModel("QE_PSA_MODEL", "abc");
```

ActivateObjective

Syntax

ActivateObjective(*Model_Name*, *Objective_Name*)

Description

Use the ActivateObjective method to activate the specified objective for an optimization model.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Model_Name	Specify the name of the model.
Objective_Name	Specify the name of the objective.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

DeactivateModel

Syntax

DeactivateModel(*ModelID*)

Description

The DeactivateModel method detaches the solver from the specified model.

Parameters

Parameter	Description
ModelID	Specify the name of the optimization model you want to deactivate. This must be the name of one of the models associated with the analytic type definition.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

Example

```
Local integer &result;
Local OptInterface &oi = CreateOptInterface();

&result = &oi.DeactivateModel("QE_PSA_MODEL");
```


Copyright © 1988-2007, Oracle. All rights reserved.

DumpMsgToLog

Syntax

DumpMsgToLog(*LogSeverity*, *Message*)

Description

The DumpMsgToLog method writes the specified status message to the optimization engine log file, with the specified severity.

Parameters

Parameter	Description
LogSeverity	Specify the severity level of the message, as one of the following system constants: • %Severity_Fatal • %Severity_Status • %Severity_Error • %Severity_Warn • %Severity_Info • %Severity_Trace1 • %Severity_Trace2
Message	Specify as a string the text of the log message.

Returns

None.

FindRowNum

Syntax

FindRowNum(*&Record* [, *startrow* [, *endrow* [, *field_list*]]])

Where *field_list* is a list of field names in the form:

[*fieldname1* [, *fieldname2*]]...

Description

The FindRowNum method determines the row number of a row in the datacache rowset. You provide a record with key values, and this method finds the row with the same key values and returns its row number.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&Record	Specify a record with the same structure as the records that comprise the rowset, with its key fields populated.
startrow	Specify as an integer the starting row number of the search. Specify 0 to search from the first row in the rowset.
endrow	Specify as an integer the ending row number of the search. Specify 0 to search through the last row in the rowset.
fieldname	Specify the name of a field in the input record which contains a value to be matched. You can specify one or more field names, in any order. **Note.** If you use this parameter, the fields specified here are used to search, instead of the record's key fields. Any value that doesn't correspond to a field name is ignored.

Returns

The row number of the row containing the specified key values, or 0 if no row is found.

Example

The following example searches the whole scroll to find the partial key OPT_SITE:

```
Local Record &rec = CreateRecord(Scroll.OPT_TRANSCOST);
Local Optineterface &oi;

&rec.OPT_SITE.value = "New York";
int nRowNum = &oi.FindRowNum(&rec, 0, 0, "OPT_SITE");
```

The following example searches from row 5 to row 15 with the full key values New York and San Jose:

```
Local Record &rec = CreateRecord(Scroll.OPT_TRANSCOST);
Local Optineterface &oi;

&rec.OPT_SITE.value = "New York";
&rec.OPT_STORE.value = "San Jose";
int nRowNum = &oi.FindRowNum(&rec, 5, 15);
```

GetSolution

Syntax

GetSolution(*ModelID*, *varArrayID*, *skipZero* [, *KeyFieldNames*, *KeyFieldValues* [, *&Solution*]])

Description

The GetSolution method retrieves the model solution values generated by the Solve method.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model for which you want the solution. This is the name used for the model definition in Application Designer.
varArrayID	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
skipZero	Indicate whether solutions with a value of zero should be fetched. This parameter takes a Boolean value: • True: Don't fetch solutions with a zero value. This can increase the performance of the GetSolution method if zero values aren't meaningful. • False: Do fetch solutions with a zero value.
KeyFieldNames and *KeyFieldValues*	Specify a set of key field names as an array of string and a set of key field values as an equal length array of ANY, with one key field value corresponding to each key field name. You use these arrays to restrict the set of returned solutions. Solutions are returned only for model variables with the specified key field values. **Note.** If you provide either of these arrays, you must provide both. You can include each parameter from the variable array at most only once.
&Solution	Specify a rowset to contain the solutions.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if the solver fails to solve the problem.

Example

```
Local array of string &strArray;
Local array of any &valArray;
Local integer &index;
Local Rowset &rowSet;
Local integer &result;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local boolean &bSkipZero = True;

Local OptInterface &oi = CreateOptInterface();

&strArray = CreateArrayRept("", 0);
&valArray = CreateArrayAny();
&rowSet = CreateRowset(Record.QEOPT_VAL_X_WRK);

&strArray [1] = "EMPLID";
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&valArray [1] = 1;
&strArray [2] = "ORDER_ID";
&valArray [2] = 23;

/* fetch only the part of the solution where  EMPLID = 1 and ORDER_ID = 23 */
&result = &oi.GetSolution(&modelId, &varArrayName,
 &bSkipZero, &strArray, &valArray, &rowSet);
```

GetSolutionDetail

Syntax

GetSolutionDetail(*ModelID*, *SolutionType*, *Name*, *&Solution*)

Description

The GetSolutionDetail method retrieves the model solution detail of the specified type generated by the Solve method. You can retrieve dual value, slack value, or reduced cost information.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model for which you want the solution detail. This is the name used for the model definition in Application Designer.
SolutionType	Specify a system constant indicating the type of solution detail you want to retrieve. The value you specify here determines the content of the *Name* and *&Solution* parameters. • %OPT_DUAL: Retrieve the dual value attributes of the specified constraint block. • %OPT_SLACK: Retrieve the slack value attributes of the specified constraint block. • %OPT_RCOST: Retrieve the reduced cost attributes of the specified variable array.
Name	If you specified a *SolutionType* of %OPT_DUAL or %OPT_SLACK, specify here the name of a constraint block from the active model. If you specified a *SolutionType* of %OPT_RCOST, specify here the name of a variable array from the active model.
&Solution	Specify a rowset to contain the solution details. The rowset should have the same key fields as the constraint block or the variable array you specified with the *Name* parameter.

Returns

This method returns a constant value. Valid values are:

Value	Description
%OptInter_Success	Returned if method succeeds.
%OptInter_Fail	Returned if solver fails to solve the problem.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local Rowset &dual_rowset;
Local integer &result;
Local OptInterface &oi = CreateOptInterface();
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local string &constrName = "Constraint_1";

/* fetch dual values for Contraint "Constraint_1"
 in a rowset based on the QEOPT_C1_WRK record */

&dual_rowset = CreateRowset(Record.QEOPT_C1_WRK);
&result = &oi.GetSolutionDetail(&modelId, %Opt_Dual, &constrName, &dual_rowset);
```

IsModelActive

Syntax

IsModelActive(*ModelID*)

Description

Use the IsModelActive method to determine if the model specified by *ModelId* is active before it is used.

Parameters

Parameter	Description
ModelID	Specify the model ID as a string. This is the name used for the model definition in PeopleSoft Application Designer.

Returns

A boolean value: true if the model is active, false otherwise.

RestoreBounds

Syntax

RestoreBounds(*modelID* [,*varArrayID*])

Description

The RestoreBounds method returns the bounding values of the specified variable array or arrays to the current settings in the specified model.

If you previously called the SetVariableBounds method with the *changeModelBounds* parameter set to true for any variable or variable array, those bounding values still apply.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
modelID	Specify as a string the name of the optimization model for which you want to restore the bounding values. This is the name used for the model definition in PeopleSoft Application Designer.
varArrayID	Specify as a string the name of a variable array for which you want to restore the bounding values. Your application documentation should provide this name. If you don't specify a variable array name, the bounding values are restored for all variable arrays in the specified model.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

SetVariableBounds

Syntax

```
SetVariableBounds(modelID, varArrayID, boundType, lowerBound, upperBound,
&keyRecord [, changeModelBounds])
```

Description

The SetVariableBounds method overrides the bounding values specified for a model variable array, or for a variable within the array.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
modelID	Specify as a string the name of the optimization model for which you want to override the bounding values. This is the name used for the model definition in Application Designer.
varArrayID	Specify as a string the name of the variable array being optimized. Your application documentation should provide this name.
boundType	Specify a system constant indicating which bounding values to override. The value you specify here determines how the *lowerBound* and *upperBound* parameters are applied to the specified model. • %OPT_LOWER_BOUND: Override only the lower bound as specified by the *lowerBound* parameter. The *upperBound* parameter is ignored. • %OPT_UPPER_BOUND: Override only the upper bound as specified by the *upperBound* parameter. The *lowerBound* parameter is ignored. • %OPT_BOUND_BOTH: Override both the lower bound and the upper bound as specified by the *lowerBound* and *upperBound* parameters, respectively.
lowerBound	Specify as a number the lower bound that should be applied to a variable or a variable array if the *boundType* parameter permits the override. You can also set this parameter to one of the following system constants:
upperBound	Specify as a number the upper bound that should be applied to a variable or a variable array if the *boundType* parameter permits the override. You can also set this parameter to one of the following system constants:
&keyRecord	Specify a record with the same key fields as the variable array being optimized. To override the bounding values specified for a single variable within the array, populate the record's key fields to specify the variable. To override the bounding values specified for the entire variable array, set all of the record's fields to a null value. **Note.** You must either provide values for all keys, or set them all to null values.
changeModelBounds	Specify a Boolean value: • true: Indicates that the specified model should be updated in memory to reflect the specified variable bounds. Any analytic instance that invokes this model from the active optimization engine is affected by these settings, which are propagated to the solver in memory. This is the default value if you omit this parameter. • false: Indicates that the specified model should not be updated in memory, and that the specified variable bounds apply only to the next time the Solve method is called.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

Example

```
Local Record &rec;
Local integer &result;
Local OptInterface &oi = CreateOptInterface();
Local float &objval = 0.0;
Local string &modelId = "QE_PSA_MODEL";
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local string &varArrayName = "X";
Local float &lb = 0.0;
Local float &ub = 0.0;

&rec = CreateRecord(Record.QEOPT_VAL_X_WRK);
&rec.QEOPT_RESINDEX.Value = 1;
&rec.QEOPT_SOLINDEX.Value = 2;
&rec.QEOPT_TIMEINDEX.Value = 3;

&result = &oi.SetVariableBounds(&modelId, &varArrayName,
 %Opt_Upper_Bound, &lb, &ub, &rec, False);
```

SetVariableType

Syntax

SetVariableType(*modelID*, *varArrayID*, *varType*)

Description

Use the SetVariableType method to change the data type of a model variable array.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model for which you want to change the variable type. This must be the name of one of the models associated with the analytic type definition.
varArrayID	Specify as a string the name of the variable array for which you want to change the variable type. Your application documentation should provide this name.
varType	Specify one of the following system constants representing the new variable type: • %Opt_Var_Cont: Represents a continuous variable type, which can be any floating point value. • %Opt_Var_Bin: Represents a binary variable type, for which the value can be only 0 or 1. • %Opt_Var_Int: Represents an integer variable type, which can be any integer.

Returns

%OptInter_Success if the method succeeds, %OptInter_Fail otherwise.

Example

```
Local OptInterface &oi = CreateOptInterface();
Local string &varArrayName = "X";
Local integer &result;

&result = &oi.SetVariableType("QE_PSA_MODEL",  &varArrayName, %Opt_Var_Bin);
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
If (&result <> %OptInter_Success) Then
   &oi.DumpMsgToLog(%Severity_Status, "Failed to change variable type ");
End-If;
```

Solve

Syntax

Solve(*modelID*, *SolutionType* [, *&objValue* [, *name-value_pairs*]])

Where *name-value_pairs* is a list of solver setting parameter values in the form:
[*parmname1*, *parmvalue1* [, *parmname2*, *parmvalue2*]]...

Description

The Solve method solves the specified model using the currently active solver settings, and provides an objective value as the solution output. You can override the solver setting parameters. The returned solution status is a predefined system constant.

Parameters

Parameter	Description
ModelID	Specify as a string the name of the optimization model you want to solve. This is the name used for the model definition in Application Designer.
SolutionType	Specify a system constant indicating the type of solution detail you want the model to be solved for. • %OPT_DUAL: Generate dual value attributes. • %OPT_SLACK: Generate slack value attributes. • %OPT_RCOST: Generate reduced cost attributes. You can also combine any or all of these system constants, by connecting them with a plus sign (+), for example: %OPT_DUAL + %OPT_RCOST.
&objValue	Specify a reference to a variable of type float. This variable contains the output objective value produced by the solver upon successfully solving the specified optimization model.
parmname and *parmvalue*	Specify a solver setting parameter ID and value to override the original value you specified for the solver setting in the analytic type definition. You can override any or all of the solver setting parameter values. See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Optimization Framework*, "Designing Analytic Type Definitions," Configuring Models for Optimization.

Returns

One of the following system constants:

%OptInter_Fail: The solver fails to solve the problem.

%Opt_Optimal: The solution is optimal.

%Opt_Infeasible: The solution is infeasible.

Copyright © 1988-2007, Oracle. All rights reserved.

%Opt_Unbounded: The solution is unbounded.

%Opt_Timeup: The solver reached the time limit specified in the solver setting.

%Opt_Iterlimit: The solver reached the limit on the number of iterations specified in the solver setting.

%Opt_LP_Max_Sols: The solver generated maximum number of solutions without improvement.

%Opt_Idle: The solution shows no improvement in a specified time limit.

%Opt_Unknown: The solver status is unknown.

%Opt_MIP_NumSolutions: The specified number of solutions corresponding to an MIP solver reached.

%Opt_MIP_NumNodes: The specified number of nodes corresponding to an MIP solver reached.

%Opt_Aborted: The solver aborted.

%Opt_User_Exit: A user exit was encountered.

%Opt_First_LP_NoOpt: While solving an MIP, the first LP solution obtained was not optimal.

Example

Following is an example of the basic use of the Solve method:

```
Local OptInterface &oi = CreateOptInterface();

Local float &objval = 0.0;
Local integer &result;
Local string &modelId = "QE_PSA_MODEL";
Local string &varArrayName = "X";
Local integer &solType;

&solType = %Opt_RCost + %Opt_Dual + %Opt_Slack;

/* Solve the problem */
&result = &oi.Solve("QE_PSA_MODEL", &solType, &objval);

If & result = %Opt_Optimal Then
   &oi.DumpMsgToLog(%Severity_Warn, " Solution Status =  " Optimal !!!");
Else
   &oi.DumpMsgToLog(%Severity_Warn, " Solution Status =  " | &result );
End-If;
```

Following is an example of a solver setting parameter override:

```
Local OptInterface &oi = CreateOptInterface();
Local float &objval = 0.0;
Local integer &result;

/* This overrides the solver setting for MPS_Filename and generates
 an MPS file called myfile.mps instead of the name specified
 in the current solver setting parameter. */

&result = &oi.Solve("QE_PSA_MODEL", %Opt_Primal, &objval, "MPS_FileName",⇒
 "myfile");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 26

Page Class

This chapter provides an overview of Page class and discusses the following topics:

- Shortcut considerations.
- Page object declaration.
- Scope of a Page object.
- Page class built-in function.
- Page class reference.

Understanding Page Class

Use the page class to manipulate a page in a component. The most common use of this class is to hide or display a page in a component, either based on field values or the level of security of the user.

Generally, the PeopleCode used to manipulate a page object is associated with PeopleCode in the Activate event.

Note. The page object should not be used until after the page processor has loaded the page: do not instantiate this object in RowInit PeopleCode, use it in PreBuild, PostBuild or Activate instead.

In addition, if you need to hide the current page, PeopleSoft recommends using the PreBuild event for your program.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Component Build Processing in Update Modes

Shortcut Considerations

An expression of the form

```
PAGE.pagename.property
```

is equivalent to `GetPage(PAGE.name).property`. For example, the following two lines of code are equivalent:

```
PAGE.MANAGER_APPROV.DisplayOnly = False;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
GetPage(PAGE.MANAGER_APPROV).DisplayOnly = False;
```

Data Type of a Page Object

Use the Page data type to declare Page objects. For example,

```
Local Page &MYPAGE;
```

Scope of a Page Object

A page object can be instantiated from PeopleCode only.

Use this object only in PeopleCode programs that are associated with an online process, not in an Application Engine program, a message subscription, a Component Interface, and so on.

In addition, the page object should not be used until after the page processor has loaded the page: do not instantiate this object in RowInit PeopleCode. Use it in PostBuild or Activate instead.

Page Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetPage

Page Class Properties

In this section, we discuss each Page class properties.

CopyURLLink

Description

Use this property to enable the http link for a component. Clicking this link copies the address (URL) of the page to the clipboard. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in PeopleSoft Application Designer, you cannot set any of the Pagebar properties using PeopleCode. If you deselect the Copy URL Link checkbox for the component in PeopleSoft Application Designer, you can still set this option using this property.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Page Definitions," Setting Use Properties

CustomizePageLink

Description

Use this property to enable Customize Page link for a component. Clicking this link enables the user to control the initial display of the component. This property takes a Boolean value, true, the user can control the display, false otherwise. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in the PeopleSoft Application Designer, you cannot set any of the Pagebar properties using PeopleCode. If you deselect the Customize Page Link checkbox for the component in PeopleSoft Application Designer, you can still set this option using this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Page Definitions," Setting Use Properties

DisplayOnly

Description

Use this property to make a page display-only, or to enable a page so the fields can be edited. This property takes a Boolean value: True if the page is display-only, False otherwise.

Note. You cannot enable or disable the current page.

The value of DisplayOnly for a page is restored to the value set in PeopleSoft Application Designer when the user goes to another component. This means you must set DisplayOnly for the page each time a component is accessed.

This property is read-write.

Example

```
If &MYPAGE.DisplayOnly Then
   &MYPAGE.Visible = True;
Else
   &MYPAGE.Visible = False;
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

HelpLink

Description

Use this property to enable the Help link for a component. Clicking this link accesses the online help PeopleBook entry for the current page. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in the PeopleSoft Application Designer, you cannot set and of the Pagebar properties using PeopleCode. If you deselect the Help Link checkbox for the component in PeopleSoft Application Designer, you can still set this option using this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Page Definitions," Setting Use Properties

Name

Description

This property returns a reference to the page name definition (a string value.)

This property is read-only.

Example

```
&MYPAGENAME = &MYPAGE.Name;
```

NewWindowLink

Description

Use this property to enable New Window link for a component. Clicking this link opens a new browser window with the search page for the current component. Users can view or enter data in the new window. This is a Pagebar property.

Note. If you select the Disable Pagebar checkbox for the component in the PeopleSoft Application Designer, you cannot set any of the Pagebar properties using PeopleCode. If you deselect the New Window Link checkbox for the component in PeopleSoft Application Designer, you can still set this option using this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Page Definitions," Setting Use Properties

 Copyright © 1988-2007, Oracle. All rights reserved.

Visible

Description

If this property is True, the page executing this property is displayed. If you set this property to False, the page is hidden. When the visibility is changed, the component tabs and menus are also automatically changed to reflect the new settings. Menus return to their original values when the user goes to another component.

If you set a page to be hidden in the component definition, you can change the value of the page to be visible at runtime.

PeopleSoft recommends using the PreBuild event if you need to hide the current page.

Example

In the following example, a page is hidden based on the value of the current field.

```
If PAYROLE_TYPE = "Global" Then
   PAGE.JOB_EARNINGS.Visible = False;
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Component Definitions"

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 27

PortalRegistry Classes

This chapter provides an overview of PortalRegistry classes and discusses how to:

- Use the PortalRegistry API
- Use security
- Work with ValidFrom and ValidTo
- Do error handling
- Understand the life-cycle of a PortalRegistry object
- View the PortalRegistry class hierarchy
- Use content references
- Understand naming conventions
- Delete content considerations
- Save content considerations
- Declare a PortalRegistry object
- Understand the scope of a PortalRegistry object
- Use PortalRegistry (reference section)
- Understand PortalRegistry classes examples

See Also

Enterprise PeopleTools 8.49 PeopleBook: Internet Technology, "Understanding Portal Technology"

Understanding Portal Registry

The portal registry is a tree structure in which content for a portal is organized, classified, and registered. The PortalRegistry classes (API) are used to update the portal registry. The Portal Administration pages provide a GUI access to the PortalRegistry API. You can also access them using a PeopleCode program you write yourself. How you access the portal registry depends on the type of updates required. Your organization will likely use both methods of updating the portal registry. This chapter focuses on accessing the PortalRegistry classes using PeopleCode.

A portal is a website that helps you navigate to other web-based applications and content. The PeopleSoft Portal is a *business portal*. It is similar to general purpose portals, except that its main purpose is to help end-users be more effective in accessing information to perform their jobs.

Copyright © 1988-2007, Oracle. All rights reserved.

Each PeopleSoft Portal is defined by one PeopleSoft portal registry. The PeopleSoft portal registry consists of a number of system tables and associated data in a PeopleSoft database. The portal registry must reside within one PeopleSoft database. There can be more than one portal registry in a PeopleSoft database, but only one portal registry is associated with a PeopleSoft Portal.

The portal registry consists of the following primary parts:

- Folders
- Content references
- Nodes

Folders and content references make up the majority of the portal registry, and provide a hierarchical tree structure to describe various content that is *registered* as part of a PeopleSoft portal.

Nodes provide a logical name to a specific webserver and database, so content can be registered independent of specific webservers. It is used when the portal servlet attempts to retrieve content—both internal PeopleSoft content as well as external content—and to assemble pages.

The primary function of the portal is to take a target URL (generally a URL for a PeopleSoft component) that comes in from a user's browser, and assemble a page with that content and any other content. The layout and what content to assemble on the page is defined by the *node template*, which is composed of HTML. The portal attempts to find the content reference associated (that is, registered) with the target URL to get the template, or uses a series of default templates if it cannot get the template from a content reference.

Folders

Folders are how a hierarchical structure is created within a portal registry. Each folder has a parent, except the root folder, which is the top-level folder in a portal registry. Each folder can also contain child folders and content references. Folders are roughly analogous to directories within a file system. A folder can be further defined by a number of attributes (description, security, when it expires, and so on) that are useful within the portal environment.

See Also

Chapter 27, "PortalRegistry Classes," Adding a Folder, page 1172

Content References

A content reference is simply a reference to a URL. After you create an entry for a content reference in the portal registry, it's considered registered. A content reference can be further defined by a number of attributes (description, security, when it expires, and so on) that are useful within the portal environment.

There are a number of distinct types of content references that can be broken down into the following broad categories:

- Target
- Template
- Component

A *target* type of content reference describes a registered URL that a user might reference. Typically, these would be a PeopleSoft Pure Internet Architecture component, such as a page in a transaction. When a user references a URL, the URL is looked up in the registry to find the target content reference.

A *template* type of content reference describes what, if any, other content to put on the page, and where to place that content. The portal attempts to find a template for every URL that is requested.

 Copyright © 1988-2007, Oracle. All rights reserved.

A *component* type of content reference describes a component that is typically placed on a target page or homepage.

See Also

Chapter 27, "PortalRegistry Classes," Content Reference Links, page 1101

Nodes

When a content reference is created to register some content (that is, a URL) the specific URI (that is, the scheme, webserver, and so on) can be specified in the content reference. However, this has the drawback that every time the URI for a content reference changes (the webserver name changes, and so on), the content reference must be changed. Nodes are a way to create a logical name for a specific webserver, scheme, and so on. When the content reference is created you can specify a node name.

For example, suppose the name of a webserver changed. If you don't use nodes, you must check all your content references and change them accordingly. If you use nodes, you have to change only the webserver name in one place, and all the content references that use that node now automatically reference the correct webserver.

Nodes can be optional for content where the specified URL is already a complete URL, such as for external content. Nodes can be categorized as:

- default local
- local
- non-local (remote)

See Also

Chapter 27, "PortalRegistry Classes," Using Attributes, page 1179

Security

The same security mechanism is used for folders, content references, content reference links, tab definitions, pagelets, and user homepages. All of these items can be marked as public, owner accessible, or can have explicit PeopleSoft permission values set, including cascading the permissions to its child objects (that is, the child objects have the same permissions as the parent objects, when applicable.)

When marked as public, the item is accessible by any user. Public access cannot be cascaded, that is, passed down, to child objects.

When marked as owner accessible, an item is accessible only by the same user that created that item. Owner accessible cannot be cascaded, that is, passed down, to child objects.

Items can be marked as accessible by PeopleSoft permissions. This means that if a user is a member of a role, and the role contains the permission list that the item is also associated with, the user has access to that item. Additionally, when applied to folder permissions the permission can be cascaded. This means that any child of that folder, including a content reference, automatically has that permission added to its permission list.

Role-based security can applied to the portal objects (folders, content references and content reference links) using the RolePermissions collection.

Note. You can only use role-based security for content references, folders, and content reference links that are not components or iScripts.

Copyright © 1988-2007, Oracle. All rights reserved.

You can specify which roles are allowed to access the objects. This works similarly to permission lists. If a user has a specified role, authorization is granted.

A role collection is the same as the PermissionValue collection, though there are additional properties on PermissionValue.

Attributes

Folders, content references, content reference links, PageletCategory objects, and Pagelets can have any number of attributes added to them. Attributes are simply name/value pairs. These name/value pairs are defined and used by many portal-aware applications, such as the search engine, navigational display, and so on.

See Chapter 27, "PortalRegistry Classes," Using Attributes, page 1179.

Additional Portal Objects

Using the PortalRegistry, you can also add and customize other items in your portal, such as tabs, homepages, and favorites.

The TabDefinition is a homepage tab that has been defined to the portal. It is what an administrator creates when they want to define a new homepage tab. The TabDefinition is based on content references.

The User Homepage is a personalized homepage for a user. It contains all the tabs and pagelets the user has selected for their homepage. The User Homepage is based on folders.

A Favorite represents a content reference that the user wants to keep a shortcut to. It contains the name of the content reference, and the label the user wants displayed for this favorite.

A Pagelets is an area on a page that contains content, and the template used to specify the style for that content. Pagelets are a type of content reference.

Using the PortalRegistry API

The PortalRegistry API provides the entry point to a specific portal registry. Common administrative tasks include adding, deleting, and renaming the registry objects (that is, folders, content references, tabs, and so on.) Additionally, there are many properties associated with every registry object, and all of these properties can be accessed and modified.

You can use the PortalRegistry API in batch mode to make many changes to a portal at once. For example, suppose something changed in your security system. You could write your own PeopleCode program, using the PortalRegistry classes, in an Application Engine program and change the access for all your users at once.

You can also use the PortalRegistry API to exchange data with an external system. For example, suppose an external merchant had an area on your company's portal, and the information there must be updated. One way to do this is for the merchant to use an Application Message to send the data to your system, then a subscription program would update the portal using the PortalRegistry API.

Each PortalRegistry object represents one specific PortalRegistry in a system. An empty PortalRegistry object is initially gotten from the PeopleSoft Session object. You can then open an existing PortalRegistry to view or change existing content, create a new PortalRegistry, or delete an existing PortalRegistry.

 Copyright © 1988-2007, Oracle. All rights reserved.

Using Security

The PermissionValue object associated with a folder, content reference, and so on, as well as specific properties on the object, work together to form the security for an object. In this section, we discuss how to:

- Use object properties.
- Access objects.

Using Object Properties

The properties that set permissions for an object are:

- AuthorAccess
- PublicAccess

The AuthorAccess property determines whether the author of the object has access to the object.

The PublicAccess property determines whether the object is generally accessible or not. If this property is set to True, all users have access to the object.

These properties apply only to an object. They *cannot* be cascaded, that is, passed down to child objects.

The other object property used with security, the Authorized property, indicates whether a user is authorized to access an object. The value of this property depends on whether the user has access.

Accessing Objects

When you get a folder, content reference, content reference link, PageletCategory or Pagelet collection, only the items that the end-user is authorized to access are in the collection.

An object is contained in a collection is based on the following algorithm.

- If the object is marked as PublicAccess it is automatically accessible.
- If the object is marked as AuthorAccess and the current UserId is the Author it is accessible.
- If the current user is in a permission list (class) that is in the Permissions collection for that object.
- If the current user is in a permission list (class) that is in the CascadedPermissions collection for that object.
- If the current user has the Portal Administrator role.

When you access a content reference or folder using a valid name and one of the Find methods (FindCRefByURL, FindCRefByName, FindCRefForURL, or FindFolderByName) a content reference or folder is returned whether the user is authorized to it. When you use these methods, always check the Authorized property. This is the only property you can view from an object that an end-user isn't authorized to.

In addition, if you know the specific URL for a tab, you could specify that in the FindCRefByURL. The tab is returned whether the user is authorized to it, so you should always check the Authorized property.

Copyright © 1988-2007, Oracle. All rights reserved.

Using PermissionValue, RolePermissionValue, Cascading Permissions and CascadingRolePermissions

To set non role-based permissions for a folder, content reference, content reference link, PageletCategory object, or Pagelet, you must access the PermissionValue collection for that object using the Permissions property. The PermissionValue objects refer to permission list values that already exist on the system, such as ALLPANLS, CUSTOMER, EMPLOYEE, and so on.

To set role-based permissions for a folder, content reference, or a content reference link that are not component or iScript, you must access the RolePermissionValue collection for that object using the RolePermissions property. The PermissionValue objects refer to role-based permission values that already exist on the system.

Note. The PORTAL_PAGELETS folder is the parent folder for all PageletCategories. Its security is set to public. PeopleSoft does not recommend cascading any permissions from this folder object. Cascade permissions only from the pagelet category (folder).

Both the PermissionValue collection and the RolePermissionValue collection return PermissionValue objects. Use the PermType property to determin if a particular PermissionValue is role-based or not.

For every PermissionValue object, you can chose whether all the child folders and content references of this folder have the same permissions. This is called *cascading*. You cascade permissions by setting the Cascade property to True.

There are two types of PermissionList collections you can access. One is returned by the Permissions and RolePermissions properties, the other by the CascadedPermissions and CascadedRolePermissions properties.

Permissions and RolePermissions

The Permissions and RolePermissions properties return a collection containing the permissions set at the current level, that is, set with that folder or content reference. You can add PermissionValues to this list. Use the Cascade property to cascade the permission you add to child folders and content references.

Note. Folders can contain other folders, but the other objects that use the Permissions property can't contain themselves, that is, content references can't contain other content references, and so on. Therefore, the Cascade property is applicable only to folders, not to any other object.

CascadedPermissions and CascadedRolePermissions

The CascadedPermissions and CascadedRolePermissions properties return the a collection for *all* the permissions for an object. It contains any permission list values set in any parent folder.

Note. You *cannot* add any PermissionValues to a collection returned by the CascadedPermissions or CascadedRolePermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

You can add PermissionValues for a child folder, but you *cannot* delete any of the existing ones that are cascaded. Permissions are augmented, not overwritten, that is, the permissions are the sum of both the parent permissions and whatever is set at the current level.

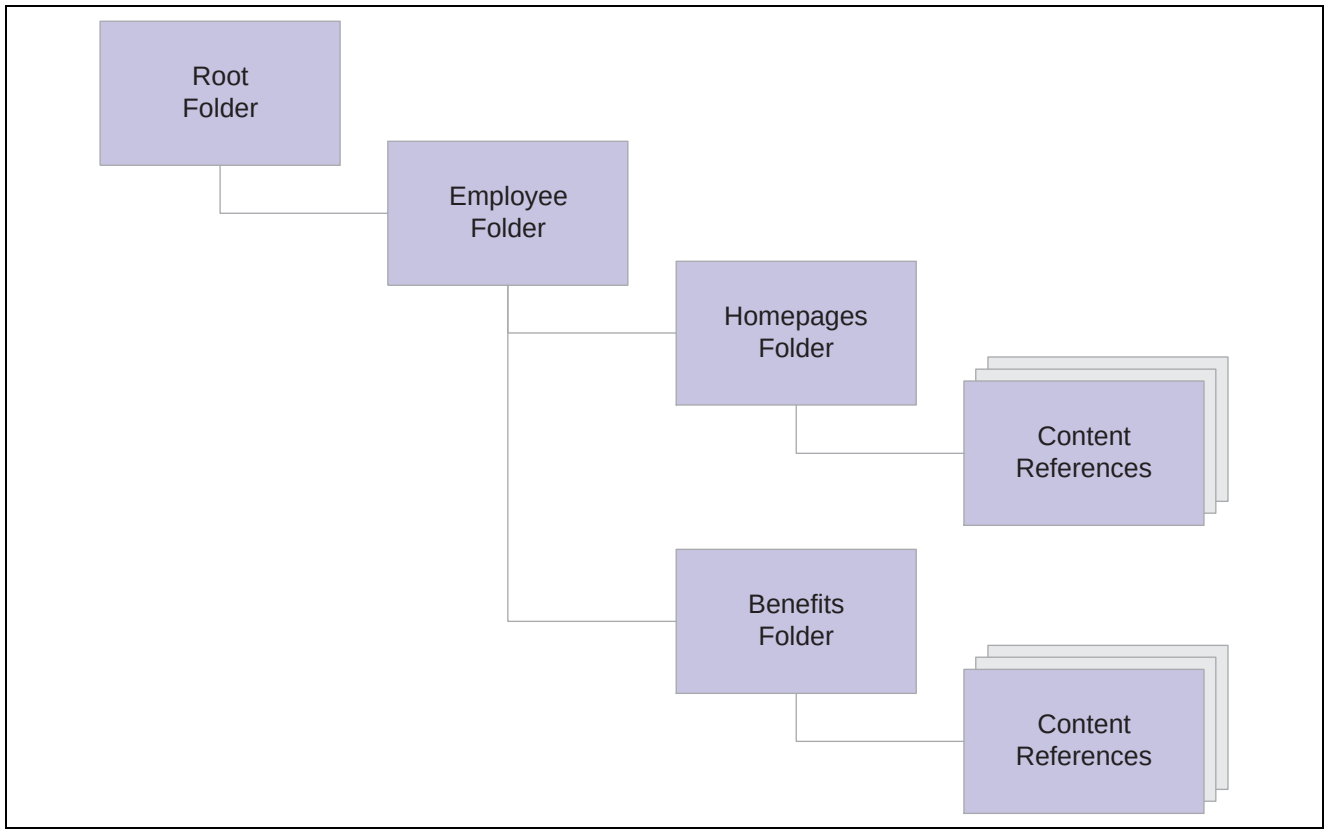
Therefore, keep the following in mind when working with PermissionValues:

- Set permissions only at the level where they're needed.
- Cascade permissions only when necessary.
- Generally, don't set cascaded PermissionValues for the root folder.

 Copyright © 1988-2007, Oracle. All rights reserved.

- Only the values in a PermissionValue object cascade. Properties on the folder itself (such as PublicAccess or AuthorAccess) do *not* cascade.

For example, suppose your PortalRegistry had the following hierarchy:



PortalRegistry hierarchy example

The following permission list values are assigned to the PermissionValue objects for the Employees folder, and both of them are cascaded:

- EMPLOYEE
- MANAGER

The Homepages and Benefits folders have the exact same permissions as the Employees folder, that is, all users associated with these two permission lists have access to these folders.

Suppose you decide that you don't want the permission list EMPLOYEE to access the Benefits folder. If you delete the EMPLOYEE PermissionValue from the Benefits PermissionValue collection, you have *not* altered the permissions. The permissions set at the parent folder, that is, at the Employees folder, can't be deleted, they can only be added to.

To make this change, you must:

- Change the Cascaded property for the EMPLOYEES PermissionValue in the Employees folder to False
- Add EMPLOYEES as a PermissionValue object to the Homepages folder.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 27, “PortalRegistry Classes,” Cascade, page 1093

Chapter 27, “PortalRegistry Classes,” Setting Permissions Using the PermissionValue Object, page 1177

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, “Setting Up Permission Lists”

Working With ValidFrom and ValidTo

Folders, content references, content reference links, and tab definitions have both ValidFrom and ValidTo dates. What’s the difference and how are they used in the PortalRegistry API?

- A ValidFrom date is when something begins.
- A ValidTo date is when something ends.

For example, suppose your HR department has a page describing the benefits for your employees, and that page changed every calendar year. This means the page for the year 2000 has a ValidFrom date of 01/01/2000 and a ValidTo date of 12/31/2000. The benefits page for the year 2001 has a ValidFrom date of 01/01/2001 and a ValidTo date of 12/31/2001.

Folders, content references, content reference links and tab definitions are returned in their collections regardless of the ValidTo and ValidFrom dates. You must take these dates into account and only display to the end-user those values that should be seen.

In addition, a ValidFrom should always come *before* a ValidTo date. If they are set incorrectly, you receive a runtime error.

For all newly created folders, content references, content reference links and tab definitions, the default value for both these properties is Null (""), that is, they begin immediately and do not expire.

In addition, you can also check the IsVisible property to see if a portal object is viewable. IsVisible verifies if an object is Hidden, as well as if the ValidTo and ValidFrom dates are within the specified dates.

Doing Error Handling

All errors for the PortalRegistry classes, like the other APIs, are logged in the PSMessages collection, instantiated from a session object.

The PortalRegistry classes log errors that occur with methods immediately, and errors that occur with properties only after a method is executed.

For example, suppose you specified an invalid name when you were trying to delete a folder. The method (DeleteItem) returns False, and the error is logged in the PSMessages collection immediately.

Now suppose you created a new folder, and specified an invalid ValidTo date. The error won’t be logged in the PSMessages collection until you tried to save your changes.

When you want to check for errors depends on your application. When users are entering data dynamically, and your program is registering their data in the portal, you may want to check for errors often. If you’re using a batch program, you may want to check for errors only after the Open, Save, Insert, and Delete methods.

 Copyright © 1988-2007, Oracle. All rights reserved.

Most methods return a Boolean value indicating success or failure. After the failure of a method, you may want to check the PSMessages collection to determine the exact error.

```
Local ApiObject &MySession;
Local ApiObject &ErrorCol;
Local ApiObject &FolderCol, &Folder, &Registry;
Local Boolean &Open;/* Access the current session */

&MySession = %Session;
If &MySession Then
   /* connection is good */

&Registry = &MySession.GetRegistry();

&Open = &Registry.Open("CUSTOMER");

If &Open Then
   /* Registry opened successfully */
   /* do processing */
Else
   /* Do error checking */

&ErrorCol = &MySession.PSMessages;
For &I = 1 to &ErrorCol.Count
   /* do processing */
End-For;

   End-If;
Else
   /* do processing for no connection */
End-If;
```

See Also

Chapter 36, “Session Class,” Error Handling, page 1512

Understanding the Life-Cycle of a PortalRegistry Object

At runtime, there are certain things you want to do with a PortalRegistry object, like getting an instance of it, adding tabs, editing content references, and so on. The following is an overview of this process. These steps are expanded in other sections.

1. Perform one of these steps.
    a. Execute the GetPortalRegistry method on the PeopleSoft session object to get a PortalRegistry object, or
    b. Use the FindPortalRegistries method on a session object to get a collection of all PortalRegistries. Select a PortalRegistry from the collection.
2. Open the PortalRegistry, using the portal name.
3. After you have an open PortalRegistry object, you can do various actions such as:

Copyright © 1988-2007, Oracle. All rights reserved.

- Add content references to the existing folders.
- Add folders and the content references for them.
- Add or change Nodes.
- Modify the PortalRegistry properties.
- Modify the existing content references, folders, tabs, homepages, and so on.

4. After you make your changes, you must save your work. The Save method must be executed at the appropriate level, such as at the folder level for changes to a folder, at the PortalRegistry level for changes to the default template, and so on.

   A property value change isn't committed to the database until the parent object is saved.

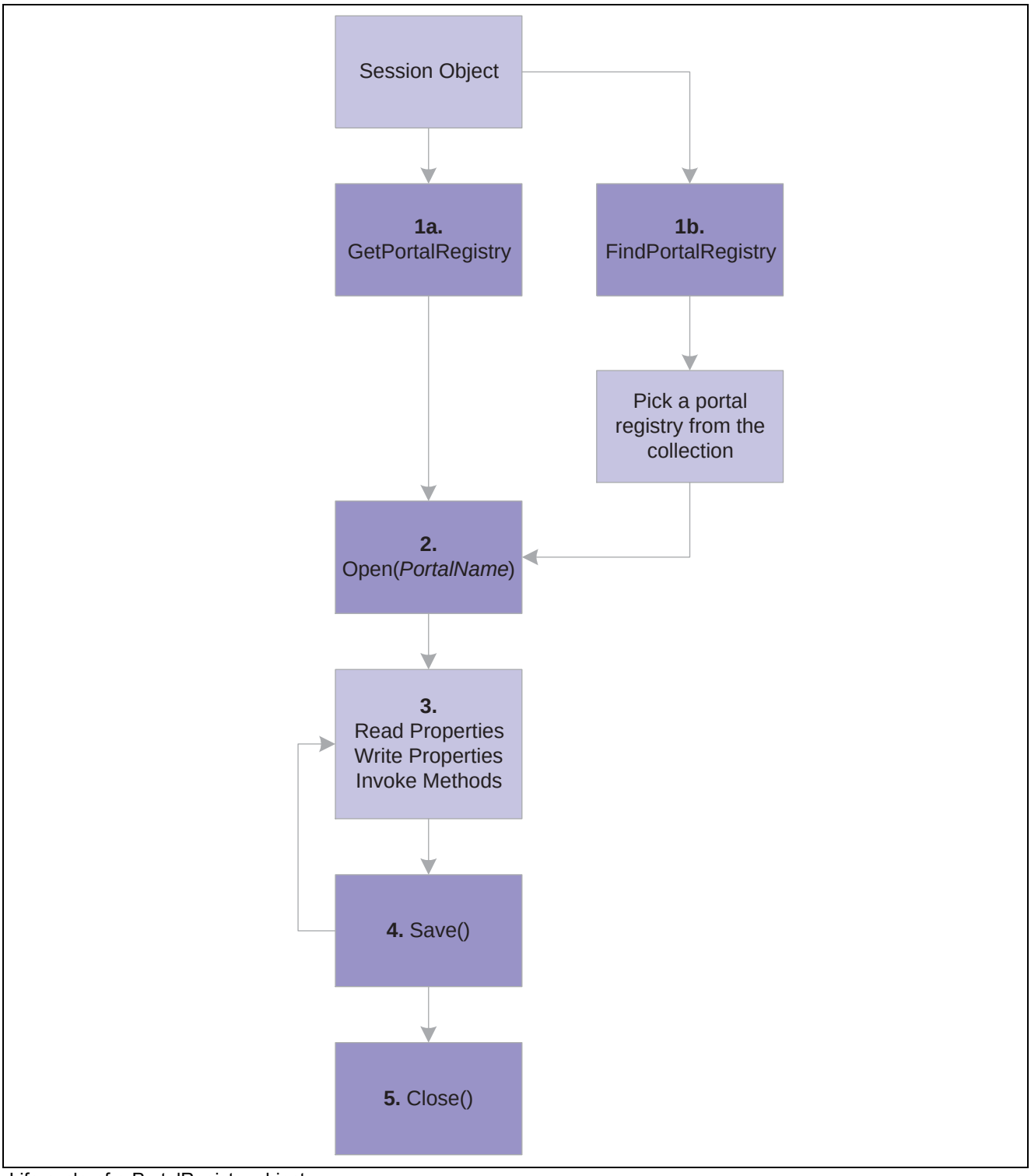
   Some methods commit data to the database only when the parent object is saved. However, other methods cause data to be committed to the database as soon as they are executed, like DeleteItem on a folder. Methods that automatically make database changes are noted as such in their description.

   See Chapter 27, "PortalRegistry Classes," Saving Content Considerations, page 1018.

5. When you finish all operations for a PortalRegistry, close the object.

Note. PeopleSoft recommends that you open and close every PortalRegistry in a single PeopleCode event. You shouldn't open a PortalRegistry object and keep it open across multiple events. You should also keep only one PortalRegistry object open at a time.

 Copyright © 1988-2007, Oracle. All rights reserved.



Life cycle of a PortalRegistry object

Examples of using the PortalRegistry objects are provided at the end of this chapter.

See Also

Chapter 27, "PortalRegistry Classes," PortalRegistry Classes Example, page 1169

Copyright © 1988-2007, Oracle. All rights reserved.

Viewing the PortalRegistry Class Hierarchy

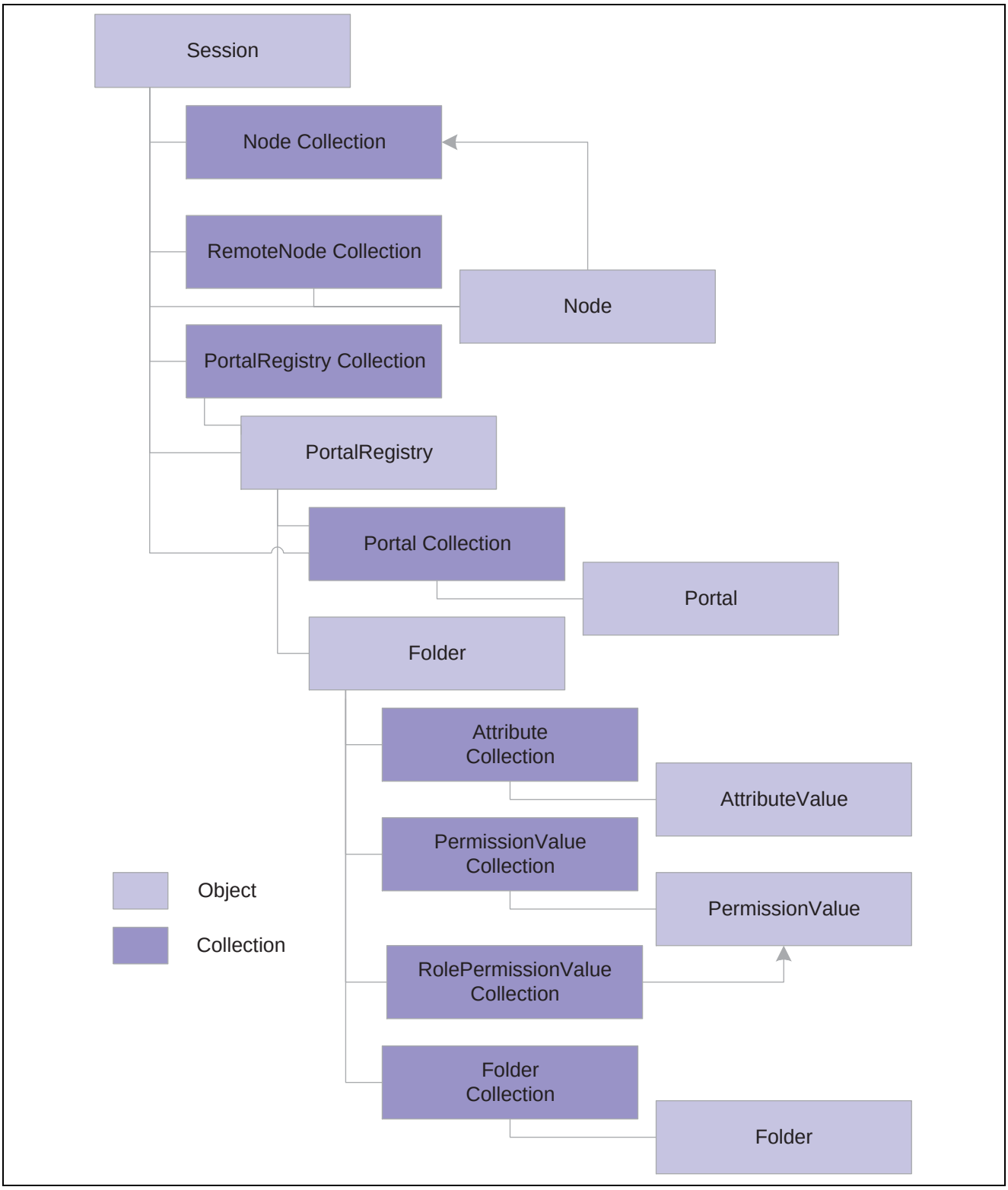
There are many different classes used with the PortalRegistry API. The following flowchart shows the different classes, and how they are interrelated.

Hierarchy Considerations

The following are considerations for the representation of the hierarchy:

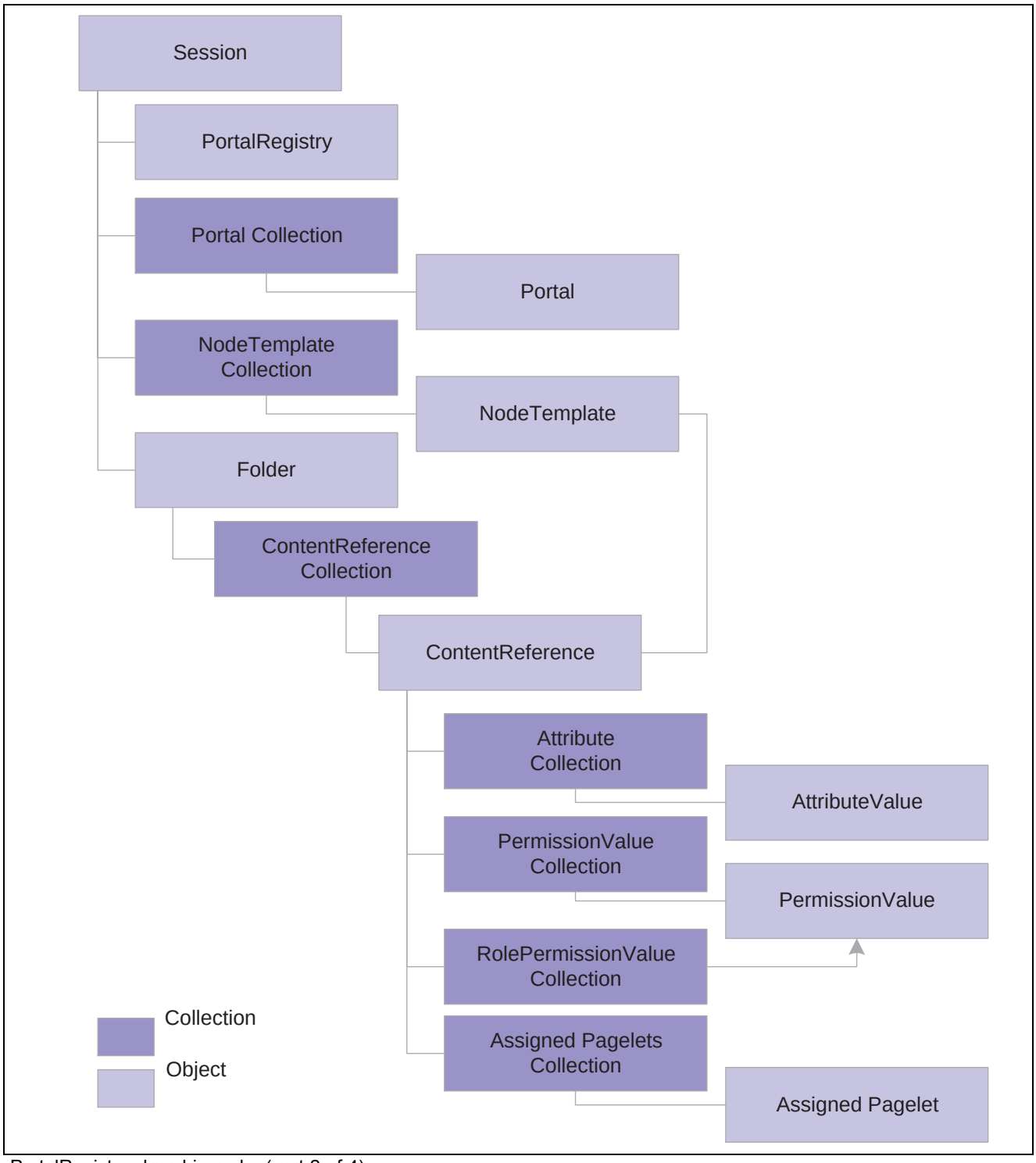
- The flowchart shows only two levels of folders. In reality, you can embed as many levels of folders as you need.
- From a PortalRegistry object, you must access the root folder before you can access the sub-folders for that PortalRegistry.

Copyright © 1988-2007, Oracle. All rights reserved.



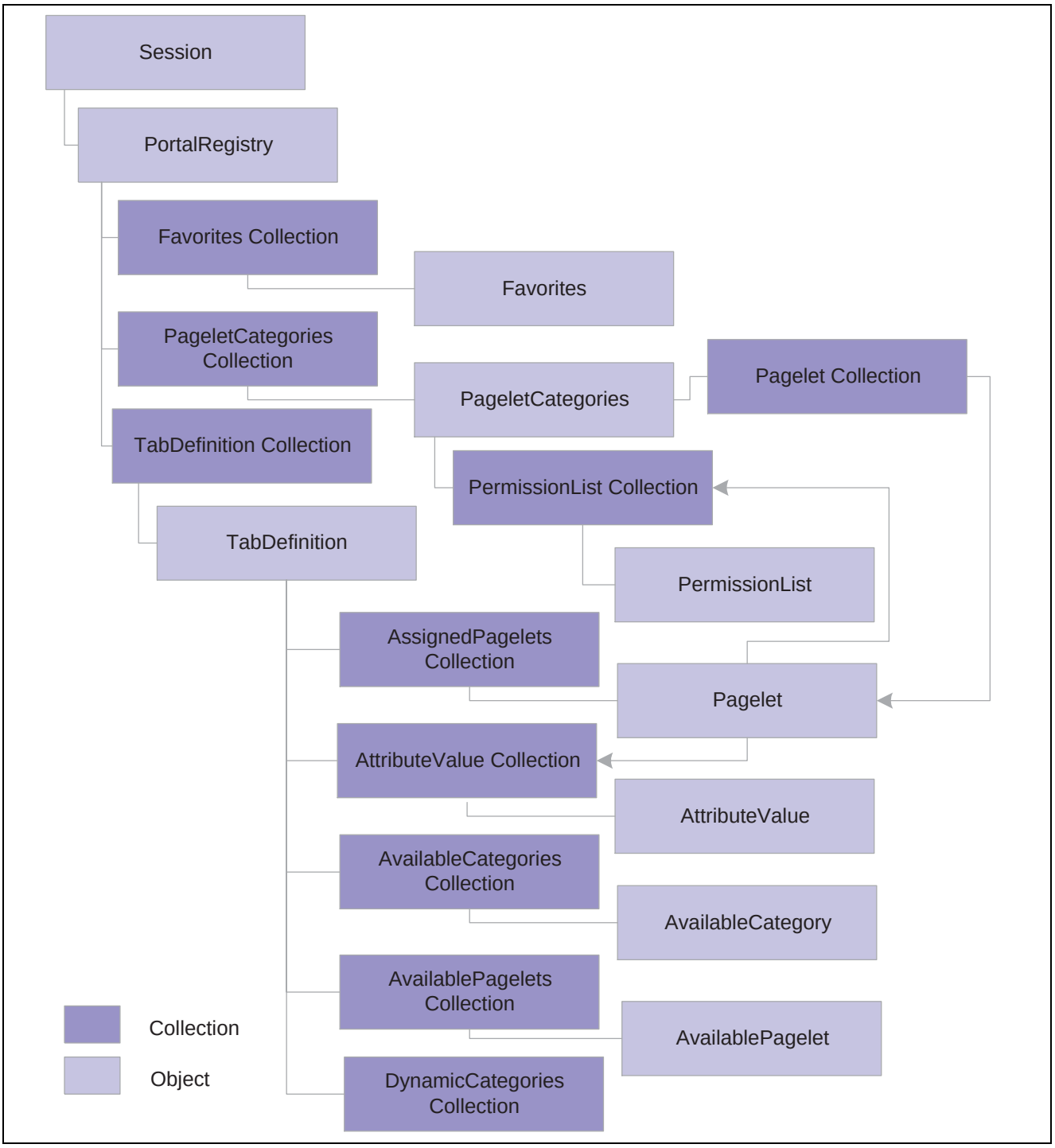
PortalRegistry class hierarchy (part 1 of 4)

Copyright © 1988-2007, Oracle. All rights reserved.



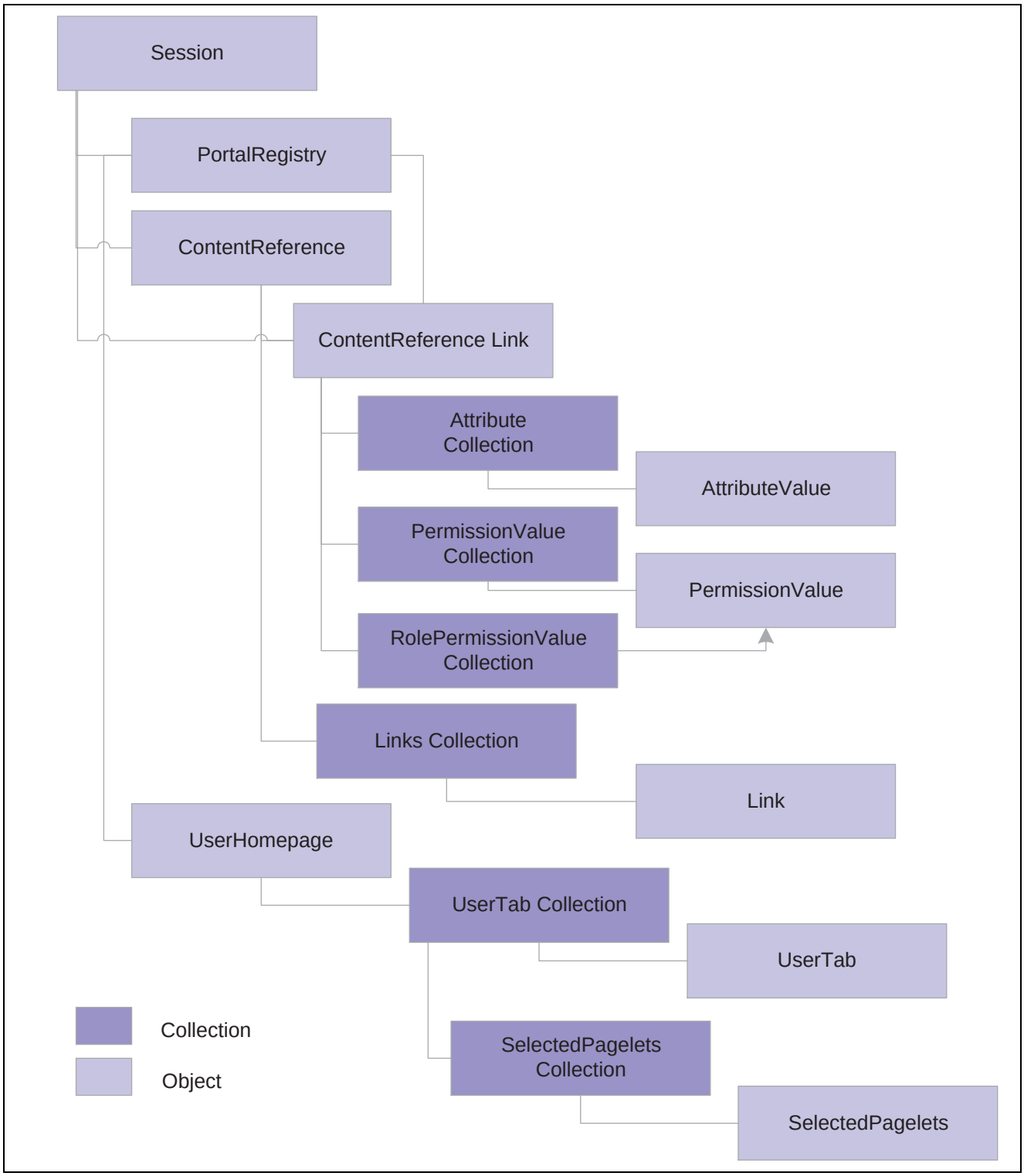
PortalRegistry class hierarchy (part 2 of 4)

Copyright © 1988-2007, Oracle. All rights reserved.



PortalRegistry class hierarchy (part 3 of 4)

Copyright © 1988-2007, Oracle. All rights reserved.



PortalRegistry class hierarchy (part 4 of 4)

Copyright © 1988-2007, Oracle. All rights reserved.

Using Content References

Content references have a number of properties, but several properties work together to define the type of Content reference. The values of these properties are interdependent, that is, the value of one indicates the values of others. The properties are:

- UsageType
- TemplateType
- URLType
- Nodes and URL

UsageType

This is the primary specifier for the type of content reference. There are a number of different types of content references, but all content references can be categorized into the following major types:

- Target
- Template
- Portal Component

Target

The *target* is the page referenced by a URL in the user's browser. This is the main transaction or page that the user wants, and the portal may place other content around this target page based on the template. The template describes any other content, and where to place it, on the page. The template is either gotten from the content reference or a set of default templates.

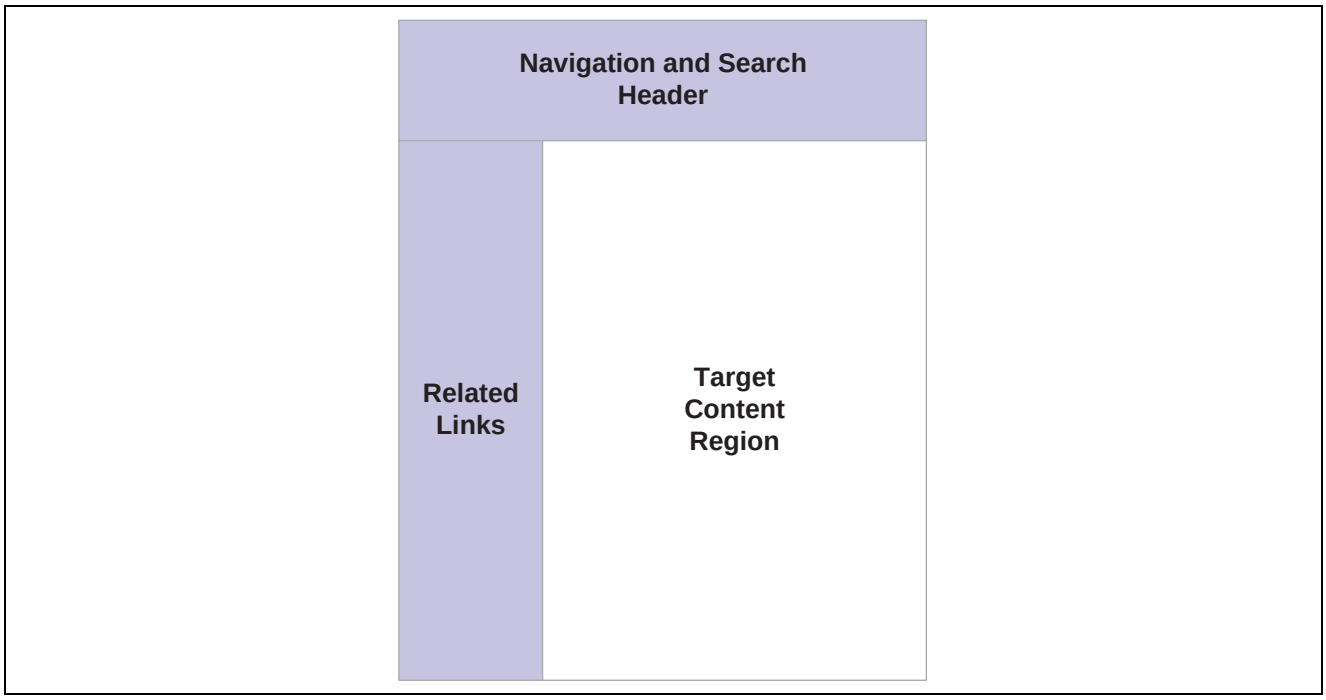
Content reference links can only be created for content references that have a type of target.

Template

A portal *template* defines how the portal creates a user's page. It's an HTML document that describes the content and where the content is placed. The template specifies one target and zero or more portal components.

The portal template in the following example is comprised of four separate template components: one for the navigation and search header, one for related links, one for the target content region, and one for the overall template which specifies where the other components should be placed. At runtime, the target content region would be filled by the HTML returned by the target page, as would the other template component regions.

Copyright © 1988-2007, Oracle. All rights reserved.



A portal template

See *Enterprise PeopleTools 8.49 PeopleBook: Internet Technology*, "Working with Portal Templates".

Portal Component

A *portal component* is an HTML document, or something that produces an HTML document. The portal component must be defined within a template.

A portal component could be one of a Homepage tab, component reference, or Homepage Pagelet.

UsageType Values

The following table matches the general types of target, template, and portal component to the actual values of the UsageType property.

General Type	UsageType Value
Target	Target (TARG)
Template	Frame template (FRMT) or HTML template (HTMT)
Pagelet	Pagelet (HPGC)
Homepage Tab	Homepage Tab (HPGT)

More specifically:

- A UsageType value of TARG specifies a content reference that is a target.
- A UsageType value of FRMT specifies a content reference that is an HTML frame template.

 Copyright © 1988-2007, Oracle. All rights reserved.

- A UsageType value of HTMP specifies a content reference that is an HTML template.
- A UsageType value of HPGC specifies a content reference that is a PeopleSoft homepage component (pagelet).
- A UsageType value of HPGT specifies a Homepage tab.
- A UsageType value of LINK specifies a content reference link.

TemplateType

This property is valid only when the UsageType property is a target. For target type content references (TARG) this controls whether the portal looks for and uses a template to wrap the target.

TemplateType	Meaning
NONE	There is no template for this target
HTML	There is some kind of HTML template for this target

URLType

This property gives information about what format the URL is in.

URLType Value	Meaning
UEXT	URL points to a non-PeopleSoft (external) URL
UMPG	URL points to a PeopleSoft mobile page
UPGE	URL points to a component
UPHP	URL points to a homepage tab
UPTM	URL points to a template
USCR	URL points to an iScript
UGEN	URL points to a generic PeopleSoft URL.

The URL property is always required (it's one of the parameters for the InsertItem method.) The format of this parameter (or property) depends on the other properties.

See Also

Enterprise PeopleTools 8.49 PeopleBook: Internet Technology, "Understanding Portal Technology"

Copyright © 1988-2007, Oracle. All rights reserved.

Nodes and URL

Nodes and the URL property work together and are interrelated. The node is how to 'register' a logical name for a webserver (the webserver name, on the servlet, and so on) not the actual details. This way, content references don't change when a webserver changes.

For example, suppose you had content that you referenced on the HRMS webserver. However, the machine name for that server changed. You can change the URI of the Node, instead of changing every content reference that referred to that content.

At least one node must be specified as the default local node. Nodes are also required for PeopleSoft components and iScripts.

Considerations When Using Nodes and the URL Property

When a content reference is created it is 'registered' with its URL (the portal, and others, typically find a content reference by its URL). The node is used to create a logical name for the webserver, servlet, and so on, so these details are not included in a content reference's URL. When details of a webserver change, such as at installation time, only the URI for the content provider must change. You don't have to change the URL for any content references.

When the node is specified, the content provider's URI is concatenated with the URL property.

The format of the URL property depends on the content it's pointing to.

Summary

The following table summarizes the interrelations between the different content reference properties.

UsageType	TemplateType	StorageType	URLType
Target (TARG)	HTML	Remote (RMTE)	Component (UPGE)
Target (TARG)	HTML	Remote (RMTE)	Internet Script (USCR)
Target (TARG)	HTML	Remote (RMTE)	External (UEXT)
Target (TARG)	NONE	Remote (RMTE)	Component (UPGE)
Target (TARG)	NONE	Remote (RMTE)	Internet Script (USCR)
Target (TARG)	NONE	Remote (RMTE)	External (UEXT)
Frame template (FRMT), HTMP template	NONE	Remote (RMTE)	Internet Script (USCR)
Frame template (FRMT), HTMP template	NONE	Local (LOCL)	N/A
Homepage (HPGT)	NONE	Local (LOCL)	N/A

 Copyright © 1988-2007, Oracle. All rights reserved.

UsageType	TemplateType	StorageType	URLType
Template component (TMPC), Homepage component (HPGC)	N/A	Remote (RMTE)	Component (UPGE)
Template component (TMPC), Homepage component (HPGC)	N/A	Remote (RMTE)	Internet Script (USCR)
Template component (TMPC), Homepage component (HPGC)	N/A	Remote (RMTE)	External (UEXT)
Template component (TMPC), Homepage component (HPGC)	N/A	Local (LOCL)	N/A

Naming Conventions

If you create two content references or with the exact same URL, the second one fails.

For example, suppose you create an external content reference with a URL of www.peoplesoft.com. Then you create a second content reference that has a node of PeopleSoft, whose URI is www.peoplesoft.com. The creation of the second content reference fails because the URL already exists.

If you have multiple nodes with the same URI, FindCRefByURL looks for the specified content reference with all those nodes.

If you have multiple nodes with the same URI, but no registered content references, the system uses the template from the alphabetically first node it finds.

Pagelet, node, and portal registry names can consist of any combination of letters, digits and underscores, but they must not contain any spaces or begin with a digit.

Content reference link names cannot have start with numbers, and can not have special characters and spaces.

Deleting Content Considerations

Be extremely careful when you delete any content. There may be more than one object relying on the content you delete.

- If you try to delete a template currently used by a content reference, the node template is set as the default template for the portal, you receive an error when you try to save the item.
- If you delete a homepage template for a user, the system tries to use the default user's template first, before resorting to the portal default template.

Copyright © 1988-2007, Oracle. All rights reserved.

- If you try to delete a content provider that is currently used by a content reference, you receive an error and cannot save the PortalRegistry object.

Warning! If you delete a folder, you delete *all* content in the folder. If you delete a folder that contains other folders, that is, a parent folder, all the child folders, and all the content references are deleted.If you delete a PortalRegistry, you delete *everything*. Your entire PortalRegistry is gone, all the folders, content references, templates, and so on. Do not delete a PortalRegistry object unless you are absolutely certain you want to.

Saving Content Considerations

The following PortalRegistry classes have Save methods:

- PortalRegistry
- Folder
- Content reference
- Content reference links
- TabDefinition
- PageletCategory
- Pagelet
- Homepage
- Favorites collection

This means if you change a folder, you must save the folder. If you change a folder and only save at the PortalRegistry level, your changes are *not* saved.

Some classes do *not* have a Save method. For these classes, you must save the parent object.

- If you change a node, node template, or remote portal, you must use the PortalRegistry Save method.
- If you change an AttributeValue you must use the Save method with the item that contains the AttributeValue (folder, content reference, PageletCategory, or Pagelet).
- If you change a PermissionValue, you must use the Save method with the item that contains the PermissionValue (folder, content reference, PageletCategory or Pagelet).
- If you change a UserTab or a SelectedPagelet, you must use the Homepage Save method.

Data Type of a PortalRegistry Object

PortalRegistry objects, and all objects instantiated from a PortalRegistry object, are declared as data type ApiObject. For example,

```
Local ApiObject &MyRegistry;
Local ApiObject &MyFolder, &MyAttributeValue;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. PortalRegistry objects can only be declared as Local.

Considerations Using Local Variables

When a local variable has a reference to an object and the end-user clicks the Back button on a browser, the local variable is set to NULL. This is always logged in the trace file.

Considerations Using Global Variables

Global variables are not available to a portal or applications on separate databases. They are available only on applications and Portals in the *same* database.

Scope of a PortalRegistry Object

A PortalRegistry object can be instantiated from the following language environments:

- PeopleCode
- C/C++
- Java

See Also

Chapter 27, "PortalRegistry Classes," PortalRegistry Classes Example, page 1169

PortalRegistry Reference

The following sections provide more detail of the properties, methods, and other objects that you can use with a PortalRegistry object. If you don't want to programmatically change a portal registry, you can use the Portal Administration Tool pages instead.

See Also

Enterprise PeopleTools 8.49 PeopleBook: Internet Technology, "Understanding Portal Technology"

Session Class Methods

PortalRegistry objects don't have any built-in functions. They are instantiated from a session object. In this section, we discuss the Session Object methods. The methods are listed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSession

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "System Variables," %Session

FindPortalRegistries

Syntax

FindPortalRegistries(*Name*)

Description

The FindPortalRegistries method returns a reference to a PortalRegistry Collection filled with zero or more PortalRegistry objects matching the *Name* parameter. The *Name* parameter takes a string value.

You can use a partial key to get a smaller subset of the PortalRegistry collection. For example, to get a collection of all the PortalRegistry objects whose names start with the letter "B", specify just the letter B for *Name*:

```
&MyColl = &MySession.FindPortalRegistries("B");
```

Parameters

Parameter	Description
Name	Specify the name of the PortalRegistry object to find. This parameter takes a string value.

Returns

A PortalRegistry Collection object.

Example

The following example returns a collection with references to all PortalRegistry objects.

```
Local ApiObject &MySession;
Local ApiObject &MyColl;

&MySession = %Session;
&MyColl = &MySession.FindPortalRegistries("");
```

GetLocalNode

Syntax

GetLocalNode()

Description

Use the GetLocalNode method to return a reference to the node defined as the local node for this session.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to a node object if successful, Null otherwise.

Example

```
&Node = %Session.GetLocalNode();
```

See Also

Chapter 27, "PortalRegistry Classes," Node Class, page 1046

GetNodes

Syntax

```
GetNodes()
```

Description

Use the GetNodes method to return a collection of both local and remote nodes for the session.

Parameters

None.

Returns

A reference to a collection of nodes if successful, Null otherwise.

Example

```
&NodeColl = %Session.GetNodes();
```

See Also

Chapter 27, "PortalRegistry Classes," Node Collection, page 1049

GetPortalRegistry

Syntax

```
GetPortalRegistry()
```

Description

The GetPortalRegistry method returns an empty PortalRegistry object. You can then open or delete an existing PortalRegistry, or create a new one.

Parameters

None.

Returns

An empty PortalRegistry object.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local ApiObject &MyPortal;

&MyPortal = %Session.GetPortalRegistry();
&PORTAL_NAME = %Request.GetParameter("PORTAL_NAME");
&Portal.Open(PORTAL_NAME);
```

GetRemoteNodes

Syntax

GetRemoteNodes()

Description

Use the GetRemoteNodes method to return a collection of nodes for the session.

Note. This method returns both local and remote nodes.

Parameters

None.

Returns

A reference to a collection of nodes if successful, Null otherwise.

Example

```
&NodeColl = %Session.GetNodes();
```

See Also

Chapter 27, “PortalRegistry Classes,” RemoteNode Collection, page 1051

PortalRegistry Class

A PortalRegistry object is returned from:

- The GetPortalRegistry session method.
- The PortalRegistry Collection Methods First, ItemByName, or Next.

Note. In addition to the following methods, the PortalRegistry class has methods used with the Search API.

See Also

Chapter 27, “PortalRegistry Classes,” GetPortalRegistry, page 1021

Chapter 27, “PortalRegistry Classes,” PortalRegistry Collection Methods, page 1044

Chapter 27, “PortalRegistry Classes,” Changing PortalRegistry Properties, page 1169

Chapter 35, “Search Classes,” page 1459

 Copyright © 1988-2007, Oracle. All rights reserved.

PortalRegistry Class Methods

In this section, we discuss the PortalRegistry class methods. The methods are discussed in alphabetical order.

Close

Syntax

Close()

Description

The Close method closes the PortalRegistry object, that is, this method sets the object to the state it was in immediately after the GetPortalRegistry was done on the PeopleSoft Session object. Any unsaved changes are discarded. The Close method can be used only on an open PortalRegistry, not a closed one. This means you must have opened the PortalRegistry with the Open method before you can close it.

Parameters

None.

Returns

A Boolean value: True if the PortalRegistry object is successfully closed, False otherwise.

Example

```
&Rslt = &MyRegistry.Close();

If Not &Rslt Then
   /* registry not closed - do error processing */
End-if;
```

See Also

Chapter 27, "PortalRegistry Classes," Open, page 1037 and Chapter 27, "PortalRegistry Classes," Save, page 1040.

CopyObject

Syntax

CopyObject(*sourcePortalName*, *sourceRefType*, *sourceObjName*, *targetPortalName*, *targetPrntFldrName*, *copyChildren*)

Description

Use the CopyObject method to copy folders and content references between portal registries.

If the object being copied has related HTML, the HTML is also copied and named according to the current portal naming convention for HTML objects, which is:

```
PR_<portalname>_<objectname>
```

where <*portalname*> is up to the first 8 characters of the portal name and <*objectname*> is up to the first 17 characters of the object name.

Copyright © 1988-2007, Oracle. All rights reserved.

Considerations Using CopyObject

The source portal name and the target portal name cannot be the same name.

The *targetPrntFldrName* must be a folder that exists in the portal identified by the target portal name. It cannot be null (or empty). If you need to copy an entire portal, the portal must first be "created" using the Create method, before copying over all the folders and content references.

The *copyChildren* parameter is ignored for content references. However, it still must be set to "Yes" or "No".

Parameters

Parameter	Description
sourcePortalName	Specify the name of the portal from which the object is to be copied.
sourceRefType	Specify the type of object to be copied. Values are: • "C" for content references • "F" for folders This parameter is case-sensitive.
sourceObjName	Specify the name of the object to be copied.
targetPortalName	Specify the name of the portal to which the object will be copied.
targetPrntFldrName	Specify the name of the folder that will be the parent folder for the source object.
copyChildren	Specify whether to also copy the children of the source object. This parameter is valid only with folder objects. Values are: • "Yes" copy child objects • "No" do not copy child objects This parameter is case-sensitive.

Returns

A Boolean value: True if object copied successfully, False otherwise.

Example

```
&Success = &Portal.CopyObject("PORTAL", "F", "PEOPLETOOLS_QUALITY", "BOGUS",⇒
 "PORTAL_ROOT_OBJECT", "Yes");
If (&Success = False) Then
   WinMessage("portal copy failed");
   Exit;
End-If;
```

Create

Syntax

Create(*RegistryName*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Create method creates a new PortalRegistry in the PortalRegistry object called *RegistryName*. The specified registry must be a new registry. The Create method returns False if the registry already exists.

The new PortalRegistry is immediately committed to the database. If you change any property of a PortalRegistry after you create the object, use the Save method to commit your changes to the database.

When a registry is created, a folder called *Root* is automatically created. This is the root folder for the registry.

Note. If you're using Visual Basic, you must check that the PortalRegistry is actually created. If you use a duplicate name, a zero is returned, but no error results.

Parameters

Parameter	Description
RegistryName	The name of the registry to create. This parameter takes a string value. If you specify a registry that already exists, this method returns a False value.

Returns

A Boolean value: True if the PortalRegistry object is successfully created, False otherwise.

Example

```
&PORTAL_NAME = MY_PORTAL_RECORD.PORTALNAME;
&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Create(&PORTAL_NAME) Then;
   /* portal not created - do error processing */
End-If;
```

See Also

Chapter 27, "PortalRegistry Classes," Open, page 1037, Chapter 27, "PortalRegistry Classes," Save, page 1040, Chapter 27, "PortalRegistry Classes," Close, page 1023 and Chapter 27, "PortalRegistry Classes," Delete, page 1027.

CreateContentRefLink

Syntax

CreateContentRefLink(*LinkName*, *LinkLabel*, *LinkParent*, *CRefPortalName*, *CRefObjectName*)

Description

Use the CreateContentRefLink method to create a link to any content reference in any portal in a local database.

For example, using this method, you can create a link to a content reference in the EMPLOYEE portal, to a content reference in the CUSTOMER portal, or to a content reference in the current portal.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
LinkName	Specify the name of the link as a string. This is the ID of the link. The name is validated as that of the content reference name. It cannot start with a number, and can not have special characters and spaces. This property is not translated.
LinkLabel	Specify the label of the link as a string. This property can be translated. This is the label that appears on the left hand navigation menu If this value is blank, a linked content reference label is displayed instead.
LinkParent	Specify the parent folder of the link, as a string.
CRefPortalName	Specify the portal name of the content reference, to which the link is pointing, as a string.
CRefObjectName	Specify the ID of the content reference to which the link is pointing, as a string.

Returns

A reference to a newly created CRefLink object.

Example

```
Local ApiObject &Portal, &CRef, &CReflink;
&Portal = PortalOpen();
&CReflink = &Portal.CreateContentRefLink(<Unique Link Name> , <LinkLable>,⇒
 "<Link's Parent folder >", "<Cref Portal Name>", <Cref Object Name>);

/*... Use the link object */

&Portal.Close();
```

CreateRemote

Syntax

CreateRemote(*PortalName*, *RemoteNodeName*)

Description

Use the CreateRemote method to create a remote portal. Although you can have more than one portal on a node, they must all be uniquely named.

Parameters

Parameter	Description
PortalName	Specify the name of the new portal to create on the remote node.
RemoteNodeName	Specify the name of the remote node to create the new portal on.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the PortalRegistry object is successfully created, False otherwise.

See Also

Chapter 27, "PortalRegistry Classes," Create, page 1024 and Chapter 27, "PortalRegistry Classes," Portals, page 1043.

Delete

Syntax

Delete(*RegistryName*)

Description

The Delete method deletes the PortalRegistry *from the database,* including any data and tables.

Warning! If you delete a PortalRegistry, you delete *everything*. Your entire PortalRegistry is gone, all the folders, content references, templates, and so on. Do not delete a PortalRegistry object unless you are absolutely certain that you want to.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

The Delete method can only be used with a *closed* registry, it cannot be used on an open registry. Before you use the Delete method, you must explicitly close the PortalRegistry object (with the Close method.) The Delete method returns False if you try to delete an open PortalRegistry object.

Parameters

Parameter	Description
RegistryName	The name of the registry to delete. This parameter takes a string value. If you specify a registry that doesn't exist, this method returns a False value.

Returns

A Boolean value: True if the PortalRegistry object is successfully deleted, False otherwise.

Example

The following example deletes all PortalRegistry objects that start with HRMS_99.

```
Local ApiObject &MySession;
Local ApiObject &MyPortal;

&MySession = %Session;

&MyPortal = &MySession.GetPortalRegistry();

&MyPortal.Delete("HRMS_99");
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 27, "PortalRegistry Classes," Close, page 1023 and Chapter 27, "PortalRegistry Classes," Open, page 1037.

DeleteHomepage

Syntax
DeleteHomepage()

Description

The DeleteHomepage method deletes the current user's homepage. This method is valid only after a user has opened a PortalRegistry object.

Parameters

None.

Returns

A Boolean value: True if the homepage is successfully deleted, False otherwise.

See Also
Chapter 27, "PortalRegistry Classes," UserHomepage Class, page 1154

FindCRefByName

Syntax
FindCRefByName(*Name*)

Description

The FindCRefByName method returns the content reference object corresponding to *Name*. The name is a unique identifier for each content reference.

Considerations on Returned Content References

This method returns content reference objects that aren't yet valid as well as ones that are no longer valid. When you create your program, you must check for these invalid values if you don't want to use them. You can check using ValidTo, ValidFrom, or IsVisible.

This method returns content reference objects that you aren't authorized to. When you create your program, you should always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	Description
Name	A unique name within the registry that identifies the content reference. This parameter takes a string value. This parameter uses the name, not the label, of a content reference.

Returns

What this method returns depends on the condition of the content reference:

 Copyright © 1988-2007, Oracle. All rights reserved.

- If the content reference name is valid and the end-user has access to the content reference, a content reference object is returned.
- If the content reference is valid but the end-user doesn't have access to it a content reference object is returned, however, the only property you can access is the Authorized property.
- If you specified an invalid name this method returns NULL.

Example

The following example returns a GLOBAL_PAYROLL content reference object:

```
&CRef = &Portal.FindCRefByName("GLOBAL_PAYROLL");
```

See Also

Chapter 27, "PortalRegistry Classes," Content Reference Class, page 1070

FindCRefByURL

Syntax

FindCRefByURL(*URL*)

Description

The FindCRefByURL method returns the content reference object corresponding to *URL*. The URL specified by *URL* must be an absolute URL.

Note. The portal registry API needs the current URI of the local node to work. During runtime, it gets this information from the current webserver. For Application Engine programs, there isn't a current webserver, so it has to get this information from the database.

If you have multiple content providers with the same URI, FindCRefByURL looks for the specified content reference with all those content providers.

To access pagelet categories and their associated pagelets, use the PageletCategories collection from the PortalRegistry object, not FindCRefByURL.

Considerations on Returned Content References

This method returns content reference objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns content reference objects that the end-user isn't authorized to. When you create your program, always check the Authorized property. This is the only property you can view from an object that the end-user isn't authorized to view.

Parameters

Parameter	Description
URL	A URL that represents the content. This parameter takes a string value. This URL must be an absolute URL. This parameter is case-insensitive.

Returns

What this method returns depends on the condition of the content reference:

Copyright © 1988-2007, Oracle. All rights reserved.

- If the end-user has access to the URL, a content reference object is returned.
- If the content reference is registered, but the end-user doesn't have access to it, a content reference is returned, but the only property you can access is the Authorized property.
- If a URL isn't registered or is invalid, this method returns NULL.

Example

The following example finds a content reference from a URL:

```
&UserCRef = &Portal.FindCRefByURL("http://www.PeopleSoft.Com");
```

See Also

Chapter 27, "PortalRegistry Classes," GetQualifiedURL, page 1034 and Chapter 27, "PortalRegistry Classes," QualifiedURL, page 1078.

Chapter 27, "PortalRegistry Classes," Content Reference Class, page 1070

FindCRefForURL

Syntax

FindCRefForURL(*URL*)

Description

The FindCRefForURL method returns the content reference object corresponding to *URL*. The URL specified by *URL* must be an absolute URL.

If the exact content reference is not found, this method tries to look for the content reference again, after stripping off the query portion of the URL.

If you don't want the system searched without the query string, use the FindCRefByURL method.

Note. The portal registry API needs the current URI of the local node to work. During runtime, it gets this information from the current webserver. For Application Engine programs, there isn't a current webserver, so it has to get this information from the database.

If you have multiple content providers with the same URI, FindCRefForURL looks for the specified content reference with all those content providers.

To access pagelet categories and their associated pagelets, use the PageletCategories collection from the PortalRegistry object, not FindCRefForURL.

Considerations on Returned Content References

This method returns content reference objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns content reference objects that the end-user isn't authorized to. When you create your program, always check the Authorized property. This is the only property you can view from an object that the end-user isn't authorized to view.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
URL	A URL that represents the content. This parameter takes a string value. This URL must be an absolute URL. This parameter is case-insensitive.

Returns

What this method returns depends on the condition of the content reference:

- If the end-user has access to the URL, a content reference object is returned.
- If the content reference is registered, but the end-user doesn't have access to it, a content reference is returned, but the only property you can access is the Authorized property.
- If a URL isn't registered or is invalid, this method returns NULL.

Example

The following example finds a content reference from a URL. If the content reference isn't found from the full URL, the query string is stripped and the system searches again:

```
&UserCRef = &Portal.FindCRefForURL("http://www.peoplesoft.com/crefs/psportal⇒
/technologies/?url=http%3a%2f%2faugust2004%2fipass.html");
```

See Also

Chapter 27, "PortalRegistry Classes," GetQualifiedURL, page 1034 , Chapter 27, "PortalRegistry Classes," QualifiedURL, page 1078 and Chapter 27, "PortalRegistry Classes," FindCRefByURL, page 1029.

Chapter 27, "PortalRegistry Classes," Content Reference Class, page 1070

FindCRefLinkByName

Syntax

FindCRefLinkByName(*LinkName*)

Description

Use the FindCRefLinkByName method to find the existing link in the current portal. If the link is found a reference to CRefLink object is returned. A Null value is returned if the CRefLink is not found in the database, and the error message is added to message collection

Considerations on Returned Links

This method returns links that aren't yet valid as well as ones that are no longer valid. When you create your program, you must check for these properties (ValidTo and ValidFrom) if you don't want to use them. This method may also return objects that you aren't authorized to. When you create your program, you should always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
LinkName	Specify the name of the link as a string. This is the ID of the link.

Returns

A reference to an existing CRefLink object. or a Null value if the CRefLink is not found.

FindFolderByName

Syntax

FindFolderByName(*Name*)

Description

The FindFolderByName method returns the Folder object corresponding to *Name*. The name is a unique identifier for each folder.

Considerations on Returned Folders

This method returns Folder objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns folder objects that you aren't authorized to. When you create your program, always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	Description
Name	A unique name within the registry that identifies the folder. This parameter takes a string value. This parameter takes the name of a folder, not the label.

Returns

What this method returns depends on the condition of the folder:

- If the folder name is valid and the end-user has access to the folder, a folder object is returned.
- If the folder is valid but the end-user doesn't have access to it a folder object is returned, however, the only property you can access is the Authorized property.
- If you specified an invalid name this method returns NULL.

Example

The following example returns a folder named ROOT:

```
&MyFolder = &MyPortal.FindFolderByName("ROOT");
```

The following example returns the folder object for an already instantiated content reference:

```
&Folder = &Portal.FindFolderByName(&CRef.ParentName);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 27, "PortalRegistry Classes," Folder Class, page 1059

FindPgltByName

Syntax

FindPgltByName(*PageletName*)

Description

The FindPgltByName method returns the pagelet object corresponding to *PageletName*. The name is a unique identifier for each pagelet.

Considerations on Returned Pagelets

This method returns pagelet objects that aren't yet valid as well as ones that are no longer valid. When you create your program, check for these properties (ValidTo and ValidFrom) if you don't want to use them.

This method returns pagelet objects that you aren't authorized to. When you use create your program, always check the Authorized property. This is the only property you can view from an object that you're not authorized to view.

Parameters

Parameter	Description
PageletName	A unique name within the registry that identifies the Pagelet. This parameter takes a string value. This parameter takes the name of a Pagelet, not the label.

Returns

What this method returns depends on the condition of the Pagelet:

- If the Pagelet name is valid and the end-user has access to the Pagelet, a Pagelet object is returned.
- If the Pagelet is valid but the end-user doesn't have access to it a Pagelet object is returned, however, the only property you can access is the Authorized property.
- If you specified an invalid name this method returns NULL.

Example

The following example returns a pagelet named HomePg_Dictionary:

```
&MyPglt = &MyPortal.FindPgltByName("HomePg_Dictionary");
```

See Also

Chapter 27, "PortalRegistry Classes," Pagelet Class, page 1145

GetAbsoluteContentURL

Syntax

GetAbsoluteContentURL(*NodeName*, *URL*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The GetAbsoluteContentURL method returns the absolute unwrapped simple URL of the content, in the context of the current portal. For example, if the PortalRegistry object accessed a portal called Employees, and the method were called like this:

```
&Registry.getAbsoluteContentURL(Node.CRM,  "/c/SERVICES.ORDERS.GBL");
```

It would return the following string:

```
http://crmserver/servlets/psc/crmHome/Employees/CRM/c/SERVICES.ORDERS.GBL
```

In the returned string, the portion of the string from server name through *PS_Home* (crmHome in the example) are the ones associated with the content node (CRM in the example). The portal is the current portal, and the rest of the URL is the URL string passed in.

Parameters

Parameter	Description
NodeName	Specify the name of the node that contains the content. You can also use a string, such as %Node, for this value.
URL	Specify the relative URL pointing to the content that you want the absolute URL generated for.

Returns

A string containing the absolute URL.

GetDefaultHPTabOID

Syntax

GetDefaultHPTabOID()

Description

Use the GetDefaultHPTabOID method to return the name of the first homepage tab that is found and authorized for the current UserID.

The search order of the tabs depends on Sequence number of the tab name and is sorted alphabetically.

Parameters

None.

Returns

A string

GetQualifiedURL

Syntax

GetQualifiedURL(*ContentProvider, RelativeURL*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Note. This method is maintained only for backward compatibility. If your existing code uses this method, it actually returns the value from GetAbsoluteContentURL method. New applications should use the GetAbsoluteContentURL method.

See Also

Chapter 27, "PortalRegistry Classes," GetAbsoluteContentURL, page 1033

GrantPermissionForComponent

Syntax

```
GrantPermissionForComponent(MenuName, ComponentName, Market, PermListName, NodeName)
```

Description

Use the GrantPermissionForComponent method to grant the specified permission and cascaded upwards on parent folders to the specified component. In addition, the specified permission is granted to any component references that point to the specified component, and any parent folders.

Components that have query strings are also searched and permissions are applied on them.

If you use the string "LOCAL_NODE" as *NodeName*, the system uses the node name currently defined as local.

All component entries in the portal registry are affected for all the portals. Not just the current portal.

Parameters

Parameter	Description
MenuName	Specify the menu name, as a string, that the component you want to grant permissions for is associated with.
ComponentName	Specify the component name, as a string, that you want to grant permissions for.
Market	Specify the market, as a string, associated with the component that you want to grant permissions for.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to grant permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

The My Profile Component Reference points to the USERMAINT_SELF component. Using the following example, "ALLPANLS" permission is granted to the MY_PROFILE component reference, as well as "MY INFO", the parent folder.

```
&Portal.GrantPermissionForComponent("MAINTAIN_SECURITY", "USERMAINT_SELF", "GBL",⇒
 "ALLPANLS", "LOCAL_NODE");
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 27, “PortalRegistry Classes,” RevokePermissionForComponent, page 1038

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, “Setting Up Permission Lists”

GrantPermissionForScript

Syntax

```
GrantPermissionForScript(RecordName, FieldName, EventName, FuncName,
PermListName, NodeName)
```

Description

Use the GrantPermissionForScript method to grant the specified permission to the specified iScript. In addition, the specified permission is granted to any component references that point to the specified iScript, and any parent folders.

If you use the string "LOCAL_NODE" as *NodeName*, the system uses the node name currently defined as local.

All iScript entries in the portal registry are affected for all the portals. Not just the current portal.

Parameters

Parameter	Description
RecordName	Specify the record name, as a string, of the record containing the iScript. All iScripts are contained on records whose names start with "WEBLIB".
FieldName	Specify the name of the field, as a string, containing the iScript.
EventName	Specify the name of the event, as a string, containing the iScript. Generally iScripts are contained in the FieldFormula event.
FuncName	Specify the name of the function, as a string, containing the iScript.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to grant permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

```
&Portal.GrantPermissionForScript("WEBLIB_ALERT", "ALERTCOUNT", "FieldFormula",⇒
 "Connect_Alert", "ALLPGS", "Local_Node");
```

See Also

Chapter 27, “PortalRegistry Classes,” RevokePermissionForComponent, page 1038

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, “Setting Up Permission Lists”

 Copyright © 1988-2007, Oracle. All rights reserved.

Open

Syntax

Open(*RegistryName*)

Description

The Open method opens the PortalRegistry specified by the parameters. The registry must already exist. The Open method can be used only with a *closed* PortalRegistry, it cannot be used on an open registry.

Parameters

Parameter	Description
RegistryName	The name of the registry to open. This parameter takes a string value. If you specify a registry that doesn't exist, this method returns a False value.

Returns

A Boolean value: True if the PortalRegistry object is successfully opened, False otherwise.

Example

In the following example, the name of the portal is stored as the value of a field in a record.

```
&PORTAL_NAME = EO_PE_REG_AET.PORTAL_NAME;
&Portal = %Session.GetPortalRegistry();

&Portal.Open(&PORTAL_NAME);
```

See Also

Chapter 27, "PortalRegistry Classes," Open, page 1037, Chapter 27, "PortalRegistry Classes," Save, page 1040, Chapter 27, "PortalRegistry Classes," Close, page 1023 and Chapter 27, "PortalRegistry Classes," Delete, page 1027.

PermissionListDelete

Syntax

PermissionListDelete(*PermListName*)

Description

Use the PermissionListDelete method to delete the specified permission list from the PortalRegistry.

If you try to delete a permission list that is still in use, you receive an error when you try to save the object.

The permission list is deleted from all the portal objects for all the portal.

Parameters

Parameter	Description
PermListName	Specify the permission list to delete from the PortalRegistry.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the permission list is successfully deleted, False otherwise.

PermissionListSaveAs

Syntax

PermissionListSaveAs(*PermListSourceName, PermListTargetName*)

Description

Use the PermissionListSaveAs method to copy the specified permission list in the PortalRegistry.

Parameters

Parameter	Description
PermListSourceName	Specify the name of the PermissionList to copy.
PermListTargetName	Specify the name that you want to give the new PermissionList.

Returns

A Boolean value: True if the permission list is successfully copied, False otherwise.

RevokePermissionForComponent

Syntax

RevokePermissionForComponent(*MenuName, ComponentName, Market, PermListName, NodeName*)

Description

Use the RevokePermissionForComponent method to revoke the specified permission to the specified component. In addition, the specified permission is revoked for any component references that point to the specified component.

If you use the string "LOCAL_NODE" as *NodeName*, the system uses the node name currently defined as local.

All component entries in the portal registry are affected for all the portals. Not just the current portal.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
MenuName	Specify the menu name, as a string, that the component that you want to revoke permissions for is associated with.
ComponentName	Specify the component name, as a string, that you want to revoke permissions for.
Market	Specify the market, as a string, associated with the component that you want to revoke permissions for.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to revoke permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

The My Profile Component Reference points to the USERMAINT_SELF component. Using the following example, "ALLPANLS" permission is revoked to the MY_PROFILE component reference.

```
&Portal.RevokePermissionForComponent("MAINTAIN_SECURITY", "USERMAINT_SELF", "GBL",⇒
 "ALLPANLS", "LOCAL_NODE");
```

See Also

Chapter 27, “PortalRegistry Classes,” GrantPermissionForComponent, page 1035

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, “Setting Up Permission Lists”

RevokePermissionForScript

Syntax

```
RevokePermissionForScript(RecordName, FieldName, EventName, FuncName,
PermListName, NodeName)
```

Description

Use the RevokePermissionForScript method to revoke the specified permission to the specified iScript. In addition, the specified permission is revoked for any component references that point to the specified iScript.

If you use the string "LOCAL_NODE" as *NodeName*, the system uses the node name currently defined as local.

All component entries in the portal registry are affected for all the portals. Not just the current portal.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RecordName	Specify the record name, as a string, of the record containing the iScript. All iScripts are contained on records whose names start with "WEBLIB".
FieldName	Specify the name of the field, as a string, containing the iScript.
EventName	Specify the name of the event, as a string, containing the iScript.
FuncName	Specify the name of the function, as a string, containing the iScript. Generally iScripts are contained in the FieldFormula event.
PermListName	Specify the name of the permission list, as a string, that you want to use.
NodeName	Specify the node of the component reference, as a string, that points to the component reference that you want to revoke permissions for. If you use the string "LOCAL_NODE", the system uses the node name currently defined as local.

Returns

A Boolean value: True if method completed successfully, False otherwise.

Example

```
&Portal.RevokePermissionForScript("WEBLIB_ALERT", "ALERTCOUNT", "FieldFormula",⇒
 "Connect_Alert", "ALLPGS", "Local_Node");
```

See Also

Chapter 27, “PortalRegistry Classes,” GrantPermissionForScript, page 1036

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, “Setting Up Permission Lists”

Save

Syntax

```
Save()
```

Description

The Save method saves changes that you made to the PortalRegistry or to a node template. It does *not* save any changes that you made to a folder, content reference, and so on.

Note. To save changes for a folder or content reference use the save method associated with that object.

Parameters

None.

Returns

A Boolean value: True if the PortalRegistry object is successfully saved, False otherwise.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
If Not(&MyPortal.Save()) Then
   /* do error checking */
End-if:
```

See Also

Chapter 27, "PortalRegistry Classes," Save, page 1060
folder method

Chapter 27, "PortalRegistry Classes," Save, page 1071
content reference method

PortalRegistry Class Properties

In this section, we discuss the PortalRegistry class properties. The properties are discussed in alphabetical order.

DefaultTemplate

Description

This property returns or sets the name of the default template for this PortalRegistry object as a string.

If you delete a template for a content reference, and none of the other content references on a page have a template, the default template specified with the Node is used. If there's no template for the Node, the template specified with this property is used.

To return a reference to the content reference that contains the template specified by this property, use the TemplateObject property.

Note. If you change the Default Template for a portal, the template won't take effect until you close all the existing browser windows with the session and open a new browser window.

This property takes only the first 30 characters of a value. If you specify a value longer than 30 characters, the remaining characters are ignored.

This property is read-write.

See Also

Chapter 27, "PortalRegistry Classes," TemplateObject, page 1044

Description

Description

This property returns or sets the description of this PortalRegistry object as a string

The length of this property is 256 characters.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Favorites

Description

This property returns a reference to the Favorites Collection for the current end-user.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Favorite Collection, page 1166

FolderNavObject

Description

This property specifies the content reference name to be used for the folder navigation object. When the folder navigation is tunred on, this content reference is displayed as the folder navigation homepage. This property takes a string value.

This property is read-write.

See Also

Chapter 27, "PortalRegistry Classes," IsFolderNavigation, page 1042

Homepage

Description

This property returns a reference to the Homepage for the current end-user.

This property is read-only.

IsFolderNavigation

Description

This property specifies if there is folder navigation. When the folder navigation is turned on, the user can see a folder homepage when clicked on the folder in the lefthand navigation menu. This property takes a boolean value: true if folder navigation is on, false otherwise.

This property is read-write.

See Also

Chapter 27, "PortalRegistry Classes," FolderNavObject, page 1042

Name

Description

This property returns the name of the PortalRegistry object as a string.

The length of this property is 30 characters.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

NodeTemplates

Description

This property returns a reference to a NodeTemplate Collection. This property can be used with a *closed* portal registry, that is, before you open it with the Open method.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," NodeTemplate Collection, page 1057

OwnerId

Description

This property returns or sets the owner ID of the PortalRegistry object as a string.

This property is read-write.

PageletCategories

Description

This property returns a reference to a PageletCategories Collection that contains all the PageletCategories for a portal.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," PageletCategory Collection, page 1142

Portals

Description

This property returns a reference to a Portal collection that contains references to all the portals in the database.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Portal Collection, page 1054

RootFolder

Description

This property returns the root folder object for this PortalRegistry object.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

TabDefinitions

Description

This property returns reference to a TabDefinitions Collection that contains all the TabDefinitions for a portal.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” TabDefinition Collection, page 1119

TemplateObject

Description

This property returns a reference to a content reference object that contains the template specified by the DefaultTemplate property. If no template is specified with DefaultTemplate, this property returns Null.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” DefaultTemplate, page 1041

PortalRegistry Collection

A PortalRegistry Collection is returned by the FindPortalRegistries session class method.

See Chapter 27, “PortalRegistry Classes,” FindPortalRegistries, page 1020.

PortalRegistry Collection Methods

In this section, we discuss the PortalRegistry collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first PortalRegistry object in the PortalRegistry collection.

Example

```
&MyRegistry = &MyCollection.First();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Item

Syntax

Item(*number*)

Description

The Item method returns the PortalRegistry object with the position in the PortalRegistry collection specified by *number*.

Parameters

Parameter	Description
number	Specify the position number in the collection of the PortalRegistry object that you want returned.

Returns

A PortalRegistry object if successful, NULL otherwise.

Example

```
For &I = 1 to &MyCollection.Count
   &MyRegistry = &MyCollection.Item(&I);
   /* Do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next PortalRegistry object in the PortalRegistry collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyRegistry = &MyCollection.Next();
```

PortalRegistry Collection Property

This section discusses the Count property.

Count

Description

This property returns the number of PortalRegistry objects in the PortalRegistry collection, as a number.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Node Class

The node indicates the URI of the content server. If the node is a PeopleSoft content server, it has a PeopleSoft server URI. If the node has external content, the URI indicates the external content.

Use the IsLocal or IsDefault to determine if a node from a RemoteNode collection is remote or local.

Node objects are instantiated from the following:

- From a Session object with the GetLocalNode method.
- From a Node Collection with the First, InsertItem, ItemByName, and Next methods.
- From a RemoteNode Collection with the First, InsertItem, ItemByName, and Next methods.

See Chapter 27, "PortalRegistry Classes," GetLocalNode, page 1020; Chapter 27, "PortalRegistry Classes," Node Collection, page 1049 and Chapter 27, "PortalRegistry Classes," RemoteNode Collection, page 1051.

Node Class Properties

In this section, we discuss the Node class properties. The properties are discussed in alphabetical order.

ActiveNode

Description

This property indicates whether the node has been specified as an active node. This property returns a Boolean value: True, the node is active, False otherwise.

This property is read-only.

AppsRelease

Description

This property returns the release of the PeopleSoft Applications hosted on this node as a string. This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," NodeType, page 1048

 Copyright © 1988-2007, Oracle. All rights reserved.

ContentURI

Description

This property returns the URI for the content webserver of this node as a string.

This property is read-only.

DefaultPortalName

Description

This property returns the name of the portal associated with the node as the default portal for the node.

This property is read-only.

Description

Description

This property returns the description for this node as a string.

The length of this property depends on your system database limit for LONG fields.

This property is read-only.

IsDefault

Description

This property indicates whether the node has been specified as the default local node. This property takes a Boolean value: True, this node has been specified as the default local node, False otherwise.

This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” NodeType, page 1048

IsLocal

Description

This property indicates whether this node has been specified as the local PeopleSoft node. This property takes a Boolean value: True, this node has been specified as a local node, False otherwise.

This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” NodeType, page 1048

Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property returns the name for this node object as a string. The name is a unique identifier for each node object.

Every node name must be unique in the database.

This property is read-only.

NodePassword

Description

This property returns the password for this node object as a string.

This property is read-only.

NodeType

Description

This property indicates what the node is used for. Nodes can either define PeopleSoft or external systems. PeopleSoft nodes have more capabilities than external nodes, therefore several other properties depend on this property.

Values are:

Value	Description
PIA	PIA (Peoplesoft 8.4 simple URL format)
EX	External Node
ICT	ICType (used for 8.1x content)

PIA is the default value on a new Node.

This property is read-only.

PortalURI

Description

This property returns the URI for the portal webserver of this node as a string. This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” NodeType, page 1048

 Copyright © 1988-2007, Oracle. All rights reserved.

ToolsRelease

Description

This property returns the release of PeopleTools running on this node as a string. This property is valid only when the NodeType property is set to PeopleSoft (PIA).

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," NodeType, page 1048

Node Collection

A node collection contains a set of references to all nodes defined in the database.

Node Collection Methods

In this section, we discuss the Node collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first Node object in the Node collection.

Parameters

None.

Returns

A Node object.

Example

```
&MyNode = &MyCollection.First();
```

ItemByName

Syntax

```
ItemByName(NodeName)
```

Description

The ItemByName method returns the Node object with the name specified by *NodeName*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
NodeName	Specify the name of an existing node in the Node collection. If you specify an invalid name, the object returned is NULL.

Returns

A Node object if successful, NULL otherwise.

Example

```
&MyOldNode = &MyPortal.Nodes.ItemByName("HRMS");
```

Next

Syntax

Next()

Description

The Next method returns the next Node object in the Node collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Node object.

Example

```
&My_Node = &MyCollection.Next();
```

Node Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Node objects in the Node Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

RemoteNode Collection

A remote node collection contains a set of references to all the nodes defined in the database, both local and remote.

Use the IsLocal property of the returned node object to determine if the node is remote or local.

RemoteNode Collection Methods

In this section, we discuss the RemoteNode collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first Node object in the RemoteNode collection.

Parameters

None.

Returns

A Node object.

Example

```
&MyNode = &MyCollection.First();
```

ItemByName

Syntax

ItemByName(*NodeName*)

Description

The ItemByName method returns the Node object with the name specified by *NodeName*.

Parameters

Parameter	Description
NodeName	Specify the name of an existing node in the RemoteNode collection. If you specify an invalid name, the object returned is NULL.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Node object if successful, NULL otherwise.

Example

```
&MyOldNode = &MyPortal.RemoteNodes.ItemByName("HRMS");
```

Next

Syntax

Next()

Description

The Next method returns the next Node object in the RemoteNode collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Node object.

Example

```
&My<i>Node</i> = &MyCollection.Next();
```

RemoteNode Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Node objects in the RemoteNode Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Portal Class

The portal class provides access to portals declared in the session.

 Copyright © 1988-2007, Oracle. All rights reserved.

A portal object is instantiated by the First, InsertItem, ItemByName, and Next Portal Collection methods

See Chapter 27, "PortalRegistry Classes," Portal Collection, page 1054.

Portal Class Method

In this section, we discuss the Portal class methods. The methods are discussed in alphabetical order.

Save

Syntax

```
Save()
```

Description

Use the Save method to save changes you made to the portal object.

Parameters

None.

Returns

A Boolean value: True if the portal object is successfully saved, False otherwise.

See Also

Chapter 27, "PortalRegistry Classes," Save, page 1040
PortalRegistry method

Portal Class Properties

In this section, we discuss the Portal class properties. The properties are discussed in alphabetical order.

HostNodeName

Description

This property sets or returns the name of the node hosting the portal as a string.

This property is read-write.

IsLocal

Description

This property indicates whether the portal is defined as local or remote. This property returns a Boolean value: True, the portal is defined as local, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Name

Description

This property returns the name of this portal as a string. This name exactly matches the portal name of the portal registry defined in the database.

This property is read-only.

Portal Collection

The portal collection contains a set of references to each portal defined in the database.

A portal collection is instantiated by the Portals PortalRegistry property.

See Chapter 27, "PortalRegistry Classes," Portals, page 1043.

Portal Collection Methods

In this section, we discuss the Portal collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first Portal object in the Portal collection.

Parameters

None.

Returns

A Portal object.

Example

```
&MyPortal = &MyCollection.First();
```

ItemByName

Syntax

ItemByName(*PortalName*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The ItemByName method returns the Portal object with the name *PortalName*.

Parameters

Parameter	Description
PortalName	Specify the name of an existing portal within the collection. If you specify an invalid name, the object returns NULL.

Returns

A Portal object if successful, NULL otherwise.

Example

```
&MyPortal = &MyPortal.Portals.ItemByName("HRMS");
```

Next

Syntax

Next()

Description

The Next method returns the next Portal object in the Portal collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Portal object.

Example

```
&MyPortal = &MyCollection.Next();
```

Portal Collection Properties

In this section, we discuss the Count property.

Count

Description

This property returns the number of Portal objects in the Portal Collection, as a number.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

NodeTemplate Class

The NodeTemplate class provides access to the default template assigned to a node in this portal.

A NodeTemplate object is instantiated by the First, InsertItem, ItemByName, and Next NodeTemplate Collection methods.

See Chapter 27, "PortalRegistry Classes," NodeTemplate Collection, page 1057.

NodeTemplate Class Properties

In this section, we discuss the NodeTemplate class properties. The properties are discussed in alphabetical order.

DefaultTemplate

Description

This property specifies the name of the template to be applied to the node.

This property is read-write.

Name

Description

This property specifies the name of node that the template is to be applied to.

This property is read-write.

TemplateObject

Description

This property returns a reference to the template object associated with this node as a content reference.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Content Reference Class, page 1070

 Copyright © 1988-2007, Oracle. All rights reserved.

NodeTemplate Collection

The NodeTemplate collection contains a set of references to each node template object in a portal.

A NodeTemplate collection is instantiated by the NodeTemplates PortalRegistry property.

See Chapter 27, "PortalRegistry Classes," NodeTemplates, page 1043.

NodeTemplate Collection Methods

In this section, we discuss the NodeTemplate collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*NodeName*)

Description

The DeleteItem method deletes the NodeTemplate object identified by *NodeName* from the NodeTemplate Collection.

This method is not executed automatically. It is executed only when the PortalRegistry is saved.

Parameters

Parameter	Description
NodeName	Specify the name of a NodeTemplate to delete.

Returns

A Boolean value: True if the NodeTemplate was deleted, False otherwise.

Example

```
If Not &MyNodeTemplates.DeleteItem("HRMS") Then
   /* Do error processing */
End-if
```

First

Syntax

First()

Description

The First method returns the first NodeTemplate object in the NodeTemplate collection.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A NodeTemplate object.

Example

```
&MyNodeTemplate = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*NodeName*)

Description

The InsertItem method inserts the NodeTemplate object identified by *NodeName* into the NodeTemplate Collection.

This method is not executed automatically. It is executed only when the PortalRegistry is saved.

Parameters

Parameter	Description
NodeName	Specify the name of the node template to insert.

Returns

A reference to the new NodeTemplate object if the method executed successfully, NULL otherwise.

Example

```
&NewNodeTemplate = &MyPortal.NodeTemplates.InsertItem("CRM");
```

ItemByName

Syntax

ItemByName(*NodeName*)

Description

The ItemByName method returns the NodeTemplate object with the name *NodeName*.

Parameters

Parameter	Description
NodeName	Specify the name of an existing NodeTemplate object in the collection. If you specify an invalid name, the object is NULL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A NodeTemplate object if successful, NULL otherwise.

Example

```
&MyNodeTemplate = &MyPortal.NodeTemplates.ItemByName("HRMS");
```

Next

Syntax

Next()

Description

The Next method returns the next NodeTemplate object in the NodeTemplate collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A NodeTemplate object.

Example

```
&MyNodeTemplate = &MyCollection.Next();
```

NodeTemplate Collection Properties

In this section, we discuss the Count property.

Count

Description

This property returns the number of NodeTemplate objects in the NodeTemplate Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Folder Class

Folder objects are instantiated from other classes as follows:

Copyright © 1988-2007, Oracle. All rights reserved.

- From a PortalRegistry object with the RootFolder property or the FindFolderByName method.
- From a Folder Collection with the First, ItemByName, or Next methods

See Chapter 27, "PortalRegistry Classes," RootFolder, page 1043; Chapter 27, "PortalRegistry Classes," FindFolderByName, page 1032 and Chapter 27, "PortalRegistry Classes," Folder Collection, page 1066.

See Chapter 27, "PortalRegistry Classes," Adding a Folder, page 1172.

Folder Class Method

In this section, we discuss the Folder class methods. The methods are discussed in alphabetical order.

Save

Syntax

Save()

Description

The Save method saves any changes you made to the folder, for example, a changed description or ValidFrom date.

Using this method also saves any changes you made to PermissionValue (both role-based and non role-based) and AttributeValue objects associated with this folder. Security permissions added are cascade upward until the root folder or first public folder. Removed security permissions are removed from all the parent folders until root or public folder in folder hierarchy.

Parameters

None.

Returns

A Boolean value: True if the Folder object is successfully saved, False otherwise.

Example

```
If Not(&MyFolder.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 27, "PortalRegistry Classes," Save, page 1040
PortalRegistry method

Chapter 27, "PortalRegistry Classes," Save, page 1071
content reference method

 Copyright © 1988-2007, Oracle. All rights reserved.

Folder Class Properties

In this section, we discuss the Folder class properties. The properties are discussed in alphabetical order.

Attributes

Description

This property returns an Attribute Collection containing the AttributeValue objects for this folder.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Attribute Collection, page 1089

Author

Description

This property returns the author (PeopleSoft user ID) for this folder as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the folder has access to the folder. This property takes a Boolean value. The default value for this property for a newly created object is True. This property is not cascaded.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this Folder.

This property is used when you access a particular Folder using FindFolderByName. If you've specified a valid Folder with this method, a Folder is always returned, whether you are authorized to view it or not. This is the only property you can view from an object that you are not authorized to.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this folder.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," FindFolderByName, page 1032

Copyright © 1988-2007, Oracle. All rights reserved.

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for all the child and parent objects (up to the root folder). To determine only the permissions of the object use the Permission property instead. To determine the role-based permissions of the object use the CascadedRolePermissions property instead.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," CascadedRolePermissions, page 1062, Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094 and Chapter 27, "PortalRegistry Classes," Permissions, page 1065.

CascadedRolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for all the child and parent objects (up to the root folder). To determine only the role-based permissions of the object use the RolePermission property instead. To determine the non role-based permissions of the object use the CascadedPermissions property.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedRolePermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," CascadedPermissions, page 1062 , Chapter 27, "PortalRegistry Classes," RolePermissions, page 1065 and Chapter 27, "PortalRegistry Classes," RolePermissionValue Collection, page 1097.

ContentRefs

Description

This property returns the ContentReference Collection for this folder.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Content Reference Collection, page 1084

 Copyright © 1988-2007, Oracle. All rights reserved.

CreationDate

Description

This property returns the creation date for this folder as a string.

This property is read-only.

Description

Description

This property returns or sets the description for this folder as a string.

The length of this property is 256 characters. This property is translatable.

This property is read-write.

Folders

Description

This property returns a reference to the Folder Collection for this folder.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Folder Collection, page 1066

IsMobile

Description

This property returns True if the folder is used with mobile applications, False otherwise.

This property is read-write.

IsVisible

Description

This property returns True if the Hide from Portal Navigation check box is *not* selected when the folder is created. If the folder is hidden from portal navigation, this property returns False.

Considerations Using IsVisible

If you do not specifically set this property on a new or copied folder, the IsVisible property is set as follows:

- If the ValidFrom date is less then or equal to that day's date, the IsVisible property is set to True.
- If there is no ValidTo date the IsVisible property is set to True.
- If the ValidTo date is greater then or equal to that day's date, the IsVisible property is set to True.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 27, “PortalRegistry Classes,” ValidFrom, page 1066 and Chapter 27, “PortalRegistry Classes,” ValidTo, page 1066.

Label

Description

This property returns or sets the label for this folder as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this folder as a sting. The name is a unique identifier for each folder.

Every folder name must be unique across the portal, not just in the parent folder.

This property is *not* translatable. However, the values for the Label and Description properties are translatable.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” Description, page 1063 and Chapter 27, “PortalRegistry Classes,” Label, page 1064.

OwnerId

Description

This property returns the name of the owner for this folder as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this folder as a string. This property is valid only if the folder is contained within another folder. If the folder using this property is the root folder, this property returns an empty string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Path

Description

The Path property returns a path to this folder, with each element of the path separated by a period. The path has the following syntax:

```
FolderLabel{Name}.[ChildFolderLabel{Name}.]. . .
```

This property is read-only.

Example

```
Departments{PORTA_ROOT_OBJECT}.HR{EastCoast}.AdministerWorkforce{Global}
```

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for this folder. To determine the permissions for all the parent objects (up to the root folder) use the CascadedPermissions property. To access the role-based permissions, use the RolePermissions property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” PermissionValue Collection, page 1094, Chapter 27, “PortalRegistry Classes,” CascadedPermissions, page 1062 and Chapter 27, “PortalRegistry Classes,” RolePermissionValue Collection, page 1097.

Product

Description

This property returns or sets the PeopleSoft product for this folder as a string.

The length of this property is 4 characters.

This property is read-write.

RolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for this folder. To determine the role-based permissions for all the parent objects (up to the root folder) use the CascadedRolePermissions property. To access the non role-based permissions, use the Permissions property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” CascadedRolePermissions, page 1062 , Chapter 27, “PortalRegistry Classes,” Permissions, page 1065 and Chapter 27, “PortalRegistry Classes,” RolePermissionValue Collection, page 1097.

Copyright © 1988-2007, Oracle. All rights reserved.

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned folders is based on the sequence number. Use this property to reorder folders.

If there are duplicates in the sequence number, the folders are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a folder is generally accessible, that is, if this property is set to True, any user can access the folder. This property is not cascaded.

This property takes a Boolean value.

The default value for this property for a newly created object is False.

This property is read-write.

ValidFrom

Description

This property returns or sets the date this folder is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this folder is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

Folder Collection

The Folder Collection provides access to a collection of folders in a Folder object.

The Folder Collection is instantiated from the Folders Folder Class property.

See Chapter 27, "PortalRegistry Classes," Folders, page 1063.

 Copyright © 1988-2007, Oracle. All rights reserved.

Folder Collection Methods

In this section, we discuss the Folder collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

```
DeleteItem(FolderName)
```

Description

The DeleteItem method deletes the folder object identified by *FolderName* from the database. If the folder contains other folders, all child folders and their contents are also deleted.

Warning! If you delete a folder, you delete *all* content in the folder. If you delete a folder that contains other folders, that is, a parent folder, all the child folders, and all the content references are deleted.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	Description
FolderName	Specify the name of a folder existing in the folder collection.

Returns

A Boolean value: True if the folder was deleted, False otherwise.

Example

```
If Not &MyFolderColl.DeleteItem("MYFOLDER") Then
   /* Folder not deleted. Do error checking */
End-If;
```

First

Syntax

```
First()
```

Description

The First method returns the first Folder object in the folder collection.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

Folder object.

Example

```
&MyFolder = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*FolderName*, *Label*)

Description

The InsertItem method inserts the folder object identified by *FolderName* from the Folder Collection. You must specify both a name and a label for all folders. This method returns a reference to the new folder object. You must specify a unique *FolderName*, or you receive an error.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	Description
FolderName	Specify the name of a folder existing in the folder collection. This parameter takes a string value.
Label	Specify a label for the new folder. This parameter takes a string value.

Returns

A reference to the new Folder object if the method executed successfully, null otherwise.

Example

```
&DeptHPFldr = &MyPortal.Folders.InsertItem("PORT0145", "HR Folder for Department⇒
 0145");
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the Folder object with the name *Name*.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of an existing folder within the folder collection. If you specify an invalid name, the object is NULL. You must specify a name, not a label.

Returns

A folder object if successful, NULL otherwise.

Example

```
&DeptFldr = &MyPortal.RootFolder.Folders.ItemByName(&Dept_Name);
```

Next

Syntax

Next()

Description

The Next method returns the next Folder object in the Folder collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A folder object.

Example

```
Local ApiObject &MySession, &Root, &Folders, &MyFolder;

&MySession = %Session;
&MyPortal = &MySession.GetPortalRegistry("ADMIN");

&Root = &MyPortal.GetRoot();
&Folders = &Root.Folders;

&MyFolder = &Folders.First();

For &I = 1 to &Folders.Count
   /* Do processing on folders */
   If &I <> &Folders.Count
      &MyFolder = &Folders.Next();
   End-If;
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Folder Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Folder objects in the Folder Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Content Reference Class

The content reference class provides access to some kind of content. The type of content depends on the content reference.

The content reference objects are instantiated from other classes:

- From a PortalRegistry object with the FindCRefByURL, FindCRefForURL, or FindCRefByName properties.
- From a Content Reference Collection (instantiated from a folder) with the First, ItemByName or Next methods

See Chapter 27, “PortalRegistry Classes,” FindCRefByURL, page 1029; Chapter 27, “PortalRegistry Classes,” FindCRefByName, page 1028; Chapter 27, “PortalRegistry Classes,” FindCRefForURL, page 1030 and Chapter 27, “PortalRegistry Classes,” Content Reference Collection, page 1084.

See Chapter 27, “PortalRegistry Classes,” Adding a Content Reference, page 1175.

Content Reference Class Methods

In this section, we discuss the Content Reference class methods. The methods are discussed in alphabetical order.

CreateLink

Syntax

CreateLink(*LinkName, Label*)

Description

Use the CreateLink method to create a link quickly to the same content reference executing the method. The link by defaults assumes the parent folder is the same as the content reference’s parent.

 Copyright © 1988-2007, Oracle. All rights reserved.

A CReflink object is returned if there is no error.

After you create a link you must use the Save method to save it to the database.

Parameters

Parameter	Description
LinkName	Specify the name of the link as a string. This parameter takes 30 characters. This is considered as ID of the link. It cannot start with number, and can not have special characters and spaces.
Label	Specify the label of the link as a string. This parameter takes 30 characters.

Returns

A reference to a CRefLink object.

Example

```
&CRef = &Portal.FindCRefByName("MyCRef");
&Link = &CRef.CreateLink("Link Object");
&Link.Save();
```

See Also

Chapter 27, “PortalRegistry Classes,” Link Class, page 1112 and Chapter 27, “PortalRegistry Classes,” Save, page 1102.

Save

Syntax

```
Save()
```

Description

The Save method saves any changes you made to the content reference, for example, a changed description. It also performs some validation.

Using this method also saves any changes you made to PermissionValue or AttributeValue objects associated with this content reference.

Parameters

None.

Returns

A Boolean value: True if the content reference and its associated objects saved successfully, False otherwise.

Example

```
If NOT(&MyCRef.Save()) Then
   /* save failed, do error processing */
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 27, “PortalRegistry Classes,” Save, page 1040
PortalRegistry method

Chapter 27, “PortalRegistry Classes,” Save, page 1060
Folder method

Content Reference Class Properties

In this section, we discuss the Content Reference class properties. The properties are discussed in alphabetical order.

AbsoluteContentURL

Description

This property returns the absolute content URL, that is, the content from the content servlet (psc).

This property is read-only.

Example

The following is an example absolute content URL:

```
http://serverx/psp/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

AbsolutePortalURL

Description

This property returns the absolute content reference portal URL.

This property is read-only.

Example

```
http://serverx/psp/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

AssignedPagelets

Description

This property returns an AssignedPagelet collection.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” AssignedPagelet Collection, page 1124

 Copyright © 1988-2007, Oracle. All rights reserved.

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this content reference object.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” Attribute Collection, page 1089

Author

Description

This property returns the author (PeopleSoft user ID) for this content reference object as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the content reference has access to the content reference. This property takes a Boolean value. The default value for this property is True.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this content reference.

This property is used when you access a particular content reference using FindCRefByURL, FindCRefForURL, or FindCRefByName. If you specified a valid content reference with either of these methods, a content reference is always returned, whether you are authorized to view it or not. This is the only property you can view from an object for which you are not authorized.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this content reference.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” FindCRefByURL, page 1029, Chapter 27, “PortalRegistry Classes,” FindCRefForURL, page 1030 and Chapter 27, “PortalRegistry Classes,” FindCRefByName, page 1028.

Copyright © 1988-2007, Oracle. All rights reserved.

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the Permissions property instead. To access the role-based permissions, use the CascadedRolePermissions property.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," CascadedRolePermissions, page 1074, Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094 and Chapter 27, "PortalRegistry Classes," Permissions, page 1077.

CascadedRolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the RolePermissions property instead. To determine non role-based permissions, use the CascadedPermissions property.

Note. You *cannot* add any RolePermissionValue objects to a collection returned by the CascadedRolePermissions property. You can add values only to the collection returned by the RolePermissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," CascadedRolePermissions, page 1074 , Chapter 27, "PortalRegistry Classes," RolePermissions, page 1079 and Chapter 27, "PortalRegistry Classes," RolePermissionValue Collection, page 1097.

ContentProvider

Description

This property returns or sets the name of the node for the content reference as a string.

This property takes the following values:

Value	Description
Node name	Specify the exact name of the node.
LOCAL_NODE	Specify the node for the content reference as the node defined as the local node.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

CreationDate

Description

This property returns the creation date for this content reference object as a string.

This property is read-only.

Data

Description

This property returns the data for this content reference. This property is valid only when the StorageType property is LOCL.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

Example

```
&MyData = &MyCRef.Data;
```

See Also

Chapter 27, "PortalRegistry Classes," StorageType, page 1079

Description

Description

This property returns or sets the description for this content reference object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

HtmlText

Description

This property returns the HTML text associated with this content reference as a string.

This property is read-only.

IsMobile

Description

This property returns True if this content reference is used with mobile applications, False otherwise.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

IsVisible

Description

This property returns True if the Hide from Portal Navigation check box is not selected when the content reference is created. If the content reference is hidden from portal navigation, this property returns False.

Considerations Using IsVisible

If you do not specifically set this property on a new or copied content reference object, the IsVisible property is set as follows:

- If the ValidFrom date is less then or equal to that day's date, the IsVisible property is set to True.
- If there is no ValidTo date the IsVisible property is set to True.
- If the ValidTo date is greater then or equal to that day's date, the IsVisible property is set to True.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," ValidFrom, page 1083 and Chapter 27, "PortalRegistry Classes," ValidTo, page 1083.

Label

Description

This property returns or sets the label for this content reference object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Links

Description

This property returns a reference to a Link collection. This collection contains all the links that are associated with this content reference.

This property is read-only. However, a link collection is updated in realtime when a new link is created on the object.

See Also

Chapter 27, "PortalRegistry Classes," Link Collection, page 1111

Name

Description

This property returns the name for this content reference object as a string. The name is a unique identifier for each content reference object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Every content reference name must be unique in the portal, not just in the parent folder.

This property is read-only.

OwnerId

Description

This property returns or sets the owner ID of the content reference object as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this content reference object as a string.

This property is read-only.

Example

The following example uses the ParentName to return a folder object for the content reference.

```
&Folder = &Portal.FindFolderByName(&CRef.ParentName);
```

Path

Description

The Path property returns a path to this content reference, with each element of the path separated by a period. The path has the following syntax:

```
ContentReferenceLabel{Name}.[ChildContentReferenceLabel{Name}.]. . .
```

This property is read-only.

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for this content reference. To determine the permissions for all the parent objects (up to the root folder) use the CascadedPermissions property.

If you want to find role-based permissions for the content reference, use the RolePermission property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” PermissionValue Collection, page 1094, Chapter 27, “PortalRegistry Classes,” CascadedPermissions, page 1074 and Chapter 27, “PortalRegistry Classes,” RolePermissions, page 1079.

Copyright © 1988-2007, Oracle. All rights reserved.

Product

Description

This property returns or sets the PeopleSoft product for this content reference object as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a content reference is generally accessible, that is, if it will always be included in the general content reference collection. This property takes a Boolean value.

The default value for this property is False.

This property is read-write.

QualifiedURL

Description

Note. This property is being kept for backward compatibility only. If your code uses this property, the value returned is actually from the AbsoluteContentURL property. New applications should use the AbsoluteContentURL property instead.

See Also

Chapter 27, "PortalRegistry Classes," AbsoluteContentURL, page 1072

RelativeURL

Description

This property returns the relative URL in the following format:

```
../../../Portal/Node/Content_Type/ContentID
```

For example, from the following URL:

```
http://mlee2038/servlets/psp/PS84/e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=⇒
view&Setid=110&Custid=99
```

The RelativeURL returns the following:

```
e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=view&Setid=110&Custid=99
```

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

RolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for this content reference. To determine the permissions for all the parent objects (up to the root folder) use the CascadedRolePermissions property.

If you want to find non role-based permissions for the content reference, use the Permission property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," CascadedRolePermissions, page 1074 , Chapter 27, "PortalRegistry Classes," Permissions, page 1077 and Chapter 27, "PortalRegistry Classes," RolePermissionValue Collection, page 1097.

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned content references is based on the sequence number. Use this property to reorder content references. This property takes a number value.

If there are duplicates in the sequence number, the content references are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

StorageType

Description

In general, content references contain information about where to get the content, and do not store the content. However, content references that are template or portal component types can have their content accessible directly from the content reference. In these cases, the Data property is valid and can be read or written, and the data is stored locally in the portal database.

StorageType	Meaning
LOCL	Local: Data property on content reference is valid.
RMTE	Remote: Data property is not valid.

When UsageType is a target this property must be set to RMTE and the URLType property should specify what format the Node and URL are in.

When UsageType is either a template or a portal component this property can be set to either LOCL or RMTE.

Note that when StorageType is LOCL, it specifies a static template or portal component. But, when StorageType is RMTE it can specify either:

Copyright © 1988-2007, Oracle. All rights reserved.

- a dynamically generated template or portal component
- a static template or portal component

In both cases, the Node and URL properties specifies how to get the template or portal component.

Only templates and homepage tab content references can have StorageType LOCL. The StorageType is always LOCL for homepage tabs. For templates LOCL means that corresponding URL is stored in database, not retrieved by URL.

The following table indicates the usage type, and what type of storage is available.

UsageType	RMTE	LOCL
FRMT (Frame template)	X	X
HTMT (HTML template)	X	X
HPGT (Homepage tab)		X
HPGC (Pagelet)	X	
TARG (Target)	X	

RMTE is the default value for a new content reference.

The length of this property is 4 characters.

This property is read-write.

See Also

Chapter 27, “PortalRegistry Classes,” Data, page 1075

Template

Description

This property returns or sets the name of the template used with this content reference as a string. You must specify the name of an existing template.

This property uses the name *not* the label of a content reference.

This property is used only when the UsageType property is specified as Target (TARG) and the TemplateType property is specified as HTML.

To return a reference to the content reference that contains the template specified by this property, use the TemplateObject property.

The length of this property is 30 characters.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 27, "PortalRegistry Classes," UsageType, page 1083, Chapter 27, "PortalRegistry Classes," TemplateType, page 1081 and Chapter 27, "PortalRegistry Classes," TemplateObject, page 1081.

TemplateObject

Description

This property returns a reference to a content reference object that contains the template specified by the Template property as a content reference. If no template is specified with Template, this property returns NULL.

This property is read-only.

See Also
Chapter 27, "PortalRegistry Classes," Template, page 1080

TemplateType

Description

This property indicates whether a template should be used to wrap the content. This property takes a string value.

Values are:

Value	Description
HTML	An HTML template should wrap the data (URL) in the content reference.
NONE	No template should be used. The text should not be wrapped.

HTML is the default value on a new content reference.

Use the NONE value if the URL should not appear in the portal. An example is when the content reference is a template itself.

If the homepage template for the user is un-retrievable, the system tries to use the default user's template first, before resorting to the portal default template.

When the content reference describes content, this property should be set to HTML.

This property is read-write.

See Also
Chapter 27, "PortalRegistry Classes," Using Content References, page 1013

URL

Description

This property returns or sets the URL for this content reference object as a string. The URL returns *exactly* as it appears in the database.

If you're setting a URL, you must use a unique URL.

Copyright © 1988-2007, Oracle. All rights reserved.

The absolute URL, that is, the URL from the node concatenated with this URL must be unique.

You receive an error if the URL your use is already registered.

To retrieve a qualified URL, that is, one that contains the node URI, use the AbsoluteContentURL property.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

The format of the value for this property depends on the setting of other properties.

See Also

Chapter 27, "PortalRegistry Classes," AbsoluteContentURL, page 1072 and Chapter 27, "PortalRegistry Classes," Using Content References, page 1013.

URLType

Description

This property indicates what kind of URL is used to retreive the content. This property takes a string value.

Note. This property is used only for content that has the StorageType property set as Remote.

Values are:

URLType Value	Meaning
UEXT	URL points to a non-PeopleSoft (external) URL
UMPG	URL points to a PeopleSoft mobile page
UPGE	URL points to a component.
UPHP	URL points to a homepage tab
UPTM	URL points to a template
USCR	URL points to an iScript
UGEN	URL points to a generic PeopleSoft URL
WRKL	URL points to a Worklist URL

UPGE is the default value on a new content reference.

This property is read-write.

See Also

 Copyright © 1988-2007, Oracle. All rights reserved.

Chapter 27, "PortalRegistry Classes," StorageType, page 1079

UsageType

Description

This property indicates what the content reference is used for. Several other properties depend on this property.

This property takes a string value.

The length of this property is 4 characters.

Value	Description
FRMT	Frame template: the content reference is a frame-based template.
HTMT	HTML template: the content reference is an HTML template.
HPGT	Homepage tab: the content reference is a homepage tab.
HPGC	Pagelet: the content reference is a pagelet used in the homepage.
TARG	Target: the content reference is the target content its template determines what else must be loaded and how it will look.
LINK	Link: the content reference is a link.

TARG is the default value on a new content reference.

This property is read-write.

See Also

Chapter 27, "PortalRegistry Classes," Using Content References, page 1013

ValidFrom

Description

This property returns or sets the date this content reference is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this content reference is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

Copyright © 1988-2007, Oracle. All rights reserved.

Content Reference Collection

The Content Reference Collection provides access to the collection of content references in a Folder object.

The Content Reference Collection is instantiated from the ContentRefs Folder property.

See Chapter 27, "PortalRegistry Classes," ContentRefs, page 1062.

Content Reference Collection Methods

In this section, we discuss the Content Reference collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*ContentReferenceName*)

Description

The DeleteItem method deletes the content reference object identified by *ContentReferenceName* from the content reference Collection.

If you delete a template for a content reference, and none of the other content references on a page have a template, the default template specified with the ContentProvider is used. If there's no template for the ContentProvider, the template for the PortalRegistry is used. However, if you delete the template for the content reference, the ContentProvider, and the PortalRegistry, you receive a runtime error.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	Description
ContentReferenceName	Specify the name of a content reference existing in the content reference collection.

Returns

A Boolean value: True if the content reference was deleted, False otherwise.

Example

```
If Not &MyCRef.DeleteItem("Test_CRef") Then
   /* can't delete test data. Do error processing */
End-if;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

First

Syntax

```
First()
```

Description

The First method returns the first content reference object in the content reference collection.

Parameters

None.

Returns

A content reference object.

Example

```
&MyCRef = &MyCollection.First();
```

InsertItem

Syntax

```
InsertItem(ContentReferenceName, ContentReferenceLabel, Node, URL)
```

Description

The InsertItem method inserts the content reference object identified by *ContentReferenceName* into the content reference Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database as soon as the method is executed.

Parameters

Parameter	Description
ContentReferenceName	Specify the name of a new content reference. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.
ContentReferenceLabel	Specify a description of the new content reference. This is the translated value. This parameter takes a string value.
Node	Specify a node. This parameter takes a string value. If you specify a fully qualified value for *URL*, you can specify a NULL (that is, two quotation marks with no space between them (""")).
URL	Specify a URL that contains the content. The format of this parameter depends on other properties, such as the type of content reference, where the data is stored, and so on.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to the new content reference object if the method executed successfully, NULL otherwise.

Example

The following example inserts an external content reference that is a template:

```
&URL = "t/" | &ITEMNAME;
&MyCRef = &CRefColl.InsertItem(&ITEMNAME, &ITEMLABEL, "", &URL);
```

The following example inserts a content reference where URLType is iScript (USCR):

```
&URL = "s/WEBLIB_PORTAL.FieldFormula.Portal_Trans_Dyn"
&MyCRef = &CRefColl.InsertItem(&ITEMNAME, &ITEMLABEL, "HRMS", &URL);
```

See Also

Chapter 27, "PortalRegistry Classes," Using Content References, page 1013

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the content reference object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing content reference within the content reference collection. If you specify an invalid name, the object is NULL.

Returns

A content reference object if successful, NULL otherwise.

Example

```
&MyCRef = &CRefColl.ItemByName("PORTAL_ADMIN);
```

Next

Syntax

Next()

Description

The Next method returns the next content reference object in the content reference collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

content reference object.

Example

```
&MyCRef = &MyCollection.Next();
```

Content Reference Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of content reference objects in the content reference Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

AttributeValue Class

The AttributeValue class provides access to attributes associated with either folders or content references.

AttributeValue objects are instantiated from an Attribute Collection Methods with the First, InsertItem, ItemByName, or Next methods.

See Chapter 27, “PortalRegistry Classes,” Attribute Collection Methods, page 1089.

See Chapter 27, “PortalRegistry Classes,” Using Attributes, page 1179.

AttributeValue Class Properties

In this section, we discuss the AttributeValue class properties. The properties are discussed in alphabetical order.

Label

Description

This property returns the label of the AttributeValue as a string. This property works with the Translatable property. If Translatable is set to True, the value of Label can be translated.

Copyright © 1988-2007, Oracle. All rights reserved.

The length of this property is 30 characters.

This property is read-write.

See Also
Chapter 27, "PortalRegistry Classes," Translatable, page 1088

Name

Description

This property returns the name of the AttributeValue as a string.

The length of this property is 30 characters.

This property is read-only.

Example

```
&AttrColl = &Folder.Attributes;
&Attr = &AttrColl.First();

   &Scroll = GetLevel0().GetRow(1).GetRowset(Scroll.PORTAL_FLDR_ATR);

   &I = 1;
   While All(&Attr)
      &Record = &Scroll.GetRow(&I).GetRecord(Record.PORTAL_FLDR_ATR);
      &Record.PORTAL_ATTR_NAM.Value = &Attr.Name;
      &Record.PORTAL_ATTR_VAL.Value = &Attr.Value;
      &Attr = &AttrColl.Next();
      /* need this check so we don't insert extra blank row */
      If All(&Attr) Then
         &Scroll.InsertRow(&I);
         &I = &I + 1;
      End-If;
End-While;
```

Translatable

Description

This property specifies if the AttributeValue is translatable. This property takes a Boolean value: True if the AttributeValue can be translated, False otherwise.

If this property is set to True, the value of the Label property can be translated.

Note. Regardless of the order in which attributes were entered, they are ordered according to their translatable property, that is, attributes that have this property set as True come first, followed by attributes that have this property set as False.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 27, “PortalRegistry Classes,” Label, page 1087

Value

Description

This property returns the value of the AttributeValue as a string.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

Example

To specify more than a single value for an AttributeValue, you can specify several values separated by a semicolon. For example:

```
&MyAtt.value = "401k;benefits;dependants;HR";
```

Attribute Collection

The Attribute Collection provides access to the collection of Attribute in a Folder or a content reference object.

An Attribute Collection is instantiated from other classes as follows:

- From a content reference object with the Attributes property.
- From a Folder object with the Attributes property.
- From a PageletCategory object with the Attributes property.
- From a Pagelet object with the Attributes property.

See Chapter 27, “PortalRegistry Classes,” Attributes, page 1073.

See Chapter 27, “PortalRegistry Classes,” Attributes, page 1061.

See Chapter 27, “PortalRegistry Classes,” Attributes, page 1138.

See Chapter 27, “PortalRegistry Classes,” Attributes, page 1146.

Attribute Collection Methods

In this section, we discuss the Attribute collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*AttributeValueName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The DeleteItem method deletes the AttributeValue object identified by *AttributeValueName* from the Attribute Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	Description
AttributeValueName	Specify the name of an AttributeValue existing in the Attribute Collection.

Returns

A Boolean value: True if the AttributeValue was deleted, False otherwise.

First

Syntax

```
First()
```

Description

The First method returns the first AttributeValue object in the Attribute Collection.

Parameters

None.

Returns

An AttributeValue object.

Example

```
&MyAttributeValue = &MyCollection.First();
```

InsertItem

Syntax

```
InsertItem(AttributeValueName)
```

Description

The InsertItem method inserts the AttributeValue object identified by *AttributeValueName* from the Attribute Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
AttributeValueName	Specify the name of an AttributeValue existing in the Attribute Collection.

Returns

A reference to the new AttributeValue object if the method executed successfully, NULL otherwise.

Example

```
For &I = 1 To &Rowset.ActiveRowCount
    If &CompName = "PORTAL_CREF_ADM" Then
       &Record = &Rowset.GetRow(&I).GetRecord(Record.PORTAL_CATR_DV);
    Else
       &Record = &Rowset.GetRow(&I).GetRecord(Record.PORTAL_FATR_DV);
    End-If;

    If &Record.PORTAL_ATTR_NAM.Value <> "" Then
       &Attr = &AttrColl.InsertItem(&Record.PORTAL_ATTR_NAM.Value);
       &Attr.Value = &Record.PORTAL_ATTR_VAL.Value;
    End-If;
End-For;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the AttributeValue object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing AttributeValue within the Attribute Collection. If you specify an invalid name, the object is NULL.

Returns

An AttributeValue object if successful, NULL otherwise.

Example

```
&Attr = &AttrColl.ItembyName("RELLINK");
```

Copyright © 1988-2007, Oracle. All rights reserved.

Next

Syntax

Next()

Description

The Next method returns the next AttributeValue object in the Attribute Collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

AttributeValue object.

Example

```
&MyAttributeValue = &MyCollection.Next();
```

Attribute Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AttributeValue objects in the Attribute Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

PermissionValue Class

The PermissionValue provides access to permission lists associated with folders, content references, PageletCategory objects, or Pagelets.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions or CascadedRolePermissions property. You can add values only to the collection returned by the Permissions or RolePermissions property.

PermissionValue objects are instantiated from the following:

 Copyright © 1988-2007, Oracle. All rights reserved.

- A PermissionValue Collection with the First, InsertItem, ItemByName, or Next methods.
- A RolePermissionValue Collection with the First, InsertItem, ItemByName, or Next methods.

See Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094.

See Chapter 27, "PortalRegistry Classes," Using Security, page 1001.

See Chapter 27, "PortalRegistry Classes," Setting Permissions Using the PermissionValue Object, page 1177.

PermissionValue Class Properties

In this section we discuss the Cascade and Name properties.

Cascade

Description

This property indicates whether the current permission should be granted to all child folders.

This property is valid only with folders, not with content references.

This property takes a Boolean value. The default value for a new PermissionValue object is False.

This property is read-write.

Name

Description

Specify the name of a permission list as the name of this object. You must specify a permission list that has already been created. This property takes a string value.

The length of this property is 30 characters.

This property is read-write.

PermType

Description

Specify the type of the permission. Values are:

Value	Description
P	Specify a non role-based permission.
R	Specify a role-based permission.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 27, "PortalRegistry Classes," RolePermissionValue Collection, page 1097 and Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094.

PermissionValue Collection

A PermissionValue collection is returned by the following:

- CascadedPermissions and Permissions folder property
- CascadedPermissions and Permissions content reference property
- CascadedPermissions and Permissions PageletCategory object property
- CascadedPermissions and Permissions Pagelet property

What is contained in the PermissionValue collection depends on the property that created it.

See Chapter 27, "PortalRegistry Classes," CascadedPermissions, page 1062 and Chapter 27, "PortalRegistry Classes," Permissions, page 1065.

See Chapter 27, "PortalRegistry Classes," CascadedPermissions, page 1074 and Chapter 27, "PortalRegistry Classes," Permissions, page 1077.

See Chapter 27, "PortalRegistry Classes," CascadedPermissions, page 1139 and Chapter 27, "PortalRegistry Classes," Permissions, page 1141.

See Chapter 27, "PortalRegistry Classes," CascadedPermissions, page 1147 and Chapter 27, "PortalRegistry Classes," Permissions, page 1149.

See Chapter 27, "PortalRegistry Classes," Using Security, page 1001.

PermissionValue Collection Methods

In this section, we discuss the PermissionValue collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*PermissionValueName*)

Description

The DeleteItem method deletes the PermissionValue object identified by *PermissionValueName* from the PermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PermissionValueName	Specify the name of a PermissionValue existing in the PermissionValue collection.

Returns

A Boolean value: True if the PermissionValue was deleted, False otherwise.

Example

```
If Not &MyPValColl.DeleteItem("ALLPNLS") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the first PermissionValue object in the PermissionValue collection.

Parameters

None.

Returns

A PermissionValue object.

Example

```
&MyPermissionValue = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*PermissionValueName*)

Description

The InsertItem method inserts the PermissionValue object identified by *PermissionValueName* into the PermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can only add values to the collection returned by the Permissions property.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PermissionValueName	Specify the name of an existing permission list.

Returns

A reference to the new PermissionValue object if the method executed successfully, NULL otherwise.

Example

```
&MyPermV = &MyPermVColl.InsertItem("ALLPNLS");

If Not &MyPermV Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the PermissionValue object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing PermissionValue within the PermissionValue collection. If you specify an invalid name, the object is NULL.

Returns

A PermissionValue object if successful, NULL otherwise.

Example

```
&MyPVal = &MyPValColl.ItemByName("CUSTOMER");
```

Next

Syntax

Next()

Description

The Next method returns the next PermissionValue object in the PermissionValue collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

PermissionValue object.

Example

```
&MyPermissionValue = &MyCollection.Next();
```

PermissionValue Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of PermissionValue objects in the PermissionValue Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

RolePermissionValue Collection

A RolePermissionValue collection is returned by the following:

- CascadedRolePermissions and RolePermissions folder property

  See Chapter 27, “PortalRegistry Classes,” CascadedRolePermissions, page 1062.

  See Chapter 27, “PortalRegistry Classes,” RolePermissions, page 1065.
- CascadedRolePermissions and RolePermissions content reference property

  See Chapter 27, “PortalRegistry Classes,” CascadedRolePermissions, page 1074.

  See Chapter 27, “PortalRegistry Classes,” RolePermissions, page 1079.
- CascadedRolePermissions and RolePermissions content reference link property

  See Chapter 27, “PortalRegistry Classes,” CascadedRolePermissions, page 1104.

  See Chapter 27, “PortalRegistry Classes,” RolePermissions, page 1108.

What is contained in the RolePermissionValue collection depends on the property that created it.

Copyright © 1988-2007, Oracle. All rights reserved.

RolePermissionValue Collection Methods

In this section, we discuss the RolePermissionValue collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*RolePermissionValueName*)

Description

The DeleteItem method deletes the RolePermissionValue object identified by *RolePermissionValueName* from the RolePermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	Description
RolePermissionValueName	Specify the name of a RolePermissionValue existing in the RolePermissionValue collection.

Returns

A Boolean value: True if the RolePermissionValue was deleted, False otherwise.

Example

```
If Not &MyPValColl.DeleteItem("ALLPNLS") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the first RolePermissionValue object in the RolePermissionValue collection.

Parameters

None.

Returns

A RolePermissionValue object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyRolePermissionValue = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*RolePermissionValueName*)

Description

The InsertItem method inserts the RolePermissionValue object identified by *RolePermissionValueName* into the RolePermissionValue Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Note. You *cannot* add any RolePermissionValue objects to a collection returned by the CascadedRolePermissions property. You can only add values to the collection returned by the RolePermissions property.

Parameters

Parameter	Description
RolePermissionValueName	Specify the name of an existing role permission list.

Returns

A reference to the new RolePermissionValue object if the method executed successfully, NULL otherwise.

Example

```
&MyPermV = &MyPermVColl.InsertItem("ALLPNLS");

If Not &MyPermV Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the RolePermissionValue object with the name *Name*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of an existing RolePermissionValue within the RolePermissionValue collection. If you specify an invalid name, the object is NULL.

Returns

A RolePermissionValue object if successful, NULL otherwise.

Example

```
&MyPVal = &MyPValColl.ItemByName("CUSTOMER");
```

Next

Syntax

Next()

Description

The Next method returns the next RolePermissionValue object in the RolePermissionValue collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

RolePermissionValue object.

Example

```
&MyRolePermissionValue = &MyCollection.Next();
```

RolePermissionValue Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of RolePermissionValue objects in the RolePermissionValue Collection, as a number.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Content Reference Links

A content reference link is a content reference object that delegates most of its properties to another content reference. It has its own separate properties, as well as a reference to the linked cref.

A content reference link is instantiated from the following:

- CreateLink content reference method
- CreateContentRefLink PortalRegistry method

Note. The CreateLink method creates a new link. To instantiate an existing link use the FindLinkByName method.

See Also

Chapter 27, “PortalRegistry Classes,” CreateContentRefLink, page 1025

Chapter 27, “PortalRegistry Classes,” FindCRefLinkByName, page 1031

Chapter 27, “PortalRegistry Classes,” CreateLink, page 1070

Content Reference Link Methods

In this section, we discuss the content reference link methods. The methods are discussed in alphabetical order.

Delete

Syntax

Delete()

Description

Use the delete method to delete the link from the database.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The link is not marked for deletion, then actually deleted later. The link is deleted from the database as soon as the method is executed.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the content reference link and its associated objects are deleted successfully, False otherwise.

Save

Syntax

```
Save()
```

Description

The Save method saves any changes you made to the content reference link, for example, a changed description or ValidFrom date.

After you create a link using either the CreateLink or CreateContentRefLink methods you must use the Save method to save the link to the database.

Using this method also saves any changes you made to PermissionValue or AttributeValue objects associated with this content reference link.

Parameters

None.

Returns

A Boolean value: True if the content reference link and its associated objects saved successfully, False otherwise. If False is returned, a message is also written to the PSMessage collection.

Example

```
If NOT(&MyCRefLink.Save()) Then
   /* save failed, do error processing */
End-If;
```

Content Reference Link Properties

In this section, we discuss the content reference link properties. The properties are discussed in alphabetical order.

AbsoluteContentURL

Description

This property returns the absolute content URL, that is, the content from the content servlet (psc). The content is displayed without any portal template.

This property is read-only.

Example

The following is an example absolute content URL:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
http://serverx/psc/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

AbsolutePortalURL

Description

This property returns the absolute content reference link portal URL. This URL also contains the content in the portal template.

This property is read-only.

Example

```
http://serverx/psp/PS84/EMPLOYEEPORTAL/CRM/c/SFA.CUSTOMERINFO.GBL?page=CUST_⇒
DATA1&&Action=U&emplid=00001
```

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this content reference link object.

This property is read-write.

See Also

Chapter 27, “PortalRegistry Classes,” Attribute Collection, page 1089

Author

Description

This property returns the author (PeopleSoft user ID) for this content reference link as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the content reference link has access to the content reference link. This property takes a Boolean value. The default value for this property is true.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this content reference link.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is used when you access a particular content reference link using FindCRefLinkByURL. If you specified a valid content reference link, a content reference link is always returned, whether you are authorized to view it or not. This is the only property you can view from an object that you are not authorized tom and the content reference is empty.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this content reference link.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," FindCRefLinkByName, page 1031

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the Permissions property instead.

Note. You cannot add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094 and Chapter 27, "PortalRegistry Classes," Permissions, page 1077.

CascadedRolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the RolePermissions property instead. To determine the non role-based permissions of the object use the Permissions property instead.

Note. You cannot add any PermissionValue objects to a collection returned by the CascadedRolePermissions property. You can add values only to the collection returned by the RolePermissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," CascadedRolePermissions, page 1104 , Chapter 27, "PortalRegistry Classes," RolePermissions, page 1108 and Chapter 27, "PortalRegistry Classes," RolePermissionValue Collection, page 1097.

 Copyright © 1988-2007, Oracle. All rights reserved.

ContentProvider

Description

This property returns the name of the node for the content reference link as a string.

This property is read-only.

CreationDate

Description

This property returns the creation date for this content reference link object as a string.

This property is read-only.

Data

Description

This property returns the data for this content reference link. This property is valid only when the StorageType property is LOCL.

The length of this property depends on your system database limit for LONG fields.

This property is read-write.

Example

```
&MyData = &MyCReflink.Data;
```

See Also

Chapter 27, “PortalRegistry Classes,” StorageType, page 1079

Description

Description

This property returns or sets the description for this content reference link object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

IsMobile

Description

This property returns True if this content reference link is used with mobile applications, False otherwise.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

IsVisible

Description

This property returns True if the Hide from Portal Navigation check box is not selected when the content reference link is created. If the content reference link is hidden from portal navigation, this property returns False. Also, if the date associated with the link is not valid (is not within the valid to and valid from dates,) this property returns False.

Considerations Using IsVisible

If you do not specifically set this property on a new or copied content reference link, the IsVisible property is set as follows:

- If the ValidFrom date is less then or equal to that day's date, the IsVisible property is set to True.
- If there is no ValidTo date the IsVisible property is set to True.
- If the ValidTo date is greater then or equal to that day's date, the IsVisible property is set to True.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," ValidFrom, page 1083 and Chapter 27, "PortalRegistry Classes," ValidTo, page 1083.

Label

Description

This property returns or sets the label for this content reference link object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this content reference link object as a string. The name is a unique identifier for each content reference link object. The name cannot start with a number, and it cannot have spaces and special characters.

Every content reference link name must be unique in the portal, not just in the parent folder.

This property is read-write.

OwnerId

Description

This property returns or sets the owner ID of the content reference link object as a string.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

ParentName

Description

This property returns the parent folder name for this content reference link object as a string.

You can use this property to move the content reference link to another folder.

This property is read-write.

Example

The following example uses the ParentName to return a folder object for the content reference link.

```
&Folder = &Portal.FindFolderByName(&CReflink.ParentName);
```

Path

Description

The Path property returns a path to this content reference link, with each element of the path separated by a period. The path has the following syntax:

```
Root{PORTAL_ROOT_OBJECT}.LinkFolder {<Id of links folder>}.CrefLink {<Id of the⇒
 cref link> }
```

This property is read-only.

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the non role-based permissions for this content reference link. To access the role-based permissions, use the RolePermissions property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” PermissionValue Collection, page 1094, Chapter 27, “PortalRegistry Classes,” CascadedPermissions, page 1074 and Chapter 27, “PortalRegistry Classes,” RolePermissions, page 1108.

Product

Description

This property returns or sets the PeopleSoft product for this content reference link object as a string.

The length of this property is 4 characters.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

PublicAccess

Description

This property specifies whether the link is public or private. This property takes a Boolean value: true if the content reference link is public and there are no permission lists associated with it. If it is false, the link uses permission list control.

For links, this values is same as that of the linked content reference. If the content reference is public, the link is public and can be viewed by anyone logged on to the system.

This property is read-only.

RelativeURL

Description

This property returns the relative URL in the following format:

```
../../../Portal/Node/Content_Type/ContentID
```

For example, from the following URL:

```
http://mlee2038/servlets/psp/PS84/e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=⇒
view&Setid=110&Custid=99
```

The RelativeURL returns the following:

```
e_procurement/fdm/c/E_PRO.CheckOut.GBL?page=view&Setid=110&Custid=99
```

This property is read-only.

RolePermissions

Description

This property returns a RolePermissionValue Collection. This collection contains the value of the role-based permissions for this content reference link. To access the non role-based permissions, use the Permissions property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” CascadedRolePermissions, page 1104 , Chapter 27, “PortalRegistry Classes,” Permissions, page 1107 and Chapter 27, “PortalRegistry Classes,” RolePermissionValue Collection, page 1097.

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned content reference links is based on the sequence number. Use this property to reorder content reference links. This property takes a number value.

If there are duplicates in the sequence number, the content reference links are returned alphabetically.

The length of this property is 4 characters.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Template

Description

This property returns or sets the name of the template used with this content reference link as a string. You must specify the name of an existing template.

This property uses the name, not the label, of a content reference link.

This property is used only when the UsageType property is specified as Target (TARG) and the TemplateType property is specified as HTML.

To return a reference to the content reference link that contains the template specified by this property, use the TemplateObject property.

The length of this property is 30 characters.

This property is read-write.

See Also

Chapter 27, "PortalRegistry Classes," UsageType, page 1083, Chapter 27, "PortalRegistry Classes," TemplateType, page 1081 and Chapter 27, "PortalRegistry Classes," TemplateObject, page 1081.

TemplateObject

Description

This property returns a reference to a content reference link object that contains the template specified by the Template property as a content reference link. If no template is specified with Template, this property returns a null value.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Template, page 1080

TemplateType

Description

This property indicates whether a template should be used to wrap the content. This property takes a string value.

Values are:

Value	Description
HTML	An HTML template should wrap the data (URL) in the content reference.
NONE	No template should be used. The text should not be wrapped.

HTML is the default value on a new content reference.

Copyright © 1988-2007, Oracle. All rights reserved.

Use the NONE value if the URL should not appear in the portal. An example is when the content reference link is a template itself.

If the homepage template for the user is un-retrievable, the system tries to use the default user's template first, before resorting to the portal default template.

When the content reference link describes content, this property should be set to HTML.

This property is read-write.

See Also
Chapter 27, "PortalRegistry Classes," Using Content References, page 1013

URL

Description

This property returns the URL for this content reference link object as a string. The content reference link derives the URL from the content reference it is linked to. The URL of the link is same as that of the linked content reference.

A query string value is added on the content reference URL to make the link unique. If the URL of the linked content reference changes, this URL is changed.

This property is read-only.

See Also
Chapter 27, "PortalRegistry Classes," AbsoluteContentURL, page 1072 and Chapter 27, "PortalRegistry Classes," Using Content References, page 1013.

URLType

Description

This property returns what kind of URL is used to retreive the content.

This property is read-only.

See Also
Chapter 27, "PortalRegistry Classes," StorageType, page 1079

UsageType

Description

This property returns what the content reference link is used for.

The value for this property is same as that of the content reference when the link is vieved as a content reference.

The usage type of the link is LINK in the database.

A content reference link can be created only for content references that have a type of TARG.

This property is read-only.

See Also
Chapter 27, "PortalRegistry Classes," Using Content References, page 1013

 Copyright © 1988-2007, Oracle. All rights reserved.

ValidFrom

Description

This property returns or sets the date this content reference link is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this content reference link is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

Link Collection

A Links collection is returned by the Links content reference property.

See Also

Chapter 27, "PortalRegistry Classes," Links, page 1076

Link Collection Methods

In this section, we discuss the link collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first link object in the link collection.

Parameters

None.

Returns

A link object.

Copyright © 1988-2007, Oracle. All rights reserved.

Next

Syntax

```
Next()
```

Description

The Next method returns the next link object in the collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A link object.

Link Collection Property

In this section, we discuss the link collection property Count.

Count

Description

This property returns the number of link objects in the link collection, as a number.

This property is read-only.

Link Class

A Link object is returned by the First and Next Link collection methods.

See Also

Chapter 27, "PortalRegistry Classes," Link Collection Methods, page 1111

Link Properties

In this section, we discuss the link properties. The properties are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

LinksObjectName

Description

This property returns the object name of the link as a string.

This property is read-only.

LinksObjectType

Description

This property returns the object type as a string. Valid value is CONTENTREF.

This property is read-only.

LinksPortalName

Description

This property returns the name of the portal from which the link originated.

This property is read-only.

TabDefinition Class

The TabDefinition class provides access to the homepage tab defined for the portal.

TabDefinition objects are instantiated from a TabDefinition Collection with the First, InsertItem, ItemByName, or Next methods.

See Chapter 27, "PortalRegistry Classes," TabDefinition Collection, page 1119.

TabDefinition Class Methods

In this section, we discuss the Save method.

Save

Syntax

```
Save()
```

Description

The Save method saves any changes you made to the TabDefinition, for example, a changed description. It also performs some validation.

Using this method also saves any changes you made to DynamicCategories and AssignedPagelets objects associated with this TabDefinition.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A Boolean value: True if the TabDefinition and its associated object saved successfully, False otherwise.

Example

```
If NOT(&MyTabDefn.Save()) Then
   /* save failed, do error processing */
End-If;
```

See Also

Chapter 27, "PortalRegistry Classes," Save, page 1040
PortalRegistry method

Chapter 27, "PortalRegistry Classes," Save, page 1060
Folder method

Chapter 27, "PortalRegistry Classes," Save, page 1071
ContentReference method

TabDefinition Class Properties

In this section, we discuss the TabDefinition class properties. The properties are discussed in alphabetical order.

AssignedPagelets

Description

This property returns a reference to an AssignedPagelet Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," AssignedPagelet Collection, page 1124

AvailableCategories

Description

This property returns a reference to an AvailableCategory Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," AvailableCategory Collection, page 1128

 Copyright © 1988-2007, Oracle. All rights reserved.

AvailablePagelets

Description

This property returns a reference to an AvailablePagelet Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," AvailablePagelet Collection, page 1132

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this TabDefinition object.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Attribute Collection, page 1089

Author

Description

This property returns the author (PeopleSoft user ID) for this TabDefinition object as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the TabDefinition has access to the TabDefinition. This property takes a Boolean value. The default value for this property is True.

This property is read-write.

ColumnLayout

Description

This property indicates whether the tab layout is for two or three columns. This property takes a string value.

The values for this property are:

Value	Description
2	Two-column table
3	Three-column table

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

CreationDate

Description

This property returns the creation date for this TabDefinition object as a string.

This property is read-only.

Description

Description

This property returns or sets the description for this TabDefinition object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

DynamicCategories

Description

This property returns a reference to a DynamicCategory Collection for this TabDefinition object.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” DynamicCategory Collection, page 1134

HelpID

Description

This property returns the help system identifier for this tab. It enables the help system to provide tab-specific help. This property takes a string value.

This property is read-write.

HtmlText

Description

This property returns the HTML text associated with this tab definition as a string.

This property is read-only.

IsHideActionBar

Description

This property hides all pagelet action bar images for all pagelets used on this tab. This property overrides any properties for the pagelets used on this tab. This property takes a Boolean value: True, hide images, False, display images

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

IsLayoutLocked

Description

This property indicates whether the user can change the tab's default number of columns. This property takes a Boolean value: True, the user can change the tab columns, False, the columns can't be changed by the user.

This property is read-write.

IsRenameable

Description

This property indicates whether the tab's label can be changed by the user. This property takes a Boolean value: True, the user can change the label, False, the user can't.

This property is read-write.

Label

Description

This property returns or sets the label for this TabDefinition object as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

LayoutBehavior

Description

This property indicates whether the user can change when and how the tab displays. This property takes a string value.

The values are:

Value	Description
4OPT	The user can choose to add this tab to their homepage. It doesn't show up by default.
3DEF	The user sees this tab the first time they log in, however, they can remove this tab if they'd like.
2REQ	The user can't remove this tab from their homepage. However, they are allowed to change its sequence or order on the page.
1FIX	The user can't remove this tab or change its position on the page.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property returns the name for this TabDefinition object as a string. The name is a unique identifier for each TabDefinition object.

Every TabDefinition name must be unique in the portal, not just in the parent folder.

This property is read-only.

OwnerId

Description

This property returns or sets the owner ID of the TabDefinition object as a string.

This property is read-write.

Product

Description

This property returns or sets the PeopleSoft product for this TabDefinition object as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a TabDefinition is generally accessible, that is, if it is always included in the general TabDefinition collection. This property takes a Boolean value.

The default value for this property is False.

This property is read-write.

QualifiedURL

Description

This property returns an absolute URL. If the TabDefinition has a ContentProvider associated with it, the URI from the ContentProvider is concatenated with the URL from the TabDefinition. If there is no ContentProvider, this property returns the text in the URL property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” URL, page 1081

 Copyright © 1988-2007, Oracle. All rights reserved.

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned TabDefinitions is based on the sequence number. Use this property to reorder TabDefinitions. This property takes a number value.

If there are duplicates in the sequence number, the TabDefinitions are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

StyleSheet

Description

This property defines the style sheet to use for this tab. This property takes a string value. This property is 30 characters long.

If no style sheet is specified, the default style sheet PSSTYLEDEF is used.

This property is read-write.

ValidFrom

Description

This property returns or sets the date this TabDefinition is valid from as a string.

This property is read-write.

ValidTo

Description

This property returns or sets the date this TabDefinition is valid until as a string.

This property is read-write.

Note. The PortalRegistry API never uses the ValidTo and ValidFrom fields to determine what to return in a collection. You must check for these values in your application.

TabDefinition Collection

A TabDefinition collection is the collection of all TabDefinitions for a portal.

A TabDefinition collection is returned by the TabDefinitions PortalRegistry property.

See Chapter 27, “PortalRegistry Classes,” TabDefinitions, page 1044.

Copyright © 1988-2007, Oracle. All rights reserved.

TabDefinition Collection Methods

This section describes the TabDefinition collection methods in alphabetical order.

DeleteItem

Syntax

DeleteItem(*TabDefinitionName*)

Description

The DeleteItem method deletes the TabDefinition object identified by *TabDefinitionName* from the TabDefinition Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	Description
TabDefinitionName	Specify the name of a TabDefinition existing in the TabDefinition collection.

Returns

A Boolean value: True if the TabDefinition was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("TabDefnTest") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the first TabDefinition object in the TabDefinition collection.

Parameters

None.

Returns

A TabDefinition object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyTabDefinition = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*TabDefinitionName*)

Description

The InsertItem method inserts the TabDefinition object identified by *TabDefinitionName* into the TabDefinition Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	Description
TabDefinitionName	Specify the name of an existing permission list.

Returns

A reference to the new TabDefinition object if the method executed successfully, NULL otherwise.

Example

```
&MyTabDef = &MyColl.InsertItem("Empl_Homepage");

If Not &MyTabDef Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the TabDefinition object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing TabDefinition within the TabDefinition collection. If you specify an invalid name, the object is NULL.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A TabDefinition object if successful, NULL otherwise.

Example

```
&MyTebDefn = &MyColl.ItemByName("Empl_Homepage");
```

Next

Syntax

```
Next()
```

Description

The Next method returns the next TabDefinition object in the TabDefinition collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

TabDefinition object.

Example

```
&MyTabDefinition = &MyCollection.Next();
```

TabDefinition Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of TabDefinition objects in the TabDefinition Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

AssignedPagelet Class

An AssignedPagelet is a pagelet object that has been assigned to a particular tab. This pagelet also contains the layout data that is specific to the pagelet and tab.

AssignedPagelet objects are instantiated from an AssignedPagelet Collection with the First, InsertItem, ItemByName, or Next methods.

See Chapter 27, "PortalRegistry Classes," AssignedPagelet Collection, page 1124.

AssignedPagelet Class Properties

In this section, we discuss the AssignedPagelet class properties. The properties are discussed in alphabetical order.

Column

Description

This property returns the column in which this pagelet is displayed, as a number.

This property is read-write.

LayoutBehavior

Description

This property indicates whether the user can change when and how the pagelet displays. This property takes a string value.

The values are:

Value	Description
4OPT	The user can choose to add this pagelet to their tab. It doesn't show up by default. The user can remove and reposition this pagelet on the tab.
3DEF	The user sees this pagelet the first time they log in, however, they can remove or reposition this pagelet if they'd like.
2REQ	The user can't remove this pagelet from the tab. However, they are allowed to reposition the pagelet on the tab.
1FIX	The user can't remove this pagelet or change its position on the tab.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

PageletName

Description

This property returns the name for this Pagelet object as a string. The name is a unique identifier for each Pagelet object.

This property is read-only.

Row

Description

This property returns the row in which this pagelet is displayed, as a number.

This property is read-write.

AssignedPagelet Collection

An AssignedPagelet collection is returned by the AssignedPagelet TabDefinition method.

It contains a collection of all the pagelets explicitly assigned to the tab.

See Chapter 27, "PortalRegistry Classes," AssignedPagelets, page 1114.

See Chapter 27, "PortalRegistry Classes," AssignedPagelet Class, page 1123.

AssignedPagelet Collection Methods

In this section, we discuss the AssignedPagelet collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*PageletName*)

Description

The DeleteItem method deletes the AssignedPagelet object identified by *PageletName* from the AssignedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PageletName	Specify the name of an AssignedPagelet existing in the AssignedPagelet collection.

Returns

A Boolean value: True if the AssignedPagelet was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Weather_411") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the first AssignedPagelet object in the AssignedPagelet collection.

Parameters

None.

Returns

An AssignedPagelet object.

Example

```
&MyAssignedPagelet = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*PageletName*, *Column*, *Row*, *LayoutBehavior*)

Description

The InsertItem method inserts the AssignedPagelet object identified by the parameters into the AssignedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PageletName	Specify the name of the pagelet that you want to insert.
Column	Specify the column that you want this pagelet to display in, as a number.
Row	Specify the row that you want this pagelet to display in, as a number. You can specify a zero if you don't know which row.
LayoutBehavior	Specify whether the user can change when and how the pagelet displays. This property takes a string value. The values are:

Value	Description
4OPT	The user can choose to add this pagelet to their tab. It doesn't show up by default. The user can remove and reposition this pagelet on the tab.
3DEF	The user sees this pagelet the first time they log in, however, they can remove or reposition this pagelet if they'd like.
2REQ	The user can't remove this pagelet from the tab. However, they are allowed to reposition the pagelet on the tab.
1FIX	The user can't remove this pagelet or change its position on the tab.

Returns

A reference to the new AssignedPagelet object if the method executed successfully, NULL otherwise.

Example

```
&MyPagelet = &MyColl.InsertItem("Weather_411");

If Not &MyPagelet Then
   /* do error processing */
End-If;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the AssignedPagelet object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing AssignedPagelet within the AssignedPagelet collection. If you specify an invalid name, the object is NULL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

An AssignedPagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &MyColl.ItemByName("Dictionary");
```

Next

Syntax

Next()

Description

The Next method returns the next AssignedPagelet object in the AssignedPagelet collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

An AssignedPagelet object.

Example

```
&MyAssignedPagelet = &MyCollection.Next();
```

AssignedPagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AssignedPagelet objects in the AssignedPagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Copyright © 1988-2007, Oracle. All rights reserved.

AvailableCategory Class

An AvailableCategory class is composed of a category name and an AvailablePagelet Collection, that is, a collection of all the available pagelets associated with that category.

AvailableCategory objects are instantiated from an AvailableCategory Collection with the First, ItemByName, or Next methods.

See Chapter 27, “PortalRegistry Classes,” AvailableCategory Collection, page 1128.

AvailableCategory Class Properties

In this section, we discuss the AvailableCategory class properties. The properties are discussed in alphabetical order.

AvailablePagelets

Description

This property returns a reference to an AvailablePagelet Collection associated with the specified CategoryName.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” AvailablePagelet Collection, page 1132

CategoryName

Description

Use this property to return the name of a category to be associated with the TabDefinition, as a string.

This property is read-only.

AvailableCategory Collection

An AvailableCategory is composed of a category name and an AvailablePagelet Collection, that is, a collection of all the available pagelets associated with that category.

An AvailableCategory collection is returned by the AvailableCategories TabDefinition property.

See Chapter 27, “PortalRegistry Classes,” AvailableCategory Collection, page 1128.

 Copyright © 1988-2007, Oracle. All rights reserved.

AvailableCategory Collection Methods

In this section, we discuss the AvailableCategory collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first AvailableCategory object in the AvailableCategory collection.

Parameters

None.

Returns

An AvailableCategory object.

Example

```
&MyAvailableCategory = &MyCollection.First();
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the AvailableCategory object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing AvailableCategory within the AvailableCategory collection. If you specify an invalid name, the object is NULL.

Returns

An AvailableCategory object if successful, NULL otherwise.

Example

```
&MyCat = &MyColl.ItemByName("Dictionary");
```

Copyright © 1988-2007, Oracle. All rights reserved.

Next

Syntax

Next()

Description

The Next method returns the next AvailableCategory object in the AvailableCategory collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

AvailableCategory object.

Example

```
&MyAvailableCategory = &MyCollection.Next();
```

AvailableCategory Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AvailableCategory objects in the AvailableCategory Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

AvailablePagelet Class

An AvailablePagelet is a pagelet object can be assigned to a particular tab. This includes all pagelets from the dynamic categories, and assigned pagelets. Each pagelet also contains the layout data that is specific to that pagelet.

AvailablePagelet objects are instantiated from an AvailablePagelet Collection with the First, ItemByName, or Next methods.

See Chapter 27, "PortalRegistry Classes," AvailablePagelet Collection, page 1132.

 Copyright © 1988-2007, Oracle. All rights reserved.

AvailablePagelet Class Properties

In this section, we discuss the AvailablePagelet class properties. The properties are discussed in alphabetical order.

CategoryLabel

Description

This property turns the label of the category to which the pagelet is assigned, as a string.

This property is read-only.

CategoryName

Description

This property returns the name of the category to which the pagelet is assigned, as a string.

This property is read-only.

Column

Description

This property returns the column in which this pagelet is displayed, as a number.

This property is read-only.

LayoutBehavior

Description

This property indicates whether the user can change when and how the pagelet displays. This property takes a string value.

The values are:

Value	Description
4OPT	The user can choose to add this pagelet to their tab. It doesn't show up by default. The user can remove and reposition this pagelet on the tab.
3DEF	The user sees this pagelet the first time they log in, however, they can remove or reposition this pagelet if they'd like.
2REQ	The user can't remove this pagelet from the tab. However, they are allowed to reposition the pagelet on the tab.
1FIX	The user can't remove this pagelet or change its position on the tab.

This property is read only.

Copyright © 1988-2007, Oracle. All rights reserved.

PageletLabel

Description

This property returns the label for this property as a string.

This property is read-only.

PageletName

Description

This property returns the name for this Pagelet object as a string. The name is a unique identifier for each Pagelet object.

This property is read-only.

Row

Description

This property returns the row in which this pagelet is displayed, as a number.

This property is read-only.

AvailablePagelet Collection

An AvailablePagelet collection is returned by the AvailablePagelet TabDefinition method.

It contains a collection of all the pagelets that could be assigned to the tab. It's an aggregation of the pagelets in the AssignedPagelets collection and all the pagelets in each of the categories in the DynamicCategories collection. Pagelet category sequence numbers, followed by pagelet sequence numbers, then name, sort this collection.

If you have two pagelets assigned with the same name (that is, an assigned pagelet and a dynamic category pagelet), the assigned pagelet takes precedence over the dynamic category one, and is the only one listed.

See Chapter 27, "PortalRegistry Classes," AvailablePagelets, page 1115.

See Chapter 27, "PortalRegistry Classes," AvailablePagelet Class, page 1130.

AvailablePagelet Collection Methods

In this section, we discuss the AvailablePagelet collection methods. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

First

Syntax

First()

Description

The First method returns the first AvailablePagelet object in the AvailablePagelet collection.

Parameters

None.

Returns

An AvailablePagelet object.

Example

```
&MyAvailablePagelet = &MyCollection.First();
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the AvailablePagelet object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing AvailablePagelet within the AvailablePagelet collection. If you specify an invalid name, the object is NULL.

Returns

An AvailablePagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &MyColl.ItemByName("DICTIONARY");
```

Next

Syntax

Next()

Description

The Next method returns the next AvailablePagelet object in the AvailablePagelet collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

AvailablePagelet object.

Example

```
&MyAvailablePagelet = &MyCollection.Next();
```

AvailablePagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of AvailablePagelet objects in the AvailablePagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

DynamicCategory Collection

A DynamicCategory collection contains a collection of PageletCategory names associated with a TabDefinition. The collection is initially ordered by category name.

When a PageletCategory is dynamic, it is immediately available to an end-user.

A DynamicCategory Collection is instantiated by the DynamicCategories property of a TabDefinition.

See Chapter 27, “PortalRegistry Classes,” DynamicCategories, page 1116.

DynamicCategory Collection Methods

In this section, we discuss the DynamicCategory collection methods. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

DeleteItem

Syntax

DeleteItem(*Name*)

Description

The DeleteItem method deletes the PageletCategory object identified by *Name* from the DynamicCategory Collection. This does *not* delete the PageletCategory from the database, just from the DynamicCategory collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Note. If you delete a DynamicCategory, tabs with the attribute PORTAL_HPTAB_DYN lists must be searched for and updated (if necessary) to reflect the deleted category.

Parameters

Parameter	Description
Name	Specify the name of an existing category in the DynamicCategory collection.

Returns

A Boolean value: True if the PageletCategory was deleted from the DynamicCategory collection, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Dictionaries") Then
   /* do error processing */
End-If;
```

First

Syntax

First()

Description

The First method returns the name of the first PageletCategory in the DynamicCategory collection.

Parameters

None.

Returns

A string.

Example

```
&MyDynamicCategory = &MyCollection.First();
```

Copyright © 1988-2007, Oracle. All rights reserved.

InsertItem

Syntax

InsertItem(*Name*)

Description

The InsertItem method inserts the PageletCategory object identified by *Name* into the DynamicCategory Collection. This does *not* insert the PageletCategory into the database, just into the DynamicCategory collection. After a PageletCategory is marked as dynamic, it is immediately available to the end-user.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	Description
Name	Specify the name of the DynamicCategory to insert.

Returns

A string containing the name of the new DynamicCategory object if the method executed successfully, Null otherwise.

Example

```
&MyCat = &MyColl.InsertItem("User_Info");
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the name of the PageletCategory object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing DynamicCategory within the DynamicCategory collection. If you specify an invalid name, this method returns a Null string.

Returns

A string.

Example

```
&MyPagelet = &MyColl.ItemByName("Dictionary");
```

Next

 Copyright © 1988-2007, Oracle. All rights reserved.

Syntax

Next()

Description

The Next method returns the name of the next PageletCategory object in the DynamicCategory collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A string.

Example

```
&MyDynamicCategory = &MyCollection.Next();
```

DynamicCategory Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of PageletCategory names in the DynamicCategory Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

PageletCategory Class

A PageletCategory is a group of related pagelets, such as Weather, News, Reference, and so on. Each PageletCategory has a collection of pagelets associated with it.

A PageletCategory is instantiated from a PageletCategory Collection with the First, InsertItem, ItemByName, or Next methods.

All PageletCategory objects that are dynamic are contained in the DynamicCategory collection. The methods from this collection return the names of the PageletCategory objects only, not references to the objects themselves.

See Chapter 27, "PortalRegistry Classes," PageletCategory Collection, page 1142.

Copyright © 1988-2007, Oracle. All rights reserved.

PageletCategory Class Method

In this section, we discuss the Save method.

Save

Syntax

```
Save()
```

Description

The Save method saves any changes you made to the PageletCategory, for example, a changed description or sequence number.

Parameters

None.

Returns

A Boolean value: True if the PageletCategory object is successfully saved, False otherwise.

Example

```
If Not(&MyPageletCategory.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 27, “PortalRegistry Classes,” Save, page 1040
PortalRegistry method

Chapter 27, “PortalRegistry Classes,” Save, page 1071
Folder method

PageletCategory Class Properties

In this section, we discuss the PageletCategory class properties. The properties are discussed in alphabetical order.

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this PageletCategory object.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 27, "PortalRegistry Classes," Attribute Collection, page 1089

Author

Description

This property returns the author (PeopleSoft user ID) for this PageletCategory as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the PageletCategory has access to the PageletCategory. This property takes a Boolean value. The default value for this property for a newly created object is True. This property is not cascaded.

This property is read-write.

Authorized

Description

This property specifies whether the user is authorized to view this PageletCategory.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this PageletCategory.

This property is read-only.

CascadedPermissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for all the child and parent objects (up to the root PageletCategory). To determine only the permissions of the object use the Permission property instead.

Note. The PORTAL_PAGELETS folder is the parent folder for all PageletCategories. Its security is set to public. PeopleSoft does not recommend cascading any permissions from this folder object. Cascade permissions only from the pagelet category (folder). You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also
Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094 and Chapter 27, "PortalRegistry Classes," Permissions, page 1065.

Copyright © 1988-2007, Oracle. All rights reserved.

CreationDate

Description

This property returns the creation date for this PageletCategory as a string.

This property is read-only.

Description

Description

This property returns or sets the description for this PageletCategory as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

Label

Description

This property returns or sets the label for this PageletCategory as a string.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this PageletCategory as a sting. The name is a unique identifier for each PageletCategory.

Every PageletCategory name must be unique in across the portal, not just in the parent PageletCategory.

This property is *not* translatable. However, the values for the Label and Desrciption properties are translatable.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” Label, page 1140

OwnerId

Description

This property returns or sets the owner ID of the PageletCategory object as a string.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

Pagelets

Description

This property returns a reference to a Pagelet Collection for this PageletCategory.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Pagelet Collection, page 1151

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for this PageletCategory.

Note. The PORTAL_PAGELETS folder is the parent folder for all PageletCategories. Its security is set to public. PeopleSoft does not recommend cascading any permissions from this folder object. Cascade permissions only from the pagelet category (folder).

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094

Product

Description

This property returns or sets the PeopleSoft product for this PageletCategory as a string.

The length of this property is 4 characters.

This property is read-write.

PublicAccess

Description

This property indicates whether a PageletCategory is generally accessible, that is, if this property is set to True, any user can access the PageletCategory. This property is not cascaded.

This property takes a Boolean value.

The default value for this property for a newly created object is False.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned PageletCategory objects is based on the sequence number. Use this property to reorder PageletCategory objects.

If there are duplicates in the sequence number, the PageletCategory objects are returned in alphabetical order.

The length of this property is 4 characters.

This property is read-write.

PageletCategory Collection

The PageletCategory Collection provides access to a collection of PageletCategory objects in a PageletCategory object.

The PageletCategory Collection is instantiated from the PageletCategories PortalRegistry property.

See Chapter 27, "PortalRegistry Classes," PageletCategories, page 1043.

PageletCategory Collection Methods

In this section, we discuss the PageletCategory collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*PageletCategoryName*)

Description

The DeleteItem method deletes the PageletCategory object identified by *PageletCategoryName* from the database.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	Description
PageletCategoryName	Specify the name of a PageletCategory existing in the PageletCategory collection.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the PageletCategory was deleted, False otherwise.

Example

```
If Not &MyPageletCategoryColl.DeleteItem("MYPAGELETCATEGORY") Then
   /* PageletCategory not deleted. Do error checking */
End-If;
```

First

Syntax

First()

Description

The First method returns the first PageletCategory object in the PageletCategory collection.

Parameters

None.

Returns

PageletCategory object.

Example

```
&MyPageletCategory = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*PageletCategoryName*, *Label*)

Description

The InsertItem method inserts the PageletCategory object identified by *PageletCategoryName* from the PageletCategory Collection. You must specify both a name and a label for each PageletCategory. This method returns a reference to the new PageletCategory object. You must specify a unique *PageletCategoryName*, or you receive an error.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PageletCategoryName	Specify the name of a PageletCategory existing in the PageletCategory collection. This parameter takes a string value.
Label	Specify a label for the new PageletCategory. This parameter takes a string value.

Returns

A reference to the new PageletCategory object if the method executed successfully, False otherwise.

ItemByName

Syntax

`ItemByName``(`*`Name`*`)`

Description

The ItemByName method returns the PageletCategory object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing PageletCategory within the PageletCategory collection. If you specify an invalid name, the object is NULL. You must specify a name, not a label.

Returns

A PageletCategory object if successful, NULL otherwise.

Next

Syntax

`Next``()`

Description

The Next method returns the next PageletCategory object in the PageletCategory collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A PageletCategory object.

 Copyright © 1988-2007, Oracle. All rights reserved.

PageletCategory Collection Property

In this section, we discuss the PageletCategory collection properties. The properties are discussed in alphabetical order.

Count

Description

This property returns the number of PageletCategory objects in the PageletCategory Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Pagelet Class

A pagelet represents a URL that renders a section of content for a homepage. Pagelets are assigned to specific categories.

A pagelet is instantiated from the First, InsertItem, ItemByName, and Next Pagelet Collection methods.

See Chapter 27, "PortalRegistry Classes," Pagelet Collection, page 1151.

Pagelet Class Method

In this section, we discuss the Save method.

Save

Syntax

```
Save()
```

Description

The Save method saves any changes you made to the Pagelet, for example, a changed description or sequence number.

Parameters

None.

Returns

A Boolean value: True if the Pagelet object is successfully saved, False otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
If Not(&MyPagelet.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 27, "PortalRegistry Classes," Save, page 1040
PortalRegistry method

Chapter 27, "PortalRegistry Classes," Save, page 1071
ContentReference method

Chapter 27, "PortalRegistry Classes," Save, page 1138
PageletCategory method

Pagelet Class Properties

In this section, we discuss the Pagelet class properties. The properties are discussed in alphabetical order.

Attributes

Description

This property returns an Attribute Collection containing the AttributeValues for this pagelet object.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," Attribute Collection, page 1089

Author

Description

This property returns the author (PeopleSoft user ID) for this pagelet object as a string.

This property is read-only.

AuthorAccess

Description

This property specifies whether the author of the pagelet has access to the pagelet. This property takes a Boolean value. The default value for this property is True.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

Authorized

Description

This property specifies whether the user is authorized to view this pagelet.

The initial value of this property depends on the other permission properties (PublicAccess and AuthorAccess) and the permission list values in the PermissionValue object associated with this pagelet.

This property is read-only.

CascadedPermissions

Description

This property returns a PermissionValue collection. This collection contains the value of the permissions for all the parent objects (up to the root folder). To determine only the permissions of the object use the Permission property instead.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," PermissionValue Collection, page 1094 and Chapter 27, "PortalRegistry Classes," Permissions, page 1077.

ContentProvider

Description

This property returns or sets the name of the node for the pagelet as a string.

If no node was specified when the pagelet was created, this property returns Null.

This property is read-write.

CreationDate

Description

This property returns the creation date for this pagelet object as a string.

This property is read-only.

DefaultColumn

Description

This property returns or sets the default column this pagelet displays in. This property takes a number value.

The default value for this property is 1.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Description

This property returns or sets the description for this pagelet object as a string.

The length of this property is 256 characters.

This property is translatable.

This property is read-write.

EditPageContentProvider

Description

This is the node part of the URL to the page that enables a user to personalize a page. The default value is an empty string.

This property is read-write.

EditPageQueryString

Description

This is the query string part of the URL to the page that enables a user to personalize a page. The default value for this property is an empty string.

This property is read-write.

HelpID

Description

This property returns the help system identifier for this pagelet. It enables the help system to provide pagelet specific help. This property takes a string value.

This property is read-write.

IsHideMinimize

Description

This property specifies if the minimize image icon displays. This property takes a Boolean value: True, display the icon, False, hide the icon. The default value is True.

This property is read-write.

Label

Description

This property returns or sets the label for this pagelet object as a string.

The length of this property is 30 characters.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is translatable.

This property is read-write.

Name

Description

This property returns the name for this pagelet object as a string. The name is a unique identifier for each pagelet object.

Every pagelet name must be unique in the portal, not just in the parent folder.

This property is not translatable. However, the values for the Label and Description properties are.

This property is read-only.

OwnerId

Description

This property returns or sets the owner ID of the pagelet object as a string.

This property is read-write.

ParentName

Description

This property returns the parent folder name for this pagelet object as a string.

This property is read-write.

Permissions

Description

This property returns a PermissionValue Collection. This collection contains the value of the permissions for this pagelet. To determine the permissions for all the parent objects (up to the root folder) use the CascadedPermissions property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” PermissionValue Collection, page 1094 and Chapter 27, “PortalRegistry Classes,” CascadedPermissions, page 1074.

Product

Description

This property returns or sets the PeopleSoft product for this pagelet object as a string.

The length of this property is 4 characters.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

PublicAccess

Description

This property indicates whether a pagelet is generally accessible, that is, if it will always be included in the general pagelet collection. This property takes a Boolean value.

The default value for this property is False.

This property is read-write.

QualifiedURL

Description

This property returns an absolute URL. If the pagelet has a node associated with it, the URI from the node is concatenated with the URL from the pagelet. If there is no ContentProvider, this property returns the text in the URL property.

This property is read-only.

See Also

Chapter 27, “PortalRegistry Classes,” URL, page 1081

SequenceNumber

Description

The sequence number is used when returning a collection. The default order of the returned pagelets is based on the sequence number. Use this property to reorder pagelets. This property takes a number value.

If there are duplicates in the sequence number, the pagelets are returned alphabetically.

The length of this property is 4 characters.

This property is read-write.

URL

Description

This property returns or sets the URL for this pagelet object as a string. The URL returns *exactly* as it appears in the database.

If you’re setting a URL, you must use a unique URL.

The absolute URL, that is, the URL from the ContentProvider concatenated with this URL must be unique.

You receive an error if the URL you use is already registered.

To retrieve a qualified URL, that is, one that contains the node URI, use the AbsoluteContentURL property.

The length of this property depends on your system database limit for LONG fields.

The format of the value for this property depends on the setting of other properties.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

URLType

Description

This property indicates what kind of PeopleCode definition is used to retrieve the content of the pagelet. This property takes a string value.

Values are:

URLType Value	Meaning
UEXT	URL points to a non-PeopleSoft (external) URL
UPGE	URL points to a component.
UGEN	URL points to a generic PeopleSoft URL
USCR	URL points to an iScript

UPGE is the default value on a new pagelet.

This property is read-write.

Pagelet Collection

The Pagelet Collection provides access to the collection of pagelets in a category.

The Pagelet Collection is instantiated from the Pagelets PageletCategory property.

See Chapter 27, "PortalRegistry Classes," Pagelets, page 1141.

Pagelet Collection Methods

In this section, we discuss the Pagelet collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

```
DeleteItem(PageletName)
```

Description

The DeleteItem method deletes the pagelet object identified by *PageletName* from the pagelet Collection.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database as soon as the method is executed.

Parameters

Parameter	Description
PageletName	Specify the name of a pagelet existing in the pagelet collection.

Returns

A Boolean value: True if the pagelet was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Test_Pagelet") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

Description

The First method returns the first pagelet object in the pagelet collection.

Parameters

None.

Returns

A reference to a pagelet object if successful, NULL otherwise.

Example

```
&MyCRef = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*PageletName*, *Label*, *NodeName*, *URL*)

Description

The InsertItem method inserts the pagelet object identified by *PageletName* into the pagelet Collection.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted into the database *as soon as* the method is executed.

Parameters

Parameter	Description
PageletName	Specify the name of a new pagelet. This parameter takes a string value. If you specify a name that already exists in the collection, you receive an error.
Label	Specify the label for the new pagelet. This parameter takes a string value.
NodeName	Specify a node. This parameter takes a string value. If you specify a fully qualified URL, you can specify a Null (that is, two quotation marks with no space between them (""")).
URL	Specify a URL that contains the content. The format of this parameter depends on other properties, such as the type of pagelet, where the data is stored, and so on.

Returns

A reference to the new pagelet object if the method executed successfully, NULL otherwise.

See Also

Chapter 27, "PortalRegistry Classes," Using Content References, page 1013

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the pagelet object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing pagelet within the pagelet collection. If you specify an invalid name, the object will be NULL.

Returns

A reference to a pagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &CRefColl.ItemByName("PORTAL_ADMIN);
```

Copyright © 1988-2007, Oracle. All rights reserved.

Next

Syntax

Next()

Description

The Next method returns the next pagelet object in the pagelet collection. You can only use this method after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a pagelet object if successful, NULL otherwise.

Example

```
&MyPagelet = &MyCollection.Next();
```

Pagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of pagelet objects in the pagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

UserHomepage Class

The UserHomepage represents a homepage for a user. It contains all the tabs and pagelets the user has selected for their homepage.

A UserHomepage is instantiated from the Homepage PortalRegistry property.

See Chapter 27, "PortalRegistry Classes," Homepage, page 1042.

 Copyright © 1988-2007, Oracle. All rights reserved.

UserHomepage Class Method

In this section, we discuss the Save method.

Save

Syntax

Save()

Description

The Save method saves any changes you made to the UserHomepage, for example, a changed greeting.

Using this method also saves any changes you've made to any UserTab or SelectedPagelet objects associated with this UserHomepage.

Parameters

None.

Returns

A Boolean value: True if the UserHomepage object is successfully saved, False otherwise.

Example

```
If Not(&MyHP.Save()) Then
   /* do error checking */
End-If;
```

See Also

Chapter 27, "PortalRegistry Classes," Save, page 1040
PortalRegistry class method

Chapter 27, "PortalRegistry Classes," Save, page 1060
Folder class method

UserHomepage Class Properties

In this section, we discuss the UserHomepage properties. The properties are discussed in alphabetical order.

Greeting

Description

This property is a user-definable message that appears on their homepage when they login, such as "My Homepage" or "Welcome." This property takes a text string. The length of this character is 254.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

UserId

Description

This property returns the UserId for this UserHomepage as a string.

This property is read-only.

UserTab

Description

This property returns a reference to a UserTab collection.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," UserTab Collection, page 1158

UserTab Class

A UserTab is a personalized tab for a specific user. This user can potentially change the pagelets that appear on the tab, the name of the tab, and the order of the tab. All these changes can be allowed or denied on the TabDefinitions Class. The UserTab is used in place of the TabDefinition whenever a user has personalized a tab.

A UserTab object is instantiated from the First, ItemByName, InsertItem, or Next UserTab Collection methods.

See Chapter 27, "PortalRegistry Classes," UserTab Collection, page 1158.

UserTab Class Properties

In this section, we discuss the UserTab class properties. The properties are discussed in alphabetical order.

ColumnLayout

Description

This property specifies how many columns the UserTab contains. This property takes a numeric value. The values are:

Value	Description
2	Two-column layout
3	Three-column layout. This is the default value.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

Label

Description

This property returns or sets the label for this UserTab object as a string. This value is initially set to the Label of the TabDefinition.

The length of this property is 30 characters.

This property is translatable.

This property is read-write.

QualifiedURL

Description

This property returns an absolute URL as a string. This URL includes the URI from the TabDefinition's node and the URL that points to the actual content of the tab.

This property is read-only.

SelectedPagelets

Description

This property returns a SelectedPagelets Collection, containing the pagelets that have been selected by the user to appear on this tab.

This property is read-only.

See Also

Chapter 27, "PortalRegistry Classes," SelectedPagelet Collection, page 1162

SequenceNumber

Description

This property specifies the order that this tab should appear as on the homepage. This property takes a number value. This value is initially set by the referenced TabDefinition.

This property is read-write.

TabName

Description

This property returns the tab name as a sting.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

UserTab Collection

The UserTab collection is the collection of personalized tabs for a user.

The UserTab Collection is instantiated from the UserTabs UserHomepage property.

See Chapter 27, "PortalRegistry Classes," UserTab, page 1156.

UserTab Collection Methods

In this section, we discuss the UserTab collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*UserTabName*)

Description

The DeleteItem method deletes the UserTab object identified by *UserTabName* from the UserTab Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	Description
UserTabName	Specify the name of a UserTab existing in the UserTab collection.

Returns

A Boolean value: True if the UserTab was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Test_UserTab") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

Description

The First method returns the first UserTab object in the UserTab collection.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A UserTab object.

Example

```
&MyUserTab = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*UserTabName*)

Description

The InsertItem method inserts the UserTab object identified by *UserTabName* into the UserTab Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

The new user tab must be unique within the portal.

Parameters

Parameter	Description
UserTabName	Specify the name of a new UserTab. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.

Returns

A reference to the new UserTab object if the method executed successfully, NULL otherwise.

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the UserTab object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing UserTab within the UserTab collection. If you specify an invalid name, the object is NULL.

Returns

A UserTab object if successful, NULL otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyUserTab = &MyColl.ItemByName("PORTAL_ADMIN");
```

Next

Syntax

Next()

Description

The Next method returns the next UserTab object in the UserTab collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A UserTab object.

Example

```
&MyUserTab = &MyCollection.Next();
```

UserTab Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of UserTab objects in the UserTab Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SelectedPagelet Class

The SelectedPagelet class contains data on pagelets selected by the user that display on this tab.

A SelectedPagelet object is instantiated by the First, InsertItem, ItemByName, and Next SelectedPagelet Collection methods.

See Chapter 27, "PortalRegistry Classes," SelectedPagelet Collection, page 1162.

 Copyright © 1988-2007, Oracle. All rights reserved.

SelectedPagelet Class Properties

In this section, we discuss the SelectedPagelet class properties. The properties are discussed in alphabetical order.

CategoryName

Description

This property returns the category of the pagelet as a string.

This property is read-only.

Column

Description

This property returns the number of the column the pagelet displays in, as a number.

The values are:

Value	Description
1	First column
2	Second column
3	Third column

This property is read-write.

IsMinimized

Description

This property specifies whether the pagelet is displayed minimized. This property takes a Boolean value: True, the pagelet is minimized, False otherwise.

This property is read-write.

PageletName

Description

This property returns the name of the pagelet as a string.

This property is read-only.

Row

Description

This property specifies which row the pagelet is displayed in, as a number.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

SelectedPagelet Collection

The SelectedPagelet collection contains a collection of all the selected pagelets for a tab.

A SelectedPagelet collection is instantiated by the SelectedPagelets UserTab property.

See Chapter 27, "PortalRegistry Classes," SelectedPagelets, page 1157.

SelectedPagelet Collection Methods

In this section, we discuss the SelectedPagelet collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*SelectedPageletName*)

Description

The DeleteItem method deletes the SelectedPagelet object identified by *SelectedPageletName* from the SelectedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	Description
SelectedPageletName	Specify the name of a SelectedPagelet existing in the SelectedPagelet collection.

Returns

A Boolean value: True if the SelectedPagelet was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("Test_SelectedPagelet") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The First method returns the first SelectedPagelet object in the SelectedPagelet collection.

Parameters

None.

Returns

A SelectedPagelet object.

Example

```
&MySelectedPagelet = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*SelectedPageletName*, *Column*, *Row*)

Description

The InsertItem method inserts the SelectedPagelet object identified by *SelectedPageletName* into the SelectedPagelet Collection.

This method is not executed automatically. It is executed only when the parent object is saved.

Parameters

Parameter	Description
SelectedPageletName	Specify the name of a new SelectedPagelet. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.
Column	Specify the column number for this pagelet.
Row	Specify the row number for this pagelet.

Returns

A reference to the new SelectedPagelet object if the method executed successfully, NULL otherwise.

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the SelectedPagelet object with the name *Name*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of an existing SelectedPagelet within the SelectedPagelet collection. If you specify an invalid name, the object is NULL.

Returns

A SelectedPagelet object if successful, NULL otherwise.

Example

```
&MySelectedPagelet = &MyColl.ItemByName("PORTAL_ADMIN");
```

Next

Syntax

Next()

Description

The Next method returns the next SelectedPagelet object in the SelectedPagelet collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A SelectedPagelet object.

Example

```
&MySelectedPagelet = &MyCollection.Next();
```

SelectedPagelet Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of SelectedPagelet objects in the SelectedPagelet Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Favorite Class

A favorite represents a content reference the user wants to keep a shortcut to. It contains the name of the content reference and the label the user wants to see displayed for this favorite. Labels must be unique in each favorite collection.

Use the Save method of the Favorite class to save any changes you made to any favorite objects.

A favorite object is instantiated by the First, InsertItem, ItemByName, and Next Favorite Collection methods.

See Chapter 27, "PortalRegistry Classes," Favorite Collection, page 1166.

Favorite Class Properties

In this section, we discuss the Favorite class properties. The properties are discussed in alphabetical order.

CRefName

Description

This property returns the name of the content reference as a string.

This property is read-only.

Label

Description

This property specifies the description of the favorite that's displayed to the user. This label must be unique for the favorite collection.

This property is read-write.

QualifiedURL

Description

This property returns the qualified URL for the content reference based on the Node and the URL for the name of the content reference, as a string.

This property is read-only.

SequenceNumber

Description

This property returns the sequence number used to order favorites.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

URL

Description

This property returns the relative URL for this favorite as a string.

For example, the following code:

```
&URL = &MyFavorite.URL;
```

could return the following string:

```
c/PORTAL_PERS_HOMEPAGE.PORTAL_HOMEPAGE.GBL?PORTALPARAM_PAGE=CONTENT&tab=⇒
DEFAULT&PortalFavorite=QEDMO
```

This property is read-write.

Favorite Collection

The favorite collection provides access to the favorites for a particular user.

A favorite collection is instantiated by the Favorites PortalRegistry property.

See Chapter 27, "PortalRegistry Classes," Favorites, page 1042.

Favorite Collection Methods

In this section, we discuss the Favorite collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*FavoriteLabel*)

Description

The DeleteItem method deletes the Favorite object identified by *FavoriteLabel* from the Favorite Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for deletion, then actually deleted later. The item is deleted from the database *as soon as* the method is executed.

Parameters

Parameter	Description
FavoriteLabel	Specify the label of a Favorite existing in the Favorite collection, that is, the text used to identify the favorite to the end-user.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A Boolean value: True if the Favorite was deleted, False otherwise.

Example

```
If Not &MyColl.DeleteItem("My Local Test") Then
   /* can't delete test data. Do error processing */
End-if;
```

First

Syntax

First()

Description

The First method returns the first Favorite object in the Favorite collection.

Parameters

None.

Returns

A Favorite object.

Example

```
&MyFav = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*FavoriteLabel*, *FavoriteName*)

Description

The InsertItem method inserts the Favorite object identified by *FavoriteName* into the Favorite Collection.

Note. The PortalRegistry classes execute some methods "interactively", that is, as they happen. The item won't be marked for insertion, then actually inserted later. The item is inserted to the database *as soon as* the method is executed.

Parameters

Parameter	Description
FavoriteLabel	Specify a label for the new Favorite. This is the text by which the favorite is identified to the end-user.
FavoriteName	Specify the name of a new Favorite. This parameter takes a string value. If you specify a name that already exists in the collection, you get an error.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to the new Favorite object if the method executed successfully, NULL otherwise.

ItemByLabel

Syntax

ItemByLabel(*FavoriteLabel*)

Description

The ItemByLabel method returns the Favorite object with the label *FavoriteLabel*. The label is the text used to identify the favorite to the end-user.

Parameters

Parameter	Description
FavoriteLabel	Specify the label of an existing Favorite within the Favorite collection. If you specify an invalid favorite, the object returned is Null.

Returns

A Favorite object if successful, Null otherwise.

Example

```
&MyFavorite = &MyColl.ItemByLabel("My Local Login");
```

Next

Syntax

Next()

Description

The Next method returns the next Favorite object in the Favorite collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A Favorite object.

Example

```
&MyFavorite = &MyCollection.Next();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Save

Syntax

```
Save()
```

Description

Use the Save method to save any changes you made to the Favorite collection.

Parameters

None.

Returns

A Boolean value: True, the collection saved successfully, False otherwise.

Favorite Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Favorite objects in the Favorite Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count
```

PortalRegistry Classes Example

There are several actions you may want to perform using a PortalRegistry, such as adding a folder, changing permissions, and so on. The following examples discuss the most common tasks. In addition, there are example programs of accessing the PortalRegistry classes using language environments other than PeopleCode.

Changing PortalRegistry Properties

This example shows how to change the default template used by a PortalRegistry. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MySession;
Local ApiObject &MyPortal, &MyPortalColl;

/* Access the current session */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&MySession = %Session;
If NOT &MySession Then
   /* do error processing */
End-if;

&MyPortalColl = &MySession.FindPortalRegistries();

For &I = 1 to &MyPortalColl.Count

   &MyPortal = &MyPortalColl.Item(&I);

   If &MyPortal.DefaultTemplate = "HR99" Then
      &MyPortal.DefaultTemplate = "HR00"
      &MyPortal.Save();
   End-If

End-For;
```

To change PortalRegistry properties:

1. Get a Session object.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.)

   ```
   &MySession = %Session;
   ```

2. Get a PortalRegistry collection.

   Use the FindPortalRegistries method with no parameters to return a collection of all the PortalRegistries because you want to check all the registries for the invalid template.

   ```
   &MyPortalColl = &MySession.FindPortalRegistries();
   ```

   Use the Count property on the collection to loop through all the registries.

   ```
   For &I = 1 to &MyPortalColl.Count
   ```

3. Get a PortalRegistry object.

   You can access a PortalRegistry object from the PortalRegistry collection using the Item method.

   ```
   &MyPortal = &MyPortalColl.Item(&I);
   ```

4. Check for the invalid template and correct if necessary.

   Use the DefaultTemplate property both to check for the invalid template name, and to set the correct template. Save the PortalRegistry when you make a correction.

   ```
   If &MyPortal.DefaultTemplate = "HR99" Then
      &MyPortal.DefaultTemplate = "HR00"
      &MyPortal.Save();
   End-If;
   ```

You may want to do error checking after you save the PortalRegistry.

 Copyright © 1988-2007, Oracle. All rights reserved.

Adding a ContentProvider

The following example adds a ContentProvider to an existing PortalRegistry. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyPortal;
Local ApiObject &MyCPColl, &MyCProvider;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

/* Add a ContentProvider */

&MyCPColl = &MyPortal.ContentProviders;

&MyCProvider = &MyCPColl.InsertItem("HRMS_00");

&MyCProvider.URI = "http://MYMACHINE103100/servlets/iclientservlet/HRMS/";
&MyCProvider.DefaultTemplate = "MYPORTAL_HRMS";
&MyCProvider.Description = "Updated Content Provider for HRMS";

If NOT(&MyPortal.Save()) Then
   &ErrorCol = &MySession.PSMessages;
   For &I = 1 to &ErrorCol.Count
      &Error = &ErrorCol.Item(&I);
   /* do error processing */
   End-For;
&ErrorCol.DeleteAll();
End-If;
```

To add a ContentProvider:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   ```

Copyright © 1988-2007, Oracle. All rights reserved.

```
End-if;
```

3. Access the ContentProvider collection.

   You can only add a ContentProvider to a ContentProvider collection. So you must access a ContentProvider collection first, using the ContentProviders property on a PortalRegistry.

```
&MyCPColl = &MyPortal.ContentProviders;
```

4. Add the ContentProvider.

   You must use the name of the ContentProvider with the InsertItem method. This example uses HRMS_00. This method does *not* execute automatically, that is, the ContentProvider isn't actually inserted into the database until the PortalRegistry is saved.

```
&MyCProvider = &MyCPColl.InsertItem("HRMS_00");
```

5. Further define the ContentProvider.

   The URI of a ContentProvider is used in conjunction with content references to define the location of content. So you should define the URI. If a template isn't found for any of the content references on a page at runtime, the system next tries to use the DefaultTemplate defined for a ContentProvider, so you should define a default template.

```
&MyCProvider.URI = "http://MYMACHINE103100/servlets/iclientservlet/HRMS/";
&MyCProvider.DefaultTemplate = "MYPORTAL_HRMS";
&MyCProvider.Description = "Updated Content Provider for HRMS";
```

6. Save the PortalRegistry and check for errors.

   Because there is no Save method with a ContentProvider, you must save the PortalRegistry to complete your changes.

   You can check if there were any errors using the PSMessages property on the session object.

```
If NOT(&MyPortal.Save()) Then
   /* save didn&rsquo;t complete */
   &ErrorCol = &MySession.PSMessages;
   For &I = 1 to &ErrorCol.Count
      &Error = &ErrorCol.Item(&I);
      /* do error processing */
   End-For;
   &ErrorCol.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, delete it from the PSMessages collection.

Note. If you've called the PortalRegistry API an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Chapter 36, "Session Class," Error Handling, page 1512.

Adding a Folder

The following example checks to see if a folder for a user exists. If it doesn't, it creates one.

 Copyright © 1988-2007, Oracle. All rights reserved.

The folder hierarchy for this example is as follows. The Users folder is where the portal stores things such as user homepages:

```
Root folder
  Users folder
     UserId1 folder
     UserId2 folder
     UserId3 folder
     . . .
```

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyPortal;
Local ApiObject &UserFldrColl, &UserFldr;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

&UserFldrColl = &MyPortal.RootFolder.Folders.ItemByName("Users").Folders;

&UserFldr = &UserFldrColl.ItemByName(%UserId);

If &UserFldr = Null Then

/* add Folder */

   &UserFldr = &UserFldrColl.InsertItem(%UserId, %UserId);
   &UserFldr.Description = %UserId;

/* Set dates */

   &UserFldr.ValidFrom = %Date;
   &ToDate = AddToDate(%Date, 1, 1, 0);
   &UserFldr.ValidTo = &ToDate;

/* Set properties */

   &UserFldr.PublicAccess = True;

/* save the folder */

   &UserFldr.Save();

End-If;
```

To add a folder:

1. Get a Session object and a PortalRegistry.

Copyright © 1988-2007, Oracle. All rights reserved.

Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

```
&MyPortal = %Session.GetPortalRegistry();
```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

```
If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;
```

3. Access the User folder collection.

   One of the strengths of dot notation is being able to string together many methods and properties into a single line of code. The result of this single line of code is to return a reference to the folder collection within the Users folder.

```
&UserFldrColl = &MyPortal.RootFolder.Folders.ItemByName("Users").Folders;
```

   Note that in this kind of code, you must access the RootFolder for the PortalRegistry before you can access the folders collection.

   If there were additional folders in your hierarchy, you could continue in the same way, using ItemByName and Folders.

4. Check if the user already has a folder.

   The system variable %UserId returns the user ID of the current user. Then this example uses the ItemByName method on the folder collection to see if the user already has a folder. This assumes that the user folders are names according to the UserId. The ItemByName method returns a reference to the folder if it exists. If the folder does not exist, ItemByName returns NULL.

```
&UserFldr = &UserFldrColl.ItemByName(%UserId);


If &UserFldr = Null Then
```

5. Add a folder if one doesn’t exist.

   Use the UserId as the name of the folder, as well as for the label and the description.

```
&UserFldr = &UserFldrColl.InsertItem(%UserId, %UserId);
&UserFldr.Description = %UserId;
```

6. Set the ValidFrom and ValidTo dates.

   For this example, the folder should be accessible from the date it’s created, so the example sets the ValidFrom property to be today’s date.

   When you first create a folder, the ValidTo date, that is, the date that this folder "expires", is set to null, that is an empty string. This means it never expires. In this example, we set the ValidTo date to one year and one day after the current date.

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&UserFldr.ValidFrom = %Date;
 &ToDate = AddToDate(%Date, 1, 1, 0);
 &UserFldr.ValidTo = &ToDate;
```

7. Set permissions for the folder.

   There are several properties that work together to determine the access of a folder or content reference. This example sets the PublicAccess property to True, which means that everyone has access to it, and it's included in all collections of folders.

   ```
   &UserFldr.PublicAccess = True;
   ```

   This example sets only the permission properties on the folder. It doesn't set all the possible permissions, such as the PermissionValue objects for the folder.

8. Save the folder.

   After you have finished your changes to the folder, save it.

   ```
   &UserFldr.Save();
   ```

   You may want to check for errors after you save each folder.

See Chapter 27, "PortalRegistry Classes," Using Security, page 1001.

Adding a Content Reference

The following code example adds a content reference that's a target component to the Approve Expenses folder.

The folder hierarchy for this example is:

```
Root folder
  Expenses folder
     Create Expenses folder
     Review Expenses folder
     Approve Expenses folder
```

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyPortal;
Local ApiObject &CRef, &CRefColl;
Local ApiObject &Fldr;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

&Fldr = &MyPortal.RootFolder.Folders.ItemByName("Expenses").Folders.ItemByName⇒
("Approve Expenses");

/* add content reference */

&CRef = &Fldr.ContentRefs.InsertItem(&CRefname, MYCP, MYRECORD.MYURL);
   &CRef.Description = &CRefname;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Set dates */
&CRef.ValidFrom = %Date;
&ToDate = AddToDate(%Date, 1, 1, 0);
&CRef.ValidTo = &ToDate;

/* Set content reference  types */
&CRef.UsageType = "TARG";   /* Target content reference */
&CRef.URLType = "UPGE";     /* Component type of content reference */
&CRef.StorageType = "RMTE";  /* Template is remote. */
&CRef.Template = "EXPENSES_TEMPLATE";      /* name of template */
&CRef.TemplateType = "HTML";   /* type of Template */
/* Set URL */
&CRef.URL = "ICType=Panel&Menu=MANAGE_EXPENSES&Market=GBL&Component=APPROVE_FINAL"

/* Save the content reference */

&CRef.Save();
```

To add a content reference:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   End-if;
   ```

3. Access the folder collection.

   One of the strengths of dot notation is being able to string together many methods and properties into a single line of code. The result of this single line of code is to return a reference to the folder collection within the folder.

   ```
   &Fldr = &MyPortal.RootFolder.Folders.ItemByName("Expenses").Folders.ItemByName⇒
   ("Approve Expenses");
   ```

   The Adding a Folder example has more explanation on the break down of this line of code.

4. Add the content reference.

 Copyright © 1988-2007, Oracle. All rights reserved.

You can only add content references from the collection of content references for the folder. After you have the content reference collection, use the InsertItem method to add the content reference. The ContentProvider for this content reference is named MYCP. The URL is stored in the record MYRECORD in the field MYURL.

```
&CRef = &Fldr.ContentRefs.InsertItem(&CRefname, MYCP, MYRECORD.MYURL);
&CRef.Description = &CRefname;
```

Note that the InsertItem method adds the content reference to the database as soon as it's executed. You could check for errors at this point to see if the content reference was added correctly.

5. Set the ValidFrom and ValidTo dates.

   For this example, the content reference should be accessible from the date it's created, so the example sets the ValidFrom property to be today's date.

   When you first create a content reference, the ValidTo date, that is, the date that this content reference "expires", is set to null, that is an empty string. This means it never expires. In this example, we set the ValidTo date to one year and one day after the current date.

   ```
   &CRef.ValidFrom = %Date;
   &ToDate = AddToDate(%Date, 1, 1, 0);
   &CRef.ValidTo = &ToDate;
   ```

6. Set the type parameters.

   Many content reference properties work together to set the type of content reference.

   In this example, the content reference is a target. It's a component type of content reference, which means it's a PeopleSoft definition. The template to be used for displaying this content reference is stored remotely, its name is EXPENSES_TEMPLATE, and it's an HTML template.

   ```
   &CRef.UsageType = "TARG"; /* Target content reference */
   &CRef.URLType = "UPGE"; /* Component type of content reference */
   &CRef.StorageType = "RMTE"; /* Template is remote. */
   &CRef.Template = "EXPENSES_TEMPLATE"; /* name of template */
   &CRef.TemplateType = "HTML"; /* type of Template */
   ```

7. Set the URL.

   The URL property of the content reference indicates where the content actually is. As this content reference is referencing a page, the URL has a specific format to indicate which page.

   ```
   &CRef.URL = "ICType=Panel&Menu=MANAGE_EXPENSES&Market=GBL&Component=APPROVE_FINAL"
   ```

8. Save the content reference.

   After you have finished adding the content reference, you should save it.

   ```
   &CRef.Save();
   ```

   You may want to check for errors after you save each content reference.

See Chapter 27, "PortalRegistry Classes," Adding a Folder, page 1172.

Setting Permissions Using the PermissionValue Object

The following code example adds permissions for a folder through the PermissionValue object.

The following is the complete code sample: the steps explain each line.

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local ApiObject &MyPortal;
Local ApiObject &UserFldr;
Local ApiObject &PermV, &PermVColl;

&MyPortal = %Session.GetPortalRegistry();

If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
   /* Do error handling */
End-if;

&UserFldr = &MyPortal.RootFolder.Folders.ItemByName("Users");

&CascadedColl = &UserFldr.CascadedPermissions;

&PermV = &CascadedColl.ItemByName("VENDOR1");

If NOT &PermV Then
   &PermVColl = &UserFldr.Permissions;
   &PermVColl.InsertItem("VENDOR1");
   &UserFldr.Save();
End-If;
```

To set permissions for a folder, using the PermissionValue object:

1. Get a Session object and a PortalRegistry.

   Before you can access a PortalRegistry object, you must get a session object. The session controls access to the registry, provides error tracing, enables you to set the runtime environment, and so on. Because you want to use the existing session, use the %Session system variable (instead of the GetSession function.) In addition, you want to get a PortalRegistry. Using the GetPortalRegistry method returns a reference to an unpopulated PortalRegistry object.

   ```
   &MyPortal = %Session.GetPortalRegistry();
   ```

2. Open the PortalRegistry.

   After you get a PortalRegistry object, you want to open it, that is, populate it with data. Use the Open method to do this. The Open method returns a Boolean value, and this example uses that value to do error checking to make sure that the PortalRegistry is actually opened. In addition, the name of the PortalRegistry is kept in the record MYRECORD, in the field PORTAL_NAME.

   ```
   If NOT &MyPortal.Open(MYRECORD.PORTAL_NAME) Then
      /* Do error handling */
   End-if;
   ```

3. Access the User folder.

   One of the strengths of dot notation is being able to string together many methods and properties into a single line of code. The result of this single line of code is to return a reference to the folder named Users.

   ```
   &UserFldr = &MyPortal.RootFolder.Folders.ItemByName("Users");
   ```

4. Access the entire, cascaded PermissionValue collection.

   There are two types of PermissionValue collections you can access for a folder.

 Copyright © 1988-2007, Oracle. All rights reserved.

- One contains only the permission list values that are set at that folder. This collection is returned with the Permissions property.
- The other contains all the permission list values for folder, that is, it contains any permission list values set in any parent folder and cascaded. This collection is returned with the CascadedPermissions property.

Note. You *cannot* add any PermissionValue objects to a collection returned by the CascadedPermissions property. You can add values only to the collection returned by the Permissions property.

This example uses the CascadedPermissions collection to check if the permission exists, because we don't want to add a permission if it already exists. Then it uses the Permissions collection to add the value.

```
&CascadedColl = &UserFldr.CascadedPermissions;
```

1. Check to see if the permission list value exits.

   Before adding the new PermissionValue object, you want to make sure one by that name doesn't already exist.

   ```
   &PermV = &CascadedColl.ItemByName("VENDOR1");
   ```

2. If the PermissionValue doesn't exist, add it.

   The ItemByName returns a reference to a PermissionValue object if it exists, or NULL if it doesn't. So this example checks for NULL, and adds the PermissionValue if it doesn't exist.

   Notice that this example is *not* changing the Cascaded property with the PermissionValue object. The default value for a new PermissionValue object is False. As this example does not want to cascade the permissions for this permission list, it retains the default value.

   In addition, if the PermissionValue object is added, the folder is saved. The InsertItem method executes only after the parent object, the folder, is saved.

   ```
   If NOT &PermV Then
      &PermVColl = &UserFldr.Permissions;
      &PermVColl.InsertItem("VENDOR1");
      &UserFldr.Save();
   End-If;
   ```

   You may want to check for errors after you save the folder.

See Chapter 27, "PortalRegistry Classes," Permissions, page 1065 and Chapter 27, "PortalRegistry Classes," CascadedPermissions, page 1062.

See Chapter 27, "PortalRegistry Classes," Using PermissionValue, RolePermissionValue, Cascading Permissions and CascadingRolePermissions, page 1002.

Using Attributes

The following example uses attributes for a folder to determine what is displayed to an end-user. The end-user still has access to the content, however, it isn't displayed. This program doesn't discuss how to hide or display content: it just shows the function used to determine if a folder should be displayed.

The following is the complete code sample: the steps explain each line.

```
Function checkVisible(&FolderId As ApiObject) Returns Boolean
   &colAttrib = &FolderId.Attributes;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV") <> Null Then
      If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV").value = "TRUE" Then
            &display = False;
      End-If;
   End-If;

    Return &display;

End-Function;
```

To use attributes for a folder:

1. Access the folder attribute collection.

   A reference to a folder is passed into the function. Then the example uses the Attributes property on the folder to access the attribute collection for the folder.

   ```
   &colAttrib = &FolderId.Attributes;
   ```

2. Check to see if the folder should be hidden.

   This step is actually two checks. The first check is to see if there is an attribute with the folder called PORTAL_HIDE_FROM_NAV. If the folder has such an attribute, the second check determines the value of this attribute. If both are true, the folder should be hidden, so the variable &display is set to True, and returned.

   ```
   If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV") <> Null Then
         If &colAttrib.ItemByName("PORTAL_HIDE_FROM_NAV").value = "TRUE" Then
               &display = False;
         End-If;
      End-If;

       Return &display;
   ```

Visual Basic Example

The following is an example referencing a PortalRegistry object using Visual Basic.

```
Dim oCref As ContentRef
Dim oCrefColl As ContentRefCollection

Set oSession = CreateObject("PeopleSoft.Session")
iResult = oSession.Connect(1, AppServStr, OperID, OperPasswd, 0)

Set oPortal = oSession.GetPortalRegistry
iResult = oPortal.Open("PORTAL")

Set oCrefColl = oPortal.RootFolder.ContentRefs
Set oCref = oCrefColl.InsertItem("papi012")
iResult = oPortal.Close

iResult = oPortal.Open("PORTAL")

Set oCref = oPortal.FindCRefByName("papi012")
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
oPortal.Close
oSession.Disconnect
Exit Sub
```

C/C++ Example

The following is a C/C++ example.

```
/**************************************************************************
*                                                                         *
* psportal_test.cpp                                                       *
*                                                                         *
**************************************************************************
*                                                                         *
*               Confidentiality Information:                              *
*                                                                         *
* This module is the confidential and proprietary information of
* PeopleSoft, Inc.; it is not to be copied, reproduced, or
* transmitted in any form, by any means, in whole or in part,
* nor is it to be used for any purpose other than that for which it
* is expressly provided without the written
* permission of PeopleSoft.
*                                                                         *
* Copyright (c) 1988-1999 PeopleSoft, Inc.  All Rights Reserved.          *
*                                                                         *
**************************************************************************
*                                                                         *
* SourceSafe Information:                                                 *
*                                                                         *
* $Logfile::                                                             $*
* $Revision::                                                            $*
* $Date::                                                                $*
*                                                                         *
**************************************************************************/
/**************************************************************************
*       includes                                                          *
**************************************************************************/

#ifdef PS_WIN32
#include "stdafx.h"
#endif

#include "bcdef.h"
#include "apiadapterdef.h"
#include "peoplesoft_peoplesoft_i.h"

#include <stdio.h>
#include <iostream.h>
#include <wchar.h>
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/*********************************************************************
*       general defines and macros                                   *
*********************************************************************/

/*********************************************************************
*       globals                                                      *
*********************************************************************/
HPSAPI_SESSION                                          hSession;


/*********************************************************************
*       function prototypes                                          *
*********************************************************************/
void DisconnectSession();
void GetFolder(HPSAPI_SESSION hSession, LPTSTR pszPortalName);
void OutError(HPSAPI_SESSION hSession);

/*********************************************************************
* Function:     main                                                 *
*                                                                    *
* Description:                                                       *
*                                                                    *
* Returns:                                                           *
*********************************************************************/
void main(int argc, char* argv[])
{
    // Declare variables
    PSAPIVARBLOB    blobExtra;
    TCHAR szServer[80] = _T("//LCHE0110:900");
    TCHAR szUserid[30] = _T("PTMO");
    TCHAR szUserPswd[80] = _T("PT");
    TCHAR szPortalName[80] = _T("PORTAL");

    memset(&blobExtra, 0, sizeof(PSAPIVARBLOB));

    // Establish a PeopleSoft Session.
   HPSAPI_SESSION hSession = PeopleSoft_Session_Create();

    if (PeopleSoft_Session_Connect(hSession, 1, szServer, szUserid, szUserPswd,⇒
 blobExtra))
    {
        GetFolder(hSession, szPortalName);
   }
   else
    {
      // Connect to AppServer Failed. Error out.
        if (PeopleSoft_Session_GetErrorPending(hSession))
      {
         wprintf(L"\nConnect to AppServer Failed.\n");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
            OutError(hSession);
        }
        else
            wprintf(L"No error pending from session.\n");
    }
    DisconnectSession();
}


/*********************************************************************
*       function implementations                                     *
*********************************************************************/
/*********************************************************************
* Function:     DisconnectSession                                    *
*                                                                    *
* Description:  Disconnects the Session object.                      *
*                                                                    *
* Returns:      None                                                 *
*********************************************************************/
void DisconnectSession()
{
    // Disconnect current session to free memory and session variables.
    if (hSession)
    {
        if (PeopleSoft_Session_Disconnect(hSession))
        {
            PeopleSoft_Session_Release(hSession);
        }
        else
        {
            wprintf(L"Disconnect to AppServer Failed.\n");
        }
    }
}


/*********************************************************************
* Function:     GetFolder                                            *
*                                                                    *
* Description:  Get root folder and display folder name              *
*                                                                    *
* Returns:      None                                                 *
*********************************************************************/
void GetFolder(HPSAPI_SESSION hSession, LPTSTR pszPortalName)
{
    LPTSTR                  szStr;
    HPSAPI_PORTALREGISTRY   hPortal;
    HPSAPI_FOLDER           hRootFolder;

    hPortal=(HPSAPI_PORTALREGISTRY) PeopleSoft_Session_GetPortalRegistry(hSession);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
    PortalRegistry_PortalRegistry_Open(hPortal, pszPortalName);
    if (PortalRegistry_PortalRegistry_Open(hPortal, pszPortalName) == false)
        _tprintf(_T("Open portal failed\n"));
    else
    {
        hRootFolder = PortalRegistry_PortalRegistry_GetRootFolder(hPortal);
        szStr = PortalRegistry_Folder_GetName(hRootFolder);
        _tprintf(_T("root folder name = %s\n"), szStr);
    }
}


/*********************************************************************
* Function:     OutError                                             *
*                                                                    *
* Description:  Output error message from session psMessage object   *
*                                                                    *
* Returns:      None                                                 *
*********************************************************************/
void OutError(HPSAPI_SESSION hSession)
{
    LPTSTR                  szStr;
    HPSAPI_PSMESSAGECOLLECTION hMsgColl;
   HPSAPI_PSMESSAGE hMsg;


    hMsgColl=(HPSAPI_PSMESSAGECOLLECTION) PeopleSoft_Session_GetPSMessages(h⇒
Session);

   wprintf(L"Get psMessage collection ok.\n");

   int i;
   PSI32 count;
   count=(PSI32) PeopleSoft_PSMessageCollection_GetCount;
   wprintf(L"Count= %d\n", count);

   for (i=0; i<count; i++)
   {
      hMsg=(HPSAPI_PSMESSAGE) PeopleSoft_PSMessageCollection_Item(hMsgColl, i);
      szStr=PeopleSoft_PSMessage_GetText(hMsg);
      wprintf(L"\nMessage from session: %s\n",szStr);
   }

}
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 28

PostReport Class

This chapter provides an overview of PostReport class and discusses the following topics:

- Data type of a PostReport object
- Scope of a PostReport object
- Determining the distribution list
- PostReport class built-in functions
- PostReport class properties
- PostReport examples

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Using Report Manager"

PostReport Class Overview

The PostReport class provides properties and method for posting reports to the Report Repository. This class enables you to create reports outside of Process Scheduler and have them posted in the Report Repository.

The request to post reports creates an entry to the Report Manager table. The Distribution Agent polls the Report Manager table for any new requests and proceeds with any request it finds by transferring the files to the repository server. The report can then be accessed from the Report Manager after the Distribution Agent successfully processes the post request.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Using Report Manager"

Determining the Distribution List

Use the AddDistributionOption method to authorize users to view the report from the Report Manager. This function enables you to grant access to the report either specifying a User ID or Role. However, if no distribution list is specified in the request, the class queries the system for the distribution list as follows:

1. If the Process Instance is specified, the system checks for the distribution list in the Process Request table. The system assumes in this case that the request was scheduled through the Process Request Dialog, but was processed by a third-party scheduler.

Copyright © 1988-2007, Oracle. All rights reserved.

2. If a Process Name and Process Type properties were specified, the system checks if a distribution list was predefined in the Process Definition table.
3. If no list was found using the above queries, the user who submitted the request has sole access to the report.

Data Type of a PostReport Object

A PostReport object is declared using the PostReport data type. For example:

```
Local PostReport &POSTRQST;
```

Scope of a PostReport Object

A PostReport object can only be instantiated from PeopleCode. However, if you need to access the Process Schedule Manger from outside of PeopleCode, like in a Visual Basic program, you can use the Process Schedule Manager API.

This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Understanding PeopleSoft Process Scheduler"

PostReport Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetPostReport

PostReport Class Methods

In this section, we discuss the PostReport class methods. The methods are discussed in alphabetical order.

AddDistributionOption

Syntax

AddDistributionOption(*DistIdType, DistId*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AddDistributionOption method to specify the users authorized to view the report once it is available in the Report Manager.

Parameters

Parameter	Description
DistIdType	Specify the distribution ID type as a string. This identifies if the value passed in the DistID is either a user or role. Values for this parameter are: • User • Role
DistID	Specify the distribution ID as a string.

Returns

Returns a number: 0 if successful, 1 if the system detected an error in the parameter passed.

Example

The following example grants access to the VP1 user and all users associated with the MANAGERS role.

```
&POSTRQST.AddDistributionOption("User", "QEDMO" );
&POSTRQST.AddDistributionOption("Role", "MANAGERS");
```

Put

Syntax

```
Put()
```

Description

The Put method inserts a request in the Report Manager table which runs according to the values in the properties of the PostReport object. In order to successfully schedule a process or job, certain properties are required.

Parameters

None.

Returns

None. If you want to verify that the method executed successfully, check the value of the Status property.

Example

```
&POSTRQST.Put();
&RPTINSTANCE = &POSTRQST.ReportId;
If (&RPTINSTANCE > 0) Then
      MessageBox(0, "", 63, 119, "Successfully processed request with Rpt. ID %1⇒
 for Process %2 to post from directory %3", &RPTINSTANCE, &POSTRQST.ProcessName,⇒
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
 &POSTRQST.SourceReportPath);
   Else
      MessageBox(0, "", 63, 122, "Not successful for process request for Process⇒
 %1 to post from directory %2", &POSTRQST.ProcessName, &POSTRQST.SourceReportPath);
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetPostReport

PostReport Class Properties

In this section, we discuss the PostReport class properties. The properties are discussed in alphabetical order.

ExpirationDate

Description

This property specifies the date the report will be deleted from the Report Manager table. If the expiration date is not specified, the system calculates the expiration date based on the Retention Days specified in the Process Scheduler's System Settings.

This property is read-write.

Example

```
&MYRQST.ExpirationDate = "2003/01/01";
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Defining PeopleSoft Process Scheduler Support Information," Defining System Settings

OutDestFormat

Description

This property specifies the output format of the file that is posted to the Report Repository. If format is not specified, the system determines the format based on the extension of the file being posted to the Report Repository.

This property is read-write.

Example

```
&MYRQST.OutDestFormat = "PDF";
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 29, “ProcessRequest Class,” Values for Output Type and Format, page 1198

ProcessInstance

Description

This property specifies the instance number assigned to request. This property is only required if the report was initially submitted through the Process Request Dialog, but was processed by a third-party scheduler.

This property is read-write.

See Also

Chapter 28, “PostReport Class,” Determining the Distribution List, page 1185

ProcessName

Description

This property specifies the name of a predefined process as a string.

This property is read-write.

Example

```
&MYRQST.ProcessName = "XRFWIN";
```

See Also

Chapter 28, “PostReport Class,” ProcessType, page 1189

Chapter 29, “ProcessRequest Class,” RunControlID, page 1223

ProcessType

Description

This property specifies the name of a predefined process type as a string.

This property is read-write.

Example

Note that spaces are included in the string for ProcessType.

```
&MYRQST.ProcessType = "Application Engine";
```

See Also

Chapter 28, “PostReport Class,” ProcessName, page 1189

Chapter 29, “ProcessRequest Class,” RunControlID, page 1223

Copyright © 1988-2007, Oracle. All rights reserved.

ReportDescr

Description

This property specifies the description as a string.

If the ReportDescr property is not specified, and both Process Name and Process Type are not null, the description is extracted from the Process Definition table.

This property is read-write.

Example

```
&MYRQST.ReportDescr = "Panel Cross Reference Report";
```

ReportFolder

Description

This property specifies the name of the report folder as a string. If you specify a folder, the folder that you specify must have already been defined using Process Scheduler's Report Folders Administration.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Using Report Manager," Understanding Report Folders

ReportId

Description

This property is a system-generated identification number. The system assigns a ReportId after the Put method processed the request.

This property is read-write.

Example

```
&MYRQST.Put();
if &MYRQST.ReportId = 0 then
     MessageBox(o, "", 63, 119, "Sucessfully processed request with Rpt. ID %1",⇒
 &MYRQST.ReportId);
Else
    /* do error processing */
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ServerName

Description

This property specifies the Process Scheduler Server name that posts the report. This property takes a string value.

This property is read-write.

Example

```
&MYRQST.ServerName = "PSNT";
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Defining PeopleSoft Process Scheduler Support Information," Defining System Settings

SourceReportPath

Description

This property specifies the path that contains the source report. You need to specify an absolute path.

This property is read-write.

PostReport Class Examples

The following example posts files found in the directory 'c:\temp\SQRDIR' and can be accessed by VP1 and users in the MANAGERS role.

```
Local PostReport &RPTINFO;
   Local number &RPTINSTANCE

   /*********************************************************************
   * Construct a PostReport Object.                                     *
   *********************************************************************/
   &RPTINFO = SetPostReport();

   &RPTINFO.ProcessName = "GLS7009";
   &RPTINFO.ProcessType = "SQR Report";
   &RPTINFO.ReportFolder = "Financial";
   &RPTINFO.SourceReportPath = "c:\temp\SQRDIR";
   &RPTINFO.ExpirationDate = "2005/01/01";
   &RPTINFO.ReportDescr = "Journal Posting Summary Report";
   &RPTINFO.ServerName = "PSNT";

   &RPTINFO.AddDistributionOption("USER", "VP1");
   &RPTINFO.AddDistributionOption("ROLE", "MANAGERS");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
  &RPTINFO.Put();
  &RPTINSTANCE = &RPTINFO.ReportId;

  If (&RPTINSTANCE > 0) Then
     MessageBox(0, "", 63, 119, "Successfully processed request with Rpt. ID %1⇒
for Process %2 to post from directory %3", &RPTINSTANCE, &RPTINFO.ProcessName,⇒
&RPTINFO.SourceReportPath);
  Else
     MessageBox(0, "", 63, 122, "Not successful for process request for Process⇒
%1 to post from directory %2", &RPTINFO.ProcessName, &RPTINFO.SourceReportPath);
  End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 29

ProcessRequest Class

This chapter provides an overview of ProcessRequest class and discusses the following topics:

- Data type of a ProcessRequest object.
- Scope of a ProcessRequest object.
- Options for items in jobs and jobsets.
- Values for output type and format.
- Alternative option to specify email or web attributes.
- ProcessRequest class built-in function.
- ProcessRequest class methods.
- ProcessRequest class properties.
- ProcessRequest examples.

Understanding ProcessRequest Class

The ProcessRequest class provides properties and a method for scheduling a pre-defined process or job. You must define the process or the job (using Process Scheduler Manager) *before* you can schedule it using the ProcessRequest class.

The properties of this class contain the same values as those required by Process Scheduler Manager for scheduling a process or job. Values you provide for these properties may override the equivalent values set in Process Scheduler Manager, depending on the override settings you make in PeopleSoft Process Scheduler pages.

Starting with PeopleSoft 8.4, PeopleSoft supports only the ProcessRequest class to schedule a process or a job in PeopleCode. The ScheduleProcess PeopleCode function is no longer supported. Any existing PeopleCode functions containing this deprecated function must be modified to use the ProcessRequest class to schedule a request.

This chapter assumes that the reader has working knowledge of PeopleSoft Process Scheduler.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler

Copyright © 1988-2007, Oracle. All rights reserved.

Data Type of a ProcessRequest Object

A ProcessRequest object is declared using the ProcessRequest data type. For example:

```
Local ProcessRequest &RQST;
```

Scope of a ProcessRequest Object

A ProcessRequest object can be instantiated only from PeopleCode. However, if you need to access the Process Schedule Manager from outside of PeopleCode, for example, in a Visual Basic program, you can use the Process Schedule Manager API.

This object can be used anywhere you have PeopleCode, that is, in an application class, Component Interface PeopleCode, record field PeopleCode, and so on.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler

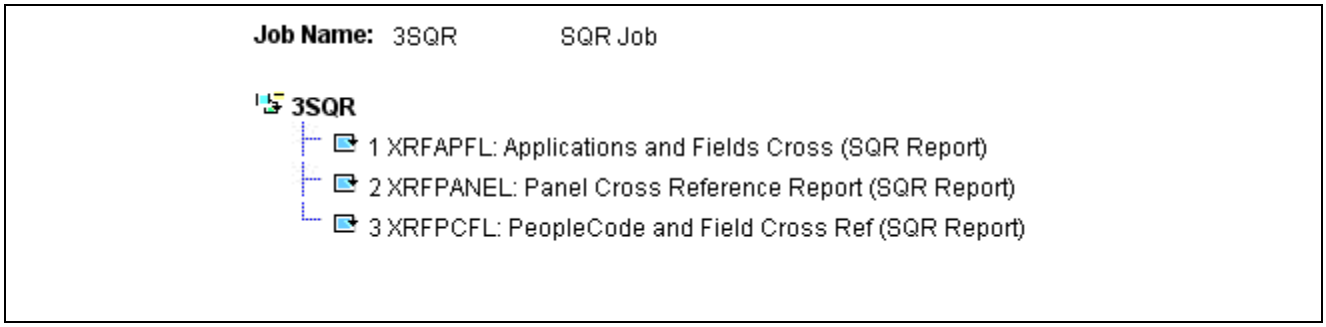
Options for Items In Jobs and Jobsets

A key feature of PeopleSoft Process Scheduler is the option to create a Job Definition consisting of other jobs. A job containing a job is called a *jobset*. Process Scheduler Server schedules the items within these jobs according to how the jobset is defined through the Process Scheduler Manager.

The ProcessRequest class requires the following properties to handle jobsets:

- JobName: indicates the name of the job that an item belongs to.
- PrcsItemLevel: indicates a job item's process item level within a jobset.
- JobSeqNo: indicates a job item's job sequence number within the Process Item Level.

The following is a simple example of a job containing only a single process:

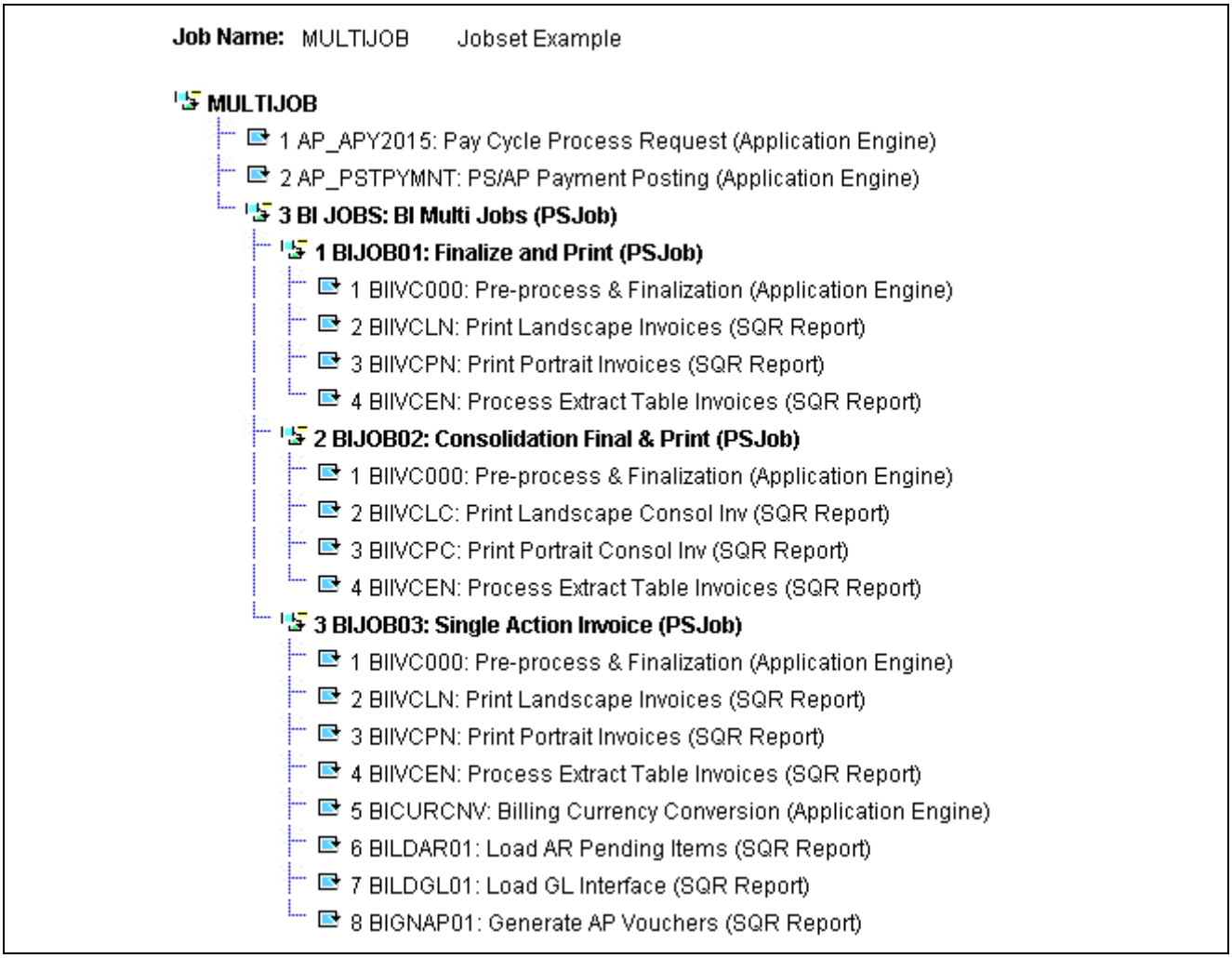


Sample job with single process

 Copyright © 1988-2007, Oracle. All rights reserved.

Process	PrcsItemLevel	JobName	JobSeqNo
3SQR	0		0
XRFAPFL	1	3SQR	1
XRFPANEL	2	3SQR	2
XRFPCFL	3	3SQR	3

The following is an example of a jobset containing multiple jobs:



Sample jobset containing multiple jobs

Copyright © 1988-2007, Oracle. All rights reserved.

Process	PrcsItemLevel	JobName	JobSeqNo
MULTIJOB	0		0
AP_APY2015	1	MULTIJOB	1
AP_PSTPYMNT	1	MULTIJOB	2
BI JOBS	1	MULTIJOB	3
BIJOB01	2	BI JOBS	1
BIIVC000	3	BIJOB01	1
BIIVCLN	3	BIJOB01	2
BIIVCPN	3	BIJOB01	3
BIIVCEN	3	BIJOB01	4
BIJOB02	2	MULTIJOB	2
BIIVC000	3	BIJOB02	1
BIIVCLC	3	BIJOB02	2
BIIVCPC	3	BIJOB02	3
BIIVCEN	3	BIJOB02	4
BIJOB03	2	MULTIJOB	3
BIIVC000	3	BIJOB03	1
BIIVCLN	3	BIJOB03	2
BIIVCPN	3	BIJOB03	3
BIIVCEN	3	BIJOB03	4

 Copyright © 1988-2007, Oracle. All rights reserved.

Process	PrcsItemLevel	JobName	JobSeqNo
BICURCNV	3	BIJOB03	5
BILDAR01	3	BIJOB03	6
BILDGL01	3	BIJOB03	7
BIGNAP01	3	BIJOB03	8

All the class methods that are used to change options for items within a job or jobset, use these attributes as optional parameters.

If a method is used without specifying any of these parameters, the system assumes the changes apply to *all* items within a jobset.

If one or more of these attributes are specified, the method applies the changes to items matching the value of these attributes.

In the case where a JobName is specified, the changes apply to all items within the specified job.

Specific methods used to manipulate options within a job or jobset include:

- AddDistributionOption
- SetEmailOption
- SetItemFolder
- SetOutputOption

When manipulating the options within a job or jobset, you must specify the process type and process name in the CreateProcessRequest function when instantiating a ProcessRequest object. This is required as the class needs to retrieve the job item information from the job definition for a specific job or jobset. The class uses this information to alter items within the job or jobset based on subsequent method calls made prior to scheduling the request. If the process type and process name are not specified when instantiating the ProcessRequest object, all PeopleCode programs trying to use any of the methods return an error message:

```
The method can not be used because the ProcessRequest is not properly initialized
```

In the previous jobset example, the ProcessRequest must be instantiated as follows:

```
Local ProcessRequest &RQST;

&RQST = CreateProcessRequest("PSJob", "MULTIJOB");
```

The following code examples use the previous example MULTIJOB, and show how items can be altered within a job or jobset:

- Change all items in the Jobset to “Web” with reports generated in PDF format:

```
&RQST.SetOutputOption("Web", "PDF", "", "MULTIJOB");
```

  Alternatively, this can also be coded as follows:

```
&RQST.SetOutputOption("Web", "PDF", "");
```

- Change the output option for job BIJOB02 to “Email” with reports generated in HTM format:

Copyright © 1988-2007, Oracle. All rights reserved.

```
&RQST.SetOutputOption("Email", "HTM", ", "BIJOB02");
```

- Change the output option for BIIVCPN in job BIJOB01 to “File” in LP (LinePrinter) format:

```
&RQST.SetOutputOption("File", "LP", "BIJOB01", 3, 3);
```

- Change all items in level three to “Web” in PS (PostScript) format:

```
&RQST.SetOutputOption("Web", "PS", "", "", 3);
```

Values for Output Type and Format

The values for output type and format are based on the generic process type assigned to a process type definition. The following table provides a cross-reference listing for all delivered process type definitions.

Generic Process Type	Process Type
AppEngine	Application Engine, Optimization Engine
COBOL	COBOL SQL
Crystal	Crystal, Crystal Check
Cube	CubeBuilder, HyperCube Builder
SQR	SQR Process, SQR Report
Winword	Winword
nVision	nVision, nVision-Report, nVision-ReportBook
Data Mover	Data Mover
Other	Essbase

The following table lists all the values for OutDestType delivered with PeopleTools.

However, the values and defaults can be customized through Process Scheduler Manager. Please refer to the PeopleSoft Process Scheduler documentation for additional information on how to accurately determine the values for your system.

 Copyright © 1988-2007, Oracle. All rights reserved.

Generic ProcessType	Valid OutDestTypes	Defaults
AppEngine	FILE, WEB, WINDOW	WEB
COBOL	NONE, WINDOW, WEB	NONE
Crystal	WEB, WINDOW, EMAIL, FILE, PRINTER	WEB
Cube	NONE	NONE
nVision	WEB, WINDOW, EMAIL, FILE, PRINTER, DEFAULT	DEFAULT
SQR	WEB, WINDOW, EMAIL, FILE, PRINTER	WEB
WinWord	WINDOW, WEB	WEB
Data Mover	WEB,WINDOW, FILE	WEB
OTHER	WEB, WINDOW, EMAIL, FILE, PRINTER, NONE	NONE

Similar to the OutDestType, the formats and defaults for a generic process type can be customized through Process Scheduler Manager. The following table lists the values and defaults as delivered.

Generic ProcessType	OutDestType	Valid OutDestFormats	Default
AppEngine	WINDOW	PDF, XLS, TXT, HTM	TXT
AppEngine	FILE	PDF, XLS, TXT, HTM	TXT
AppEngine	WEB	PDF, XLS, TXT, HTM	TXT
COBOL	NONE	NONE	NONE
COBOL	WEB	TXT	TXT
COBOL	WINDOW	TXT	TXT
Crystal	PRINTER	RPT	RPT

Copyright © 1988-2007, Oracle. All rights reserved.

Generic ProcessType	OutDestType	Valid OutDestFormats	Default
Crystal	FILE	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Crystal	WINDOW	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Crystal	WEB	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Crystal	EMAIL	RPT, RTF, TXT, PDF, HTM, XLS, DOC	PDF
Cube	NONE	NONE	NONE
Data Mover	FILE	TXT	TXT
Data Mover	WINDOW	TXT	TXT
Data Mover	WEB	TXT	TXT
nVision	EMAIL	HTM, XLS	XLS
nVision	FILE	HTM, XLS	XLS
nVision	PRINTER	HTM, XLS	XLS
nVision	WINDOW	HTM, XLS	XLS
nVision	WEB	HTM, XLS	XLS
nVision	DEFAULT	DEFAULT	DEFAULT
SQR	EMAIL	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	PDF
SQR	FILE	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	PDF
SQR	PRINTER	HP, LP, PS, WP	PS

Copyright © 1988-2007, Oracle. All rights reserved.

Generic ProcessType	OutDestType	Valid OutDestFormats	Default
SQR	WEB	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	WEB
SQR	WINDOW	CSV, HP, HTM, LP, PDF, PS, SPF,OTHER	WEB
WinWord	NONE	NONE	NONE
WinWord	WEB	DOC	DOC
WinWord	WINDOW	DOC	DOC
OTHER	NONE	NONE	NONE

Alternative Option To Specify Email or Web Attributes

To set the Email or Web attributes in PeopleTools versions prior to release 8.4, your PeopleCode program was similar to the following:

To Set Attributes For Web

The following example shows how to set web attributes in PeopleTools version prior to release 8.4.

```
&RQST.OutDestType = "Web";
&RQST.OutDestFormat = "PDF";
&RQST.OutDest = "User : VP1,Role :Managers";
```

Set Attributes For Email

The following example shows how to set email attributes in PeopleTools version prior to release 8.4.

```
Local string &Subject ;
Local string &Text;

&Subject = "SQR Report:  Cross Reference Listing";
&Text = "This text will be displayed as the text of this email ";
&RQST.OutDestType = "Email";
&RQST.OutDestFormat = "PDF";
&RQST.OutDest = "User : VP1,Role : MANAGERS";
&RQST.EmailSubject = &Subject;
&RQST.EmailText = &Text;
&RQST.EmailAttachLog = False;
```

Copyright © 1988-2007, Oracle. All rights reserved.

In PeopleTools 8.4, methods are provided that set the email or web options. The sample code shown previously can be modified as follows:

Set Attribute for Web

Modify the previous example for setting web attributes as follows:

```
&RQST.SetOutputOption("Web", "PDF", "");
&RQST.AddDistributionOption("User", "QEDMO");
&RQST.AddDistributionOption("Role", "MANAGERS");
```

Set Attribute for Email

Modify the previous example for setting email attributes as followings:

```
Local string &Subject ;
Local string &Text;

&Subject = "SQR Report:  Cross Reference Listing";
&Text = "This text will be displayed as the text of this email ";
&RQST.SetOutputOption("Email", "PDF", "");
&RQST.SetEmailOption(&Subject, &Text, False);
&RQST.AddDistributionOption("User", "QEDMO");
&RQST.AddDistributionOption("Role", "MANAGERS");
```

File Dependant Processing

You can define a process to be file dependent, which means associating a file with a process that gets scheduled once the system detects the presence of the file.

You can use the PrcsApi class methods for detecting files associated with a process, as well as for displaying additional messages about the process as it's running.

The PrcsApi class is an Application Package class that you must import into your PeopleCode in order to use its methods.

For example, you could use the PrcsApi class with a PeopleSoft Application Engine program. In the Application Engine process PROCESS_EDI, you could create a file dependency (using the FileName ProcessRequest property) and make the process dependent on the file /vendor/edi*.data.

On 10/23, the system detects the file /vendor/edi_1023.data and schedules the process PROCESS_EDI. In the Application Engine program, you can refer to the file that started the process using the getAllFileNames PrcsApi class method.

On 10/24, the system detects the file /vendor/edi_1024.data and schedules the process PROCESS_EDI. The process is started from a different file, and can access that file, so duplicate processing doesn't occur.

See Also

Chapter 29, "ProcessRequest Class," PrcsApi Class, page 1228

 Copyright © 1988-2007, Oracle. All rights reserved.

ProcessRequest Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateProcessRequest

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetNextProcessInstance

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SetupScheduleDefnItem

ProcessRequest Class Methods

In this section, we discuss the ProcessRequest class methods. The methods are discussed in alphabetical order.

AddDistributionOption

Syntax

AddDistributionOption(*DistIdType*, *DistId* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])

Description

Use the AddDistributionOption method to set the distribution options for any job item in the main job. Distribution options enable you to distribute output in different formats (HTML, PDF, Excel, and so on) to other users based on their user ID or role ID.

This function is valid only if the output destination for the request is routed either to Web or Email.

Parameters

Parameter	Description
DistIdType	Specify the distribution ID type as a string. This identifies if the value passed in the DistID is either a user or role. Values for this parameter are: • User • Role
DistID	Specify the distribution ID as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the Process Item Level as a number.

Returns

Returns a number: 0 if successful, 1 if system detected an error in the parameter passed.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example grants access to the QEDMO user all reports in MULTIJOB from the Web, while users with role of MANAGERS have access to reports created in the BIJOB03 job.

```
&RQST.SetOutputOption("Web", "PDF", "");
&RQST.AddDistributionOption("User", "QEDMO", "MULTIJOB" );
&RQST.AddDistributionOption("Role", "MANAGERS", "BIJOB03");
```

See Also

Chapter 29, "ProcessRequest Class," SetEmailOption, page 1212 and Chapter 29, "ProcessRequest Class," SetOutputOption, page 1214.

PrintJobHTMLRpt

Syntax

PrintJobHTMLRpt()

Description

Use the PrintJobHTMLRpt method to generate an HTML report file displaying all items in a job or jobset in a tree as defined in the Job Definition component.

Parameters

None.

Returns

An HTML report as a string

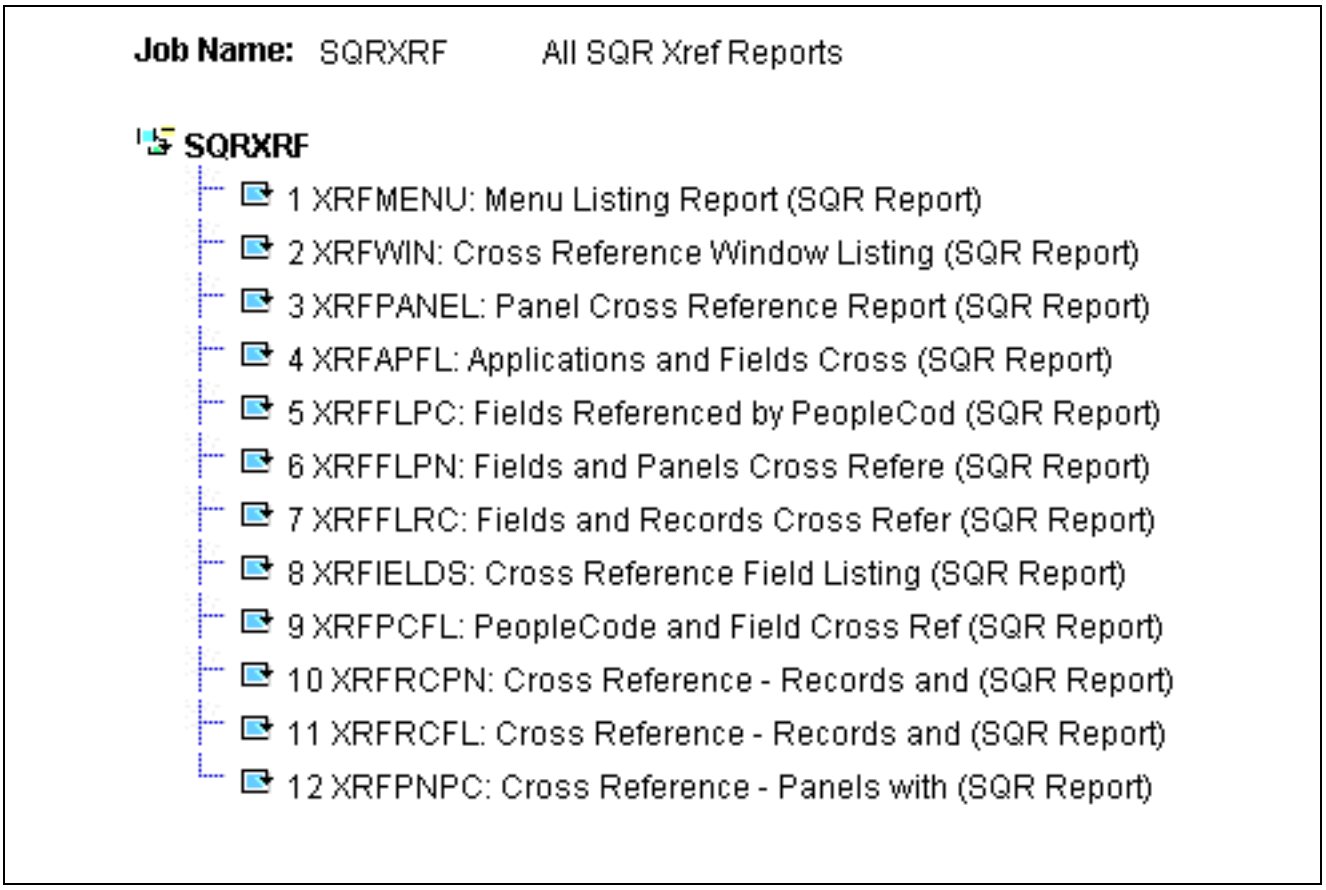
Example

The following code:

```
Local ProcessRequest &JobRQST;
Local string &sHTML;

&JobRQST = CreateProcessRequest("PSJob", "SQRXRF");
&sHTML = &JobRQST.PrintJobHTMLRpt();
```

generates the following HTML file:

 Copyright © 1988-2007, Oracle. All rights reserved.



Example HTML file

See Also

Chapter 29, "ProcessRequest Class," PrintJobRqstRpt, page 1205 and Chapter 29, "ProcessRequest Class," PrintSchdlHTMLRpt, page 1208.

PrintJobRqstRpt

Syntax

```
PrintJobRqstRpt(ProcessInstance, ItemInstance [, PrintJobTree] [, PrintDistList] [,
PrintNotifyList] [, PrintSystemMessage] [, PrintApplicationMessage] [, PrintParamList])
```

Description

Use the PrintJobRqstRpt method to generate an HTML report file displaying the current status of a specific process, job, or jobset.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ProcessInstance	Specify the ProcessInstance number assigned to the process, job, or jobset
ItemInstance	Specify either to display all items in a job or jobset, or just a specific item. • To display all items, specify 0. • To display a specific item, specify the ProcessInstance number assigned to that item. For a process, set this parameter to 0.
PrintJobTree	Specify whether to have the job tree displayed in the HTML report. This parameter takes a string value: • "1" to display the tree. • "0" to not display the tree. Default value is "1".
PrintDistList	Specify whether you want the list of Users and Roles who will be the recipient of a report generated for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the list. • "0" to not display the list. Default value is "0".
PrintSystemMessage	Specify whether you want to have the message specified from the Process Definition or Job Definition page displayed in the HTML report. This parameter takes a string value: • "1" to display the system message. • "0" to not display the system message. Default value is "0".
PrintApplicationMessage	Specify whether you want the application messages displayed in the HTML report. These are the application messages that can be viewed from the Message Log subpage of the Process Monitor Detail page. This parameter takes a string value: • "1" to display application messages. • "0" to not display application messages. Default value is "0".
PrintParamList	Specify whether you want the parameter list for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the parameter list. • "0" to display the parameter list. Default value is "0".

Returns

An HTML report as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

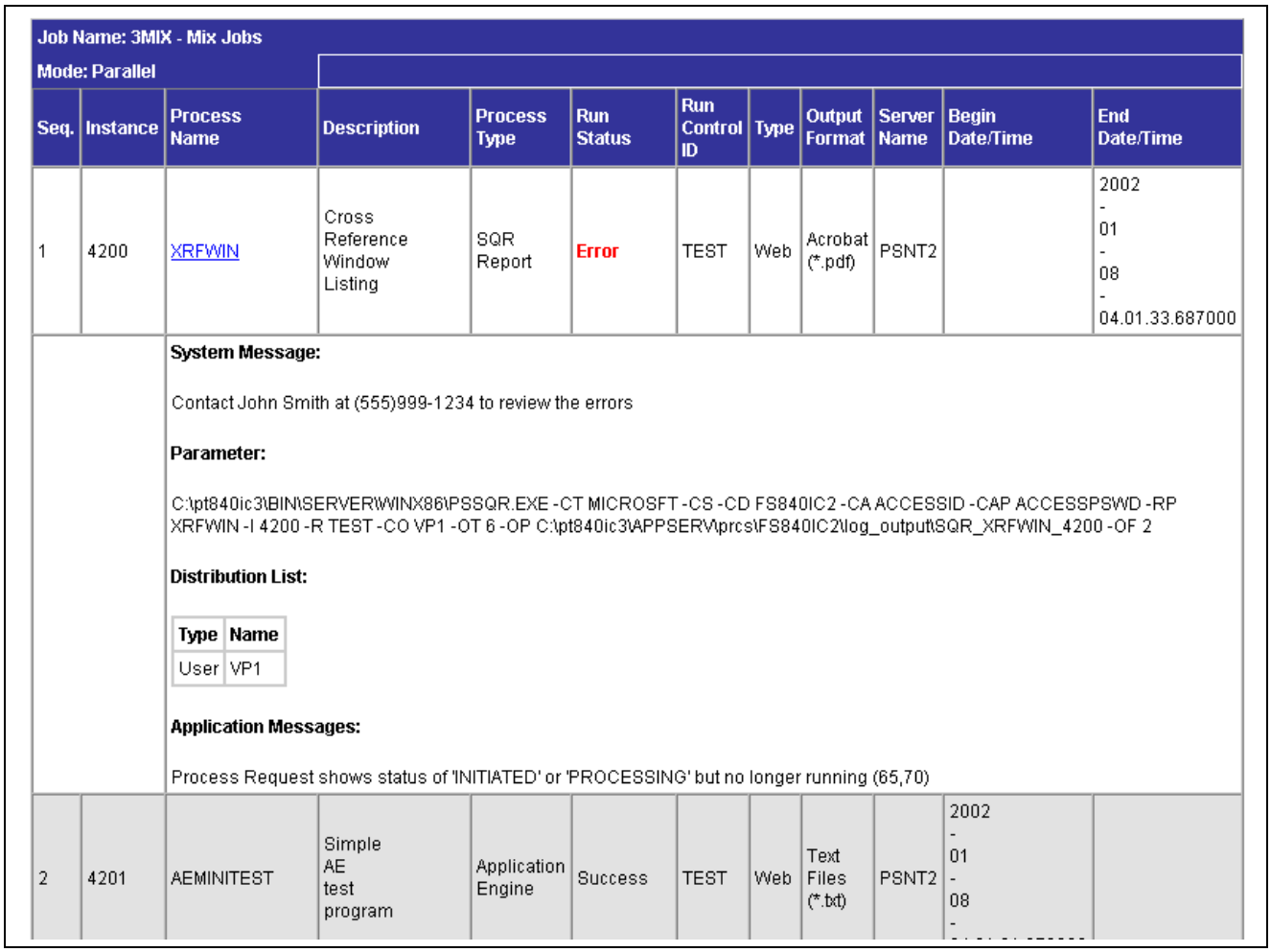
Example

The following PeopleCode program:

```
Local ProcessRequest &JobRQST;
Local string &sHTML;
Local string &sPRINT_JOBTREE;
Local string &sPRINT_DISTLIST;
Local string &sPRINT_SYSMESSAGE;
Local string &sPRINT_APPLMESSAGE;
Local string &sPRINT_PARAMLIST;

&sPRINT_JOBTREE = "0";
&sPRINT_DISTLIST = "1";
&sPRINT_SYSMESSAGE = "1";
&sPRINT_APPLMESSAGE = "1";
&sPRINT_PARAMLIST = "1";
&JobRQST = CreateProcessRequest();
&sHTML = &JobRQST.PrintJobRqstRpt(PMN_PRCSLIST.PRCSINSTANCE, 0, &sPRINT_JOBTREE,⇒
 &sPRINT_DISTLIST, &sPRINT_SYSMESSAGE, &sPRINT_APPLMESSAGE, &sPRINT_PARAMLIST);
```

Creates the following HTML report:

Copyright © 1988-2007, Oracle. All rights reserved.



Job Name: 3MIX - Mix Jobs
Mode: Parallel

Seq.	Instance	Process Name	Description	Process Type	Run Status	Run Control ID	Type	Output Format	Server Name	Begin Date/Time	End Date/Time
1	4200	XRFWIN	Cross Reference Window Listing	SQR Report	Error	TEST	Web	Acrobat (*.pdf)	PSNT2		2002 - 01 - 08 - 04.01.33.687000

System Message:

Contact John Smith at (555)999-1234 to review the errors

Parameter:

C:\pt840ic3\BIN\SERVERWINX86\PSSQR.EXE -CT MICROSFT -CS -CD FS840IC2 -CA ACCESSID -CAP ACCESSPSWD -RP XRFWIN -I 4200 -R TEST -CO VP1 -OT 6 -OP C:\pt840ic3\APPSERV\prcs\FS840IC2\log_output\SQR_XRFWIN_4200 -OF 2

Distribution List:

Type	Name
User	VP1

Application Messages:

Process Request shows status of 'INITIATED' or 'PROCESSING' but no longer running (65,70)

2	4201	AEMINITEST	Simple AE test program	Application Engine	Success	TEST	Web	Text Files (*.txt)	PSNT2	2002 - 01 - 08 -	

Example HTML report

See Also

Chapter 29, "ProcessRequest Class," PrintJobHTMLRpt, page 1204 and Chapter 29, "ProcessRequest Class," PrintSchdlHTMLRpt, page 1208.

PrintSchdlHTMLRpt

Syntax

```
PrintSchdlHTMLRpt([PrintJobTree] [, PrintDistList] [, PrintNotifyList] [,
PrintMessageList] [, PrintParamList])
```

Description

Use the PrintSchdlHTMLRpt method to generate an HTML report file displaying all items in a job or jobset as defined in the Scheduled Jobset Definition component.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PrintJobTree	Specify whether to have the job tree displayed in the HTML report. This parameter takes a string value: • "1" to display the tree. • "0" to not display the tree. Default value is "1".
PrintDistList	Specify whether you want the list of Users and Roles who will be the recipient of a report generated for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the list. • "0" to not display the list. Default value is "0".
PrintNotifyList	Specify whether you want the list of Users and Roles who will be notified for the status of a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the list. • "0" to not display the list. Default value is "0".
PrintMessageList	Specify whether you want the messages that will be emailed upon completion of a job item displayed in the HTML report. This parameter takes a string value: • "1" to display messages. • "0" to not display messages. Default value is "0".
PrintParamList	Specify whether you want the parameter list for a job item displayed in the HTML report. This parameter takes a string value: • "1" to display the parameter list. • "0" to display the parameter list. Default value is "0".

Returns

An HTML report as a string

Example

The following code:

```
Local string &sHTML;
Local string &sPRINT_JOBTREE;
Local string &sPRINT_DISTLIST;
Local string &sPRINT_NOTIFYLIST;
Local string &sPRINT_MESSAGELIST;
Local string &sPRINT_PARAMLIST;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&sPRINT_JOBTREE = "1";
&sPRINT_DISTLIST = "0";
&sPRINT_NOTIFYLIST = "0";
&sPRINT_MESSAGELIST = "0";
&sPRINT_PARAMLIST = "0";
&JobRQST = SetupScheduleDefnItem("Sample", "MULTIJOB");
&sHTML = &JobRQST.PrintSchdlHTMLRpt(&sPRINT_JOBTREE, &sPRINT_DISTLIST, &sPRINT_⇒
NOTIFYLIST, &sPRINT_MESSAGELIST, &sPRINT_PARAMLIST);
```

Creates the following HTML report:

Job Name: MULTIJOB - Jobset Example

Mode: Serial

Seq.	Process Name	Description	Process Type	Run Control ID	Type	Output Format	Destination	Server Option	Run Time
1	AP_APY2015	Pay Cycle Process Request	Application Engine	RunCntlTest	Web	Text Files (*.txt)	Distribution List	Any Server	
2	AP_PSTPYMNT	PS/AP Payment Posting	Application Engine	RunCntlTest	Web	Text Files (*.txt)	Distribution List	Any Server	
3	BI JOBS	BI Multi Jobs	PSJob	RunCntlTest	(None)	(None)		Any Server	

Job Name: BI JOBS - BI Multi Jobs

Mode: Serial **Parent Job Name: MULTIJOB**

Seq.	Process Name	Description	Process Type	Run Control ID	Type	Output Format	Destination	Server Option	Run Time
1	BIJOB01	Finalize and Print	PSJob	RunCntlTest	(None)	(None)		Any Server	
2	BIJOB02	Consolidation Final & Print	PSJob	RunCntlTest	(None)	(None)		Any Server	
3	BIJOB03	Single Action Invoice	PSJob	RunCntlTest	(None)	(None)		Any Server	

Example HTML report

See Also

Chapter 29, "ProcessRequest Class," PrintJobHTMLRpt, page 1204 and Chapter 29, "ProcessRequest Class," PrintJobRqstRpt, page 1205.

 Copyright © 1988-2007, Oracle. All rights reserved.

RunJobSetNow

Syntax

RunJobSetNow()

Description

Use the RunJobSetNow method to schedule a jobset based on settings defined in the Scheduled Jobset Definition component.

Parameters

None.

Returns

None. To verify that the method executed successfully, check the value of the Status property.

Example

```
/* Create the ProcessRequest Object & Run the Scheduled JobSet Now */
Local ProcessRequest &JobRQST;
Local integer &instanceList;

&JobRQST = SetupScheduleDefnItem("Sample","MULTIJOB");
&JobRQST.RunJobSetNow();
&instanceList = &JobRQST.ProcessInstance;
```

See Also

Chapter 29, “ProcessRequest Class,” Status, page 1225

Schedule

Syntax

Schedule()

Description

Use the Schedule method to insert a request in the Process Request table which runs according to the values in the properties of the ProcessRequest object. To successfully schedule a process or job, certain properties are required.

- If you’re scheduling a single process, you must assign values to the following properties:
  - RunControlID
  - ProcessName
  - ProcessType
- If you’re scheduling a job where job item changes are not made, you must assign values to the following properties:
  - RunControlID
  - JobName

Copyright © 1988-2007, Oracle. All rights reserved.

- If you're scheduling a job where job item changes are made, you must assign values to the RunControlID.

Note. You don't have to assign the JobName in this instance because it's set with the CreateProcessRequest function.

Parameters

None.

Returns

None. To verify that the method executed successfully, check the value of the Status property.

Example

```
&MYRQST.Schedule();
If &MYRQST.Status = 0 then
   /* Schedule succeeded. */
Else
   /* Process (job) not scheduled, do error processing */
End-If;
```

See Also

Chapter 29, "ProcessRequest Class," RunControlID, page 1223, Chapter 29, "ProcessRequest Class," JobName, page 1217, Chapter 29, "ProcessRequest Class," ProcessName, page 1221, Chapter 29, "ProcessRequest Class," ProcessType, page 1222 and Chapter 29, "ProcessRequest Class," Status, page 1225.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateProcessRequest

SetEmailOption

Syntax

```
SetEmailOption(EmailSubject, EmailText, EmailAddress, EmailWebReport, EmailAttachLog
[, JobName] [, PrcsItemLevel] [, JobSeqNo])
```

Description

Use the SetEmailOption method to set the email options for all job items in the main job.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
EmailSubject	Specify the text used in the subject of the email sent at completion of this job email subject as a string. You can specify a Null ("") for this parameter.
EmailText	Specify the email text sent at completion of this job as a string. You can specify a Null ("") for this parameter.
EmailAddress	Specify the email address of the person you want an email sent to at the completion of this job. To send more than one email address, separate the addresses with a semicolon.
EmailWebReport	Specify whether to attach the web report to the email sent at the completion of this job. This parameter takes a Boolean value: True, attach the web report, False, don't attach the web report. This parameter can be set only to True when the OutDestType property for the request is "Web".
EmailAttachLog	Specify whether to attach the log file to the email sent at the completion of this job. This parameter takes a Boolean value: True, attach the log file, False, don't attach the log file.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the Process Item Level as a number.

Returns

None.

See Also

Chapter 29, "ProcessRequest Class," SetItemFolder, page 1213, Chapter 29, "ProcessRequest Class," SetOutputOption, page 1214 and Chapter 29, "ProcessRequest Class," AddDistributionOption, page 1203.

SetItemFolder

Syntax

```
SetItemFolder(PortalFolder [, JobName] [, PrcsItemLevel] [, JobSeqNo])
```

Description

Use the SetItemFolder method to associate a report folder with any job items jobset.

The folder that you specify must have already been created using the System Settings component in Process Scheduler Manager.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PortalFolder	Specify the name of the report folder to use for the job item as a string. The folder that you specify must have already been created using Report Manager.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the Process Item Level as a number.

Returns

None.

Example

```
&RQST.SetItemFolder("GENERAL", "MULTIJOB", 1, 2);
```

See Also

Chapter 29, "ProcessRequest Class," AddDistributionOption, page 1203

SetOutputOption

Syntax

SetOutputOption(*OutputType*, *OutputFormat*, *OutputDest* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])

Description

Use the SetOutputOption method to modify the output option of one or more job items in the main job.

Note. This method can also be used for a single process.

The job sequence number and job level are optional. They are required only when the same process name is referenced more than once in a job and the user intends to modify the output option of only one item.

If the ProcessRequest object contains an invalid output type for a process, you won't be able to successfully schedule a process or job.

The values for *OutputType*, *OutDestFormat,* and *OutputDestination* are dependent upon each other as well as on other values.

See Chapter 29, "ProcessRequest Class," Values for Output Type and Format, page 1198.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
OutputType	Specify the output type as a string.
OutputFormat	Specify the output format as a string.
OutputDestination	Specify the output destination as a string.
JobName	Specify the name of the job that this item belongs to as a string.
PrcsItemLevel	Specify the job item's process item level within the main job as a number.
JobSeqNo	Specify the job item's job sequence number within the Process Item Level as a number.

Returns

None.

Example

```
&RQST.SetOutputOption("Email", "HTM", "", "MULTIJOB", 1, 2);
```

See Also

Chapter 29, "ProcessRequest Class," OutDest, page 1219, Chapter 29, "ProcessRequest Class," OutDestFormat, page 1220 and Chapter 29, "ProcessRequest Class," OutDestType, page 1220.

UpdateRunStatus

Syntax

```
UpdateRunStatus()
```

Description

Use the UpdateRunStatus method to change the RunStatus for a specific process, job, or jobset.

Parameters

None.

Returns

None.

Example

```
Local ProcessRequest &RQST;

&RQST = CreateProcessRequest();
&RQST.ProcessInstance = 5;
&RQST.RunStatus = 1; /* Cancel the request */
&RQST.UpdateRunSttus();
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 29, "ProcessRequest Class," RunStatus, page 1224

ProcessRequest Class Properties

In this section, we discuss the ProcessRequest class properties. The properties are discussed in alphabetical order.

EmailAttachLog

Description

This property specifies whether or not a log file is attached to the email sent at completion of this job (or process). This property takes a Boolean value.

This property is read-write.

Example

```
&RQST.EmailAttachLog = False;   /*  Do not attach Log File */
```

EmailSubject

Description

This property specifies the text used in the subject of the email sent at completion of this job (or process). This property takes a string value.

This property is read-write.

Example

```
&RQST.EmailSubject = "SQR Report:  Cross Reference Listing";
```

EmailText

Description

This property specifies the text of the email sent at the completion of this job (or process). This property takes a string value.

This property is read-write.

Example

```
&RQST.EmailText = "This text will be displayed as the text of this email ";
```

You can also use the text from a message in the message catalog for this property.

```
&RQST.EmailText = MsgGetText(65, 117, "Sample text for email with two parameters",⇒
 &MessageParm1, &MessageParm2);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

EmailWebReport

Description

This property specifies whether an email is sent to recipients of the report after it is posted to the report repository. The URL for the report is included in the email. This option is applicable only when the OutDesType for the request is Web. This property takes a Boolean value: True, the web report should be attached, False otherwise.

This property is read-write.

FileName

Description

This property contains the name of a file to be used with file dependant processing, that is, processes which are initiated only after a file becomes available.

For a file dependent process, the FileName property is set to override the file name specified in the process definition. You must specify the complete file path and extension for the file, as a string.

This property works with the RunLocation property. If you don't specify a run location, any value assigned to this property is ignored.

This property is read-write.

Example

The following code example initiates QE_AETESTPRG once the c:\import\ediData.dat file becomes available on PSNT server.

```
Local ProcessRequest &rqst1;

&rqst1 = CreateProcessRequest();
&rqst1.RunControlId = "2";
&rqst1.ProcessType = "Application Engine";
&rqst1.ProcessName = "QE_AETESTPRG";
&rqst1.RunLocation = "PSNT";
&rqst1.FileName="c:\import\ediData.dat";
&rqst1.Schedule();
```

See Also

Chapter 29, "ProcessRequest Class," RunLocation, page 1223

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Defining PeopleSoft Process Scheduler Support Information," Setting Process Definition Options

JobName

Description

This property contains the name of a job that you've defined in PeopleSoft Process Scheduler.

To schedule a job, you must assign values to JobName and RunControlID for the Schedule method to succeed.

Copyright © 1988-2007, Oracle. All rights reserved.

If you're scheduling a job, you don't need to set the ProcessType property.

This property is read-write.

Example

```
&MYRQST.JobName = "3SQR";
```

See Also

Chapter 29, "ProcessRequest Class," RunControlID, page 1223

LanguageCd

Description

This property enables you to specify a language code for the process or job. If you don't specify a language code, PeopleSoft Process Scheduler first looks in the process run control table to retrieve a language code. If there isn't a value there, PeopleSoft Process Scheduler uses the user's language code specified in the User Definition table.

This property takes a string value.

This property is read-write.

Example

```
&MYRQST.LanguageCd = "ESP" /* Spanish */
```

NotifyTextMsgNum

Description

Use the NotifyTextMsgNum property to specify the message that should be displayed to the end user in the Event Notification popup display. This property takes a numeric value.

Note. You must specify both the message set and message number for the message to be displayed to the end user.

This property is read-write.

Example

```
&RQST.NotifyTextMsgSet = 65;
&RQST.NotifyTextMsgNum = 237;
```

See Also

Chapter 29, "ProcessRequest Class," NotifyTextMsgSet, page 1219

 Copyright © 1988-2007, Oracle. All rights reserved.

NotifyTextMsgSet

Description

Use this property to specify the message set of the text message to be displayed to the end user in the Event Notification popup display. This property takes a numeric value.

Note. You must specify both the message set and message number for the message to be displayed to the end user.

This property is read-write.

Example

```
&RQST.NotifyTextMsgSet = 65;
&RQST.NotifyTextMsgNum = 237;
```

See Also

Chapter 29, "ProcessRequest Class," NotifyTextMsgNum, page 1218

OutDest

Description

This property specifies the output destination for the process or job to be scheduled as a string.

Values depend on the setting for the OutDestType property:

- If OutDestType is FILE, OutDest must be the name of a directory. If the OutDest property isn't set, it defaults to the setting in the Process profile. If the Process Profile was not updated with a default destination, the Process Scheduler server that picked up the request sets the directory based on the Log/Output Directory setting found in the Process Scheduler Configuration file.
- If OutDestType is PRINTER, OutDest must be the name of a printer. If the OutDest property isn't set, it defaults to the setting in the Process profile. If the Process Profile was not updated with a default printer, the Process Scheduler server that picked up the request sets the printer based on the Default Printer setting found in the Process Scheduler Configuration file.
- If OutDestType is WEB or EMAIL, OutDest should contain the list of User IDs, Role IDs, or email addresses (for email only). If you don't set this property, it's automatically assigned to the person who submitted the request.
- If OutDestType is WEB, PeopleSoft Process Scheduler uses the OutDest property to determine who has access to the output.
- If OutDestType is EMAIL, PeopleSoft Process Scheduler uses the OutDest property to determine who to send the report output to.

  In both these cases, to identify whether it's a User ID or a Role ID, the value has to be preceded by one of the following:

  U: *Username* or **User:** *Username* for a UserID

  R: *Rolename* or **Role:** *Rolename* for a Role ID

  If the ID is not preceded by either of these identifiers, Process Scheduler assumes it's an email address.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. Each ID must be separated by a semi-colon (;).

The following are two examples:

```
&RQST.OutDest = "U:PTDMO;R:Employee;robert_smith@peoplesoft.com"

&RQST.OutDest = "User:PS;Role:Employee;sue_line@XYZ.com;simon_gree@peoplesoft.com"
```

- For any other value of OutDestType, this property has no effect.

You can specify directory and printer names using the UNC (Uniform Naming Convention) protocol.

This property is read-write.

Example

```
&MYRQST.OutDest = "C:\TEMP";
```

See Also

Chapter 29, "ProcessRequest Class," OutDestType, page 1220

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Managing PeopleSoft Process Scheduler"

OutDestFormat

Description

This property specifies the output format for the process or job to be scheduled as a string. Values depend on your settings for the ProcessType and OutDestType properties:

The values for OutDestType, OutDestFormat, and OutDest are dependent upon each other as well as on other values.

See Chapter 29, "ProcessRequest Class," Values for Output Type and Format, page 1198.

This property is read-write.

Example

```
&MYRQST.OutDestFormat = "RTF";
```

See Also

Chapter 29, "ProcessRequest Class," ProcessType, page 1222 and Chapter 29, "ProcessRequest Class," OutDestType, page 1220.

OutDestType

Description

This property specifies the type of output for the process or job to be scheduled as a string.

If the ProcessRequest object contains an invalid output type for a process, you won't be able to able to successfully schedule a process or job.

 Copyright © 1988-2007, Oracle. All rights reserved.

The value specified for this property is used only if the Output Destination Type for the defined process or job defined in PeopleSoft Process Scheduler is specified as Any. Otherwise any value specified for this property is ignored.

The values for OutDestType, OutDestFormat, and OutDest are dependent upon each other as well as on other values.

See Chapter 29, "ProcessRequest Class," Values for Output Type and Format, page 1198.

This property is read-write.

Example

```
&MYRQST.OutDestType = "FILE";
```

See Also

Chapter 29, "ProcessRequest Class," ProcessType, page 1222 and Chapter 29, "ProcessRequest Class," RunControlID, page 1223.

PortalFolder

Description

This property specifies the name of the report folder associated with a job item as a string. If you're specifying a folder, the folder that you specify must have already been created using Report Manager.

This property is read-write.

ProcessInstance

Description

This property is a system-generated identification number. PeopleSoft Process Scheduler assigns a ProcessInstance at runtime to each process or job it successfully schedules.

This property is read-write.

Example

```
&MYRQST.Schedule();
If &MYRQST.Status = 0 then
   /* process successfully scheduled */
   &ProcInst = &MYRQST.ProcessInstance;
Else
   /* do error processing */
End-If;
```

ProcessName

Description

This property specifies the name of a predefined process as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

To successfully schedule a process, you must assign values to ProcessName, ProcessType, and RunControlID (that is, for the Schedule method to succeed.)

This property is read-write.

Example

```
&MYRQST.ProcessName = "XRFWIN";
```

See Also

Chapter 29, “ProcessRequest Class,” ProcessType, page 1222 and Chapter 29, “ProcessRequest Class,” RunControlID, page 1223.

ProcessType

Description

This property specifies the name of a predefined process type as a string.

To successfully schedule a process, you must assign values to ProcessName, ProcessType, and RunControlID (that is, for the Schedule method to succeed.)

The values for ProcessType depend on the types of processes you have defined in your system. There are generic process types that are delivered with your installation of PeopleSoft. These process types may include the following:

- Application Engine
- COBOL
- Crystal
- Cube
- nVision
- SQR
- WinWord
- Other

If you define your own processes, you can use the name of that process with the ProcessType property. For example, suppose you create a custom process named "Custom CBL Programs." You could use this as follows:

```
&MyRqst.ProcessType = "Custom CBL Programs";
```

Note that spaces are included in the string for ProcessType.

This property is read-write.

Example

Note that spaces are included in the string for ProcessType.

```
&MYRQST.ProcessType = "Application Engine";
```

See Also

Chapter 29, “ProcessRequest Class,” ProcessName, page 1221 and Chapter 29, “ProcessRequest Class,” RunControlID, page 1223.

 Copyright © 1988-2007, Oracle. All rights reserved.

RunControlID

Description

This property returns a string that serves, along with the UserID, as a key that identifies a predefined group of parameters to be used by a process or a job at runtime.

To successfully schedule a process, you must provide values for RunControlID, ProcessName, and ProcessType.

To successfully schedule a job, you must provide values for RunControlID and JobName.

This property is read-write.

Example

```
&MYRQST.RunControlID = "MYRUNCONTROLID";
or
&MYRQST.RunControlID = PRCSSAMPLEREC.RUN_CNTL_ID;
```

See Also

Chapter 29, "ProcessRequest Class," JobName, page 1217, Chapter 29, "ProcessRequest Class," ProcessName, page 1221 and Chapter 29, "ProcessRequest Class," ProcessType, page 1222.

RunDateTime

Description

This property contains a DateTime value that specifies when the scheduled process or job will run.

If you don't specify a value for this property, and there is no date time set for the pre-defined process or job, the process or job runs as soon as the Schedule method is executed.

This property is read-write.

Example

The following example schedules the process or job to run as soon as the Schedule method is executed:

```
&MYRQST.RunDateTime = %Datetime;
```

RunLocation

Description

This property specifies the Process Scheduler Server name the request should be scheduled on. This property takes a string value. Values for RunLocation is a specific server name, such as PSNT.

If no RunLocation is specified, the request is scheduled based on both the default operating system and load balancing options set in the System Settings page.

This property is read-write.

Example

```
&MYRQST.RunLocation = "SERVER";
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
or
&MYRQST.RunLocation = "PSNT";
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Using Process Monitor," Viewing the Status of Processes

RunRecurrence

Description

This property specifies the frequency with which a process or job is to be run as a string. The RunRecurrence value you use must be the name of a Recurrence Definition defined in Process Scheduler Manager to successfully schedule a job or process (that is, for the Schedule method to succeed.)

This property is read-write.

Example

```
&MYRQST.RunRecurrence = "M-F at 5pm";
```

RunStatus

Description

This property specifies the run status of a process request as a number. This property is used with the UpdateRunStatus method to change the status of a request. Values are:

RunStatus	Description
1	Cancel
2	Delete
3	Error
4	Hold
5	Queued
6	Initiated
7	Processing
8	Cancelled

 Copyright © 1988-2007, Oracle. All rights reserved.

RunStatus	Description
9	Success
10	Not Successful
11	Posted
12	Unable to post
13	Resend
14	Posting
15	Generated
16	Pending

This property is read-write.

Example

```
&MYRQST.RunStatus = 1;
```

See Also

Chapter 29, “ProcessRequest Class,” UpdateRunStatus, page 1215

Status

Description

This property returns a number based on the result of the last execution of the Schedule method.

Valid returns are:

- Zero if the method succeeded
- Non-zero otherwise

This property is read-only.

Example

```
&MYRQST.Schedule();
If &MYRQST.Status = 0 then
   /* Schedule succeeded. */
Else
   /* Process (job) not scheduled, do error processing */
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

TimeZone

Description

This property contains a timezone value that specifies when the scheduled process or job will run. If no value is used for this property, the server timezone is used. This property takes a string value.

This property is read-write.

Example

```
&MyRqst.TimeZone = "EST";
```

ProcessRequest Class Examples

The following section contains the following examples:

- Scheduling a single process
- Scheduling a job where job item changes are not made
- Scheduling a job where job item changes are made

Scheduling a Single Process

The following example is when you're scheduling a single process:

```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest();

/* Set all the Required Properties */
&RQST.RunControlID = &sRunCntlId;
&RQST.ProcessType = &sProcessType;
&RQST.ProcessName = &sProcessName;

/* Set any Optional Properties for this Process */
&RQST.RunLocation = &sRunLocation;
&RQST.RunDateTime = &dttmRunDateTime;
&RQST.TimeZone = &sTimeZone;
&RQST.PortalFolder = &sPortalFolder;
&RQST.RunRecurrence = &sRecurrence;
&RQST.OutDestType = &sOutDestType;
&RQST.OutDestFormat = &sOutDestFormat;
&RQST.OutDest = &sOutputDirectory;
&RQST.EmailAttachLog = &bEmailAttachLog;
&RQST.EmailWebReport = &bEmailWebReport;
&RQST.EmailSubject = &sEmailSubject;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&RQST.EmailText = &sEmailText;

/* Schedule the Process */
&RQST.Schedule();
```

Scheduling a Job Where Job Item Changes Are Not Made

The following example is when job item changes aren't made.

```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest();

/* Set all the Required Properties */
&RQST.RunControlID = &sRunCntlId;
&RQST.JobName = &sJobName;

/* Set any Optional Main Job Properties */
&RQST.RunLocation = &sRunLocation;
&RQST.RunDateTime = &dttmRunDateTime;
&RQST.TimeZone = &sTimeZone;
&RQST.PortalFolder = &sPortalFolder;
&RQST.RunRecurrence = &sRecurrence;
&RQST.OutDestType = &sOutDestType;
&RQST.OutDestFormat = &sOutDestFormat;
&RQST.OutDest = &sOutputDirectory;
&RQST.EmailAttachLog = &bEmailAttachLog;
&RQST.EmailWebReport = &bEmailWebReport;
&RQST.EmailSubject = &sEmailSubject;
&RQST.EmailText = &sEmailText;

/* Schedule the Job */
&RQST.Schedule();
```

Scheduling a Job Where Job Item Changes Are Made

The following example is when job item changes are made.

```
Local ProcessRequest &RQST;

/* Create the ProcessRequest Object */
&RQST = CreateProcessRequest("PSJob", &sJobName);

/* Set all the Required Properties */
/* Note: the JobName Property has already */
/* been set in the call above, so you don&rsquo;t */
/* have to set it here */
&RQST.RunControlID = &sRunCntlId;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Set any Optional Job Item Properties */
&RQST.SetOutputOption(&sOutputType, &sOutputFormat, &sOutputDest, &sJobName, &n⇒
PrcsItemLevel, &nJobSeqNo);

&RQST.SetEmailOption(&sEmailSubject, &sEmailText, &sEmailAddress, &bEmailWeb⇒
Report, &bEmailAttachLog, &sJobName, &nPrcsItemLevel, &nJobSeqNo);
&RQST.SetItemFolder(&sPortalFolder, &sJobName, &nPrcsItemLevel, &nJobSeqNo);
&RQST.AddDistributionOption(&sDistIdType, &sDistId, &sJobName, &nPrcsItemLevel, &n⇒
JobSeqNo);

/* Set any additional Optional Main Job Properties */
&RQST.RunLocation = &sRunLocation;
&RQST.RunDateTime = &dttmRunDateTime;
&RQST.TimeZone = &sTimeZone;
&RQST.PortalFolder = &sPortalFolder;
&RQST.RunRecurrence = &sRecurrence;
&RQST.OutDestType = &sOutDestType;
&RQST.OutDestFormat = &sOutDestFormat;
&RQST.OutDest = &sOutputDirectory;
&RQST.EmailAttachLog = &bEmailAttachLog;
&RQST.EmailWebReport = &bEmailWebReport;
&RQST.EmailSubject = &sEmailSubject;
&RQST.EmailText = &sEmailText;

/* Schedule the Job */
&RQST.Schedule();
```

PrcsApi Class

Use the PrcsApi class with file dependant processing. This section includes:

- Scope of a PrcsApi object.
- Data type of PrcsApi object.
- How to import a PrcsApi object
- How to create a PrcsApi object
- PrcsApi reference

See Chapter 29, “ProcessRequest Class,” File Dependant Processing, page 1202.

Scope of a PrcsApi Object

A PrcsApi object can be instantiated only from PeopleCode.

 Copyright © 1988-2007, Oracle. All rights reserved.

A PrcsApi object can be called from a component, an internet script, or an Application Engine program.

PrcsApi class objects can be of Local, Global, or Component scope.

Data Type of a PrcsApi Object

PrcsApi objects are of type PrcsApi.

```
Local PrcsApi &api = create PrcsApi();
```

How to Import the PrcsApi Class

The PrcsApi class is *not* a built-in class, like Rowset, Field, Record, and so on. It is an Application Class. Before you can use this class in your PeopleCode program, you must import it to your program.

An import statement names either all the classes in a package or one particular application class. For importing the PrcsApi class, PeopleSoft recommends that you import the API class.

The import statement you should use is as follows:

```
import PT_PRCS:API:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available.

See Also

Chapter 6, “Application Classes,” page 153

How to Create a PrcsApi Object

After you’ve imported the PrcsApi class, you instantiate an object of that class using the constructor for the class and the *Create* function.

The following example creates a new instance of the PrcsApi class, as the variable &api, with local scope:

```
import PT_PRCS:API:*;

Local PrcsApi &api = create PrcsApi();
```

Copyright © 1988-2007, Oracle. All rights reserved.

PrcsApi Class Constructor

You must use the constructor for the PrcsApi class to instantiate an instance of that class. The following is the constructor for the PrcsApi class.

PrcsApi

Syntax

```
PrcsApi()
```

Description

Use the PrcsApi constructor to create an instance of the PrcsApi class.

Parameters

None.

Returns

A reference to a PrcsApi object.

PrcsApi Class Methods

In this section, we discuss the PrcsApi class methods. The methods are discussed in alphabetical order.

getAllFileNames

Syntax

```
getAllFileNames(PrcsInstance)
```

Description

Use the getAllFileNames method to return a list of all the files names detected by the scheduler to intiate this process. The names are returned in an array of string, as full path names.

Parameters

Parameter	Description
PrcsInstance	Specify the process instance for which you want to get all the associated file names, as a number.

Returns

An array of string containing all the files names associated with the specified process.

Example

The following example returns a list of file names.

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/* QE_OPT_AET is the state record for the application engine program in this⇒
 example */

import PT_PRCS:API:*;

Local PrcsApi &api = create PrcsApi();

Local File &FileIO, &FileLog;

Local array of string &strList = &api.getAllFileNames(QE_OPT_AET.PROCESS_⇒
INSTANCE);
/* api call needs process instance as parameter */

/* the api returns list of file names matched by the scheduler for this instance */
For &i = 1 To &strList.Len
   &IOFilename = &strList [&i];

   &FileIO = GetFile(&IOFilename, "r", "a", %FilePath_Absolute);

   &FileIO.Delete();

   &FileIO.Close();
End-For;
```

notifyToWindow

Syntax

notifyToWindow(*PrcsInstance*, *Message*)

Description

Use the notifyToWindow method to display additional messages to the window.

If a process is not defined as going to Window, using this method has no effect. The purpose of this method is to display additional messages to the window when an Application Engine process (no other process type) is run to Window.

Parameters

Parameter	Description
PrcsInstance	Specify the process instance for which you want to display messages, as a number.
Message	Specify the message you want displayed, as a string.

Returns

A number: 0 if method failed.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following PeopleCode would be in a step in an application engine program.

```
import PT_PRCS:API:*;

Local PrcsApi &api = create PrcsApi();

/* QE_AESTATUS_AET is the sate record for this Application Engine program */

&nret = &api.notifyToWindow(QE_AESTATUS_AET.PROCESS_INSTANCE, "Hi There! this is⇒
 First step.");

/* this displays the message on the window */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 30

Query Classes

This chapter provides an overview of Query classes and discusses each class in detail.

Understanding Query Classes

You create queries using PeopleSoft Query Manager to extract the data you need from the PeopleSoft database. You can use the Query classes in PeopleCode to create a new query, or to modify or delete an existing query. You can also use methods in the Query class to execute the query and have the result set returned as either a rowset or have it format and write the result set to a file. You can also use the Query classes to create a SQL statement to be used with the SQL object. In fact, the Query classes expose all of the attributes and methods needed by the PeopleSoft Query Manager and Query Viewer applications.

Creating or deleting a query object does not create or delete query information. You must call the appropriate method for that query object directly to create or delete database information, that is, the Create or Delete method.

All of the classes, and most of the properties and methods that make up the Query Classes, have a GUI representation in PeopleSoft Query Manager. This document assumes that the reader has working knowledge of PeopleSoft Query Manager.

There aren't any external built-in functions for the Query classes: objects are instantiated from other objects or from a session object.

See Also

Chapter 36, "Session Class," page 1511

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query, "Getting Started with PeopleSoft Query"

Collections in the Query Classes

A *collection* is a set of similar things, like a group of already existing queries or QueryRecords. As with everything else in the query classes, collections have a GUI representation in PeopleSoft Query.

For example, when you want to open a query, the search page returns a list of all the available queries. This is equivalent to using the FindQueries session class method to get a Query collection.

The following collections are part of the Query classes:

- QuerySelect collection
- QueryDBRecord collection

Copyright © 1988-2007, Oracle. All rights reserved.

- QueryDBRecordField collection
- Query Collection
- QueryRecord collection
- QueryOutputField collection
- QuerySelectedField collection
- QueryCriteria collection
- QueryExpression collection
- QueryPrompt collection

Life-Cycle of a Query

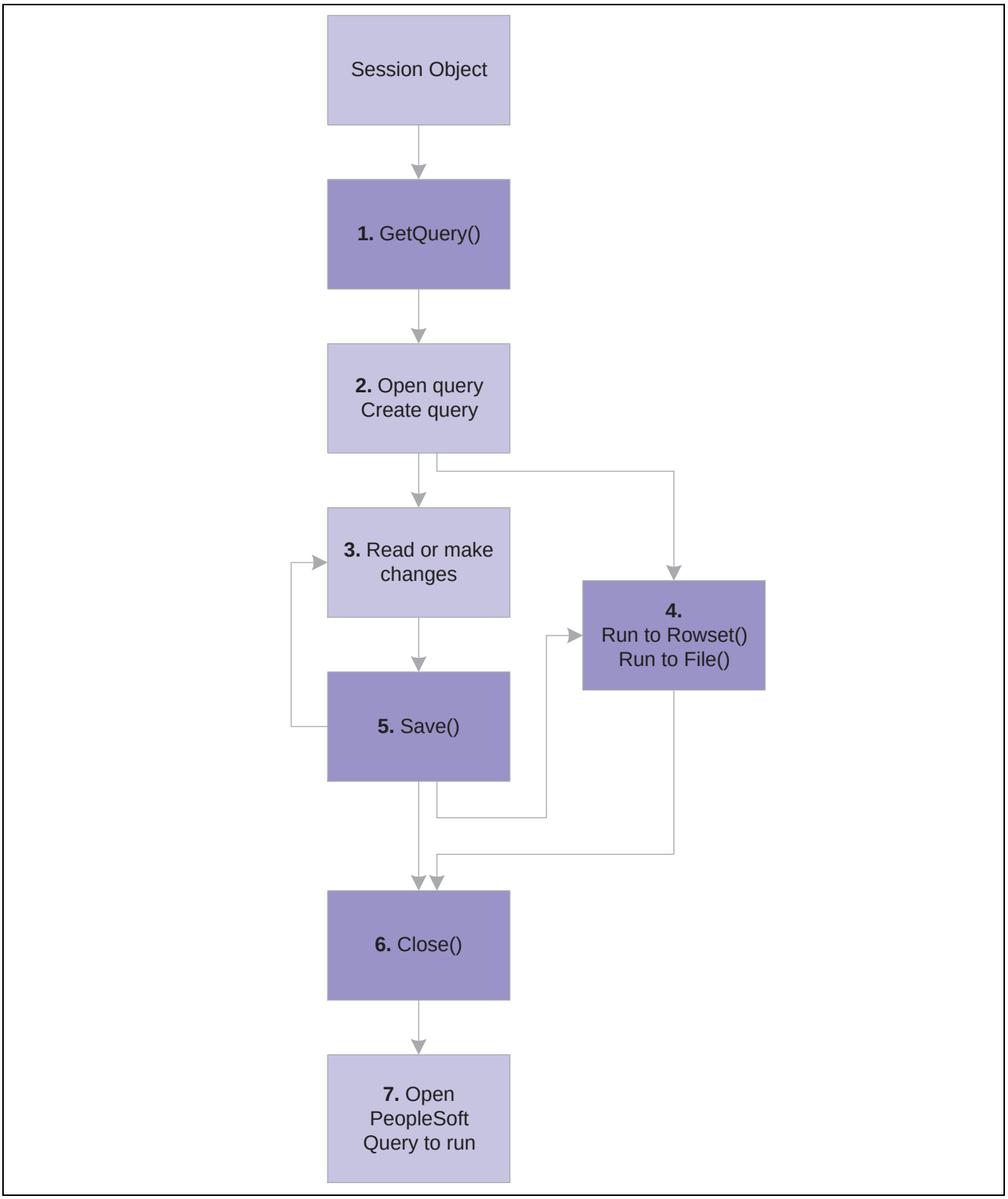
At runtime, there are certain things you want to do with a query, like creating one from scratch, updating the criteria for an existing query, running a query, and so on. The following is an overview of this process, and assumes the most common method to use the Query API. These steps are expanded in other sections.

1. Invoke the GetQuery method on the PeopleSoft session object to get a query.
2. Either open the specific query you want using Open, or create a new query using Create.
3. Read the query statistics, or make changes as appropriate, adding records, field, criteria, and so on.
4. (Optional) Use the RunToRowset method to run the query. You can also use the RunToFile method to execute a scheduled query.

   Note. Be careful whenever you write a PeopleCode program that uses the RunToRowset method because this method could potentially return a large amount of data that could potentially exceed the memory available. For this reason, RunToRowset should be used only when you know that the query being executed returns a reasonable amount of data, or be sure to use the MaxRows parameter to control the maximum amount of data that can be returned.

5. Save the changes.
6. Close the query.
7. If you want, you can navigate to PeopleSoft Query Manager, Query Viewer or the Query Designer to run the query.

 Copyright © 1988-2007, Oracle. All rights reserved.

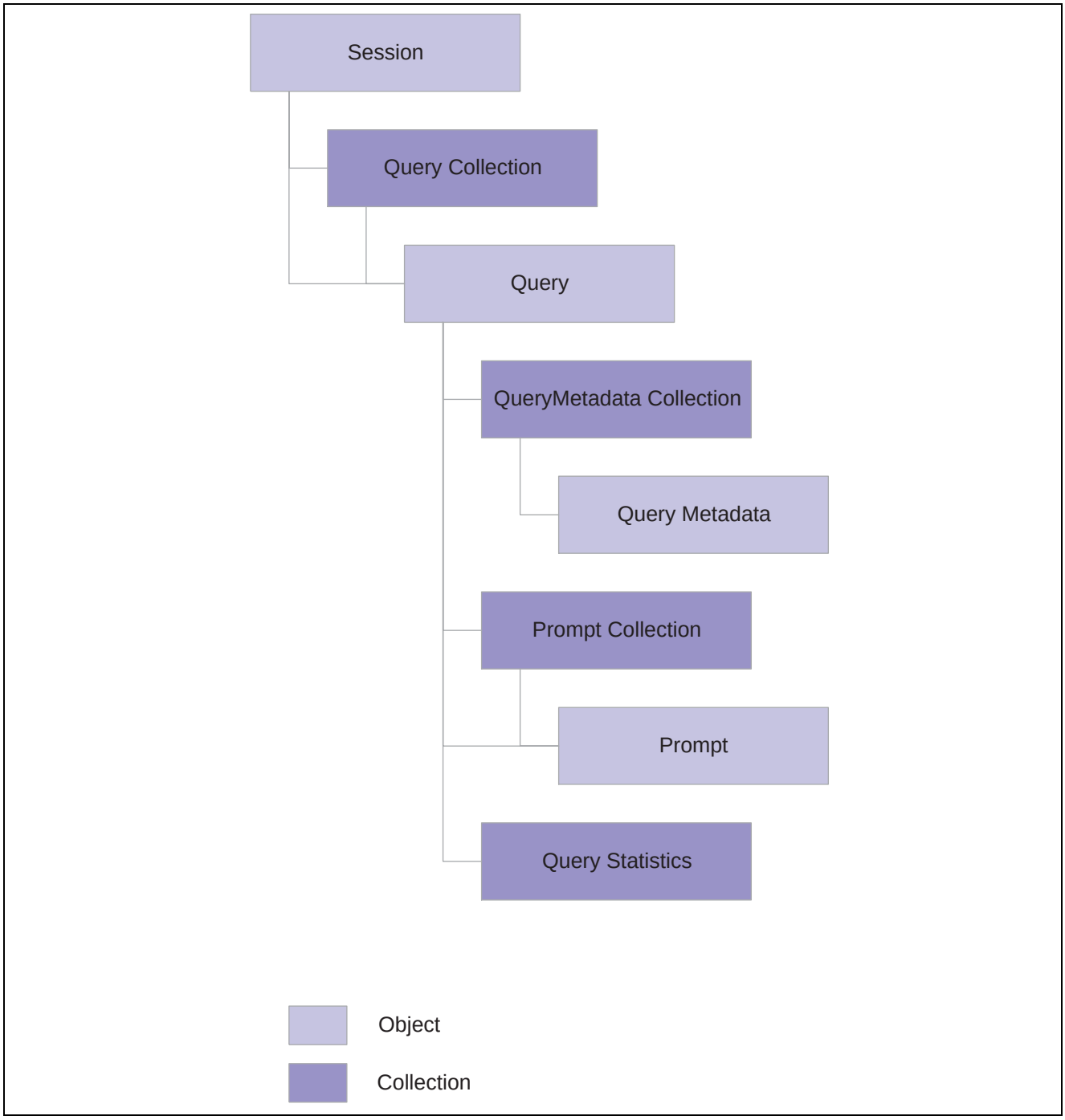


Life cycle of Query object

Copyright © 1988-2007, Oracle. All rights reserved.

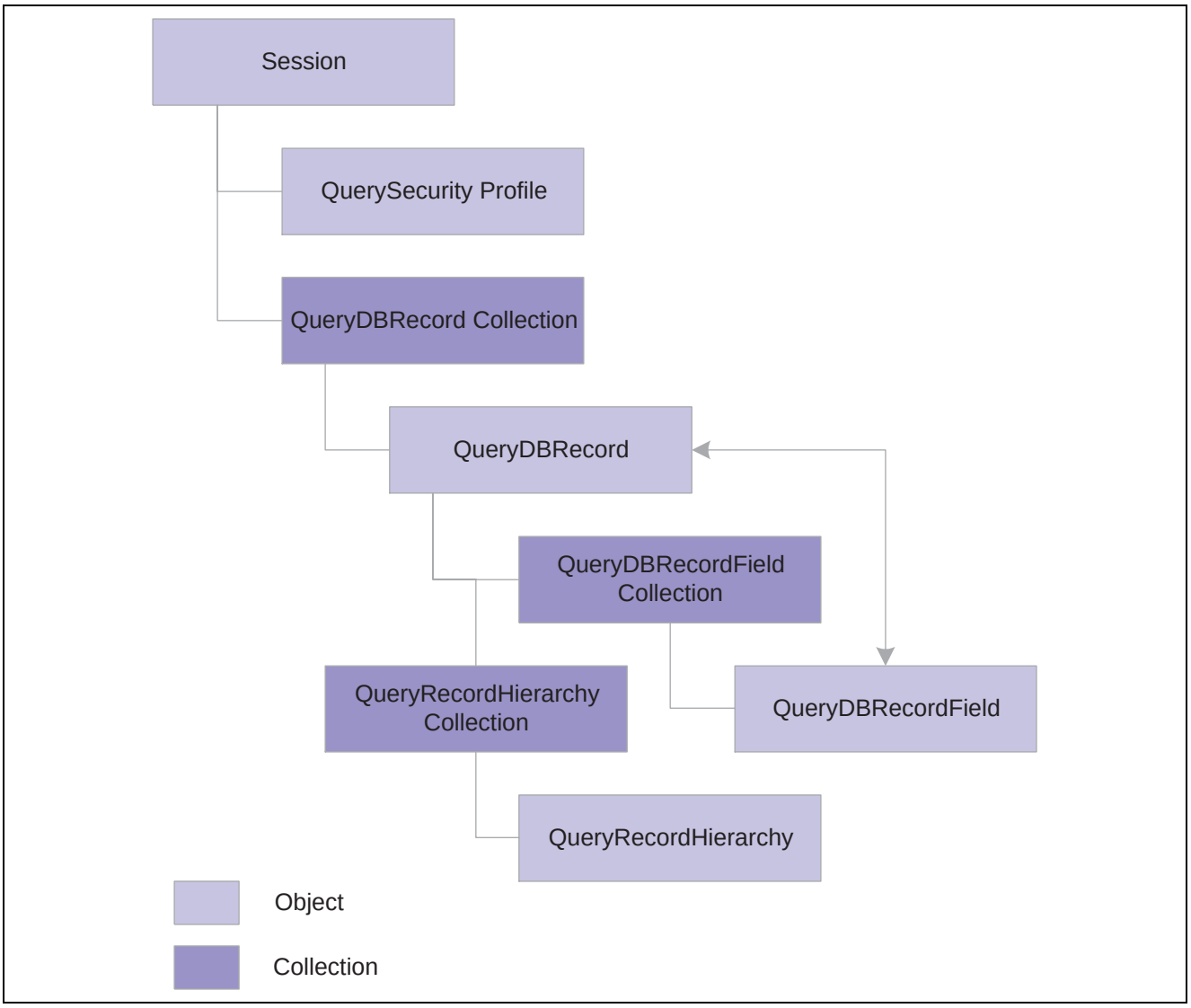
Query Class Hierarchy

There are many different classes used with the Query API. The following flowcharts illustrate all the different classes and how they're interrelated.



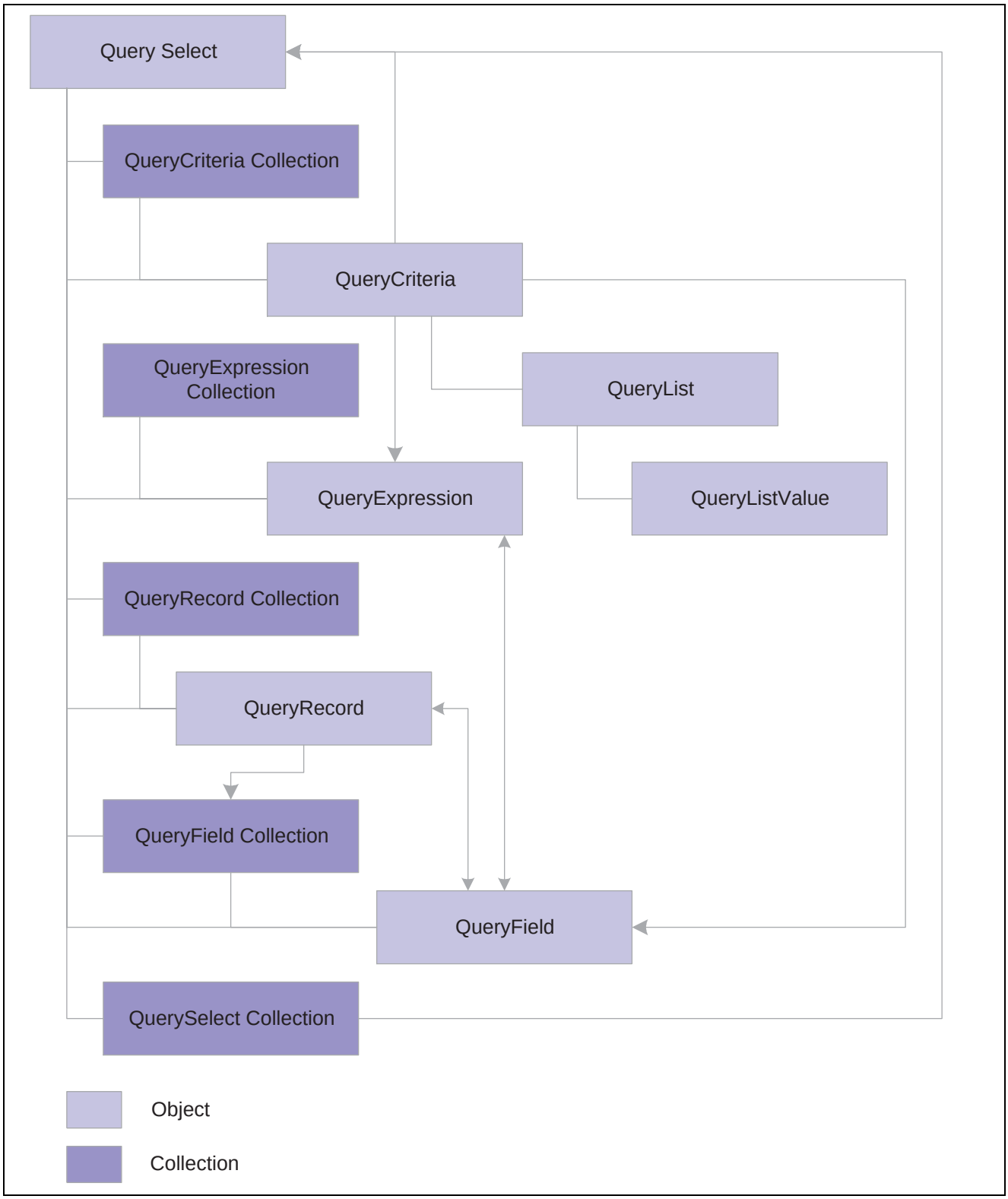
Query API class hierarchy (part 1 of 3)

Copyright © 1988-2007, Oracle. All rights reserved.



Query API class hierarchy (part 2 of 3)

Copyright © 1988-2007, Oracle. All rights reserved.



Query API class hierarchy (part 3 of 3)

Copyright © 1988-2007, Oracle. All rights reserved.

Query Classes Overview

The Query API is made up of many classes. The following are the primary parts, generally used when updating a query:

Query	The query definition.
QuerySelect	A query SELECT statement. There can be multiple QuerySelect objects for Queries that involve unions or subqueries. Each select (or union or subquery) consists of QueryRecords, QueryOutputFields, QuerySelectedFields, and QueryCriteria.
QueryRecords	The records that are part of the existing QuerySelect definition.
QueryOutputFields	The fields that you've selected to be displayed when the results of the query are run.
QuerySelectedFields	All the fields that make up the QuerySelect Definition. These include the fields selected as output fields, and fields added as part of Query Expressions.
QueryCriteria	The criteria for the query.
QueryExpression	The Query expressions that can contain SQL functions or other SQL fragments.
QueryList	The lists used in the in-list functionality for criteria.
QueryPrompt	The prompts used in criteria.
QueryDBRecords and QueryDBRecordFields	All the records available to be used as QueryRecords, and all the fields available. This list is restricted based on a user's Query Access Security Groups.

Query

A database query. You must get a query before you can access any of the other Query classes. You can then create a new query and save it to the database, or open an existing query and modify it.

QuerySelect

Each query is composed of one or more select statements:

Main Select	The instance of the first SELECT statement of the Query is the Main Select. There can only be one Main Select statement in the Query. This instance consists of the QueryOutputFIelds, QueryCriteria, and the QueryExpressions for the Main Select statement.
Union	In addition to the Main Select, a query can have one or more unions. The Unions are added by using the AddUnion method.
Sub-Select	A subquery used in the criteria of the Main Select or in a Union.

QueryRecords

The records that are part of an existing query definition. In PeopleSoft Query, these are the records listed on the query tab. Each QuerySelect has its own set of QueryRecords.

Copyright © 1988-2007, Oracle. All rights reserved.

QueryOutputFields

The fields that you've selected to be part of the query definition. They're called output fields because when you run the query, these are the fields that make up the output columns.

In PeopleSoft Query, these are the fields listed on the Fields tab.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the Query Classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

QuerySelectedFields

All the fields that make up the query definition. These include the fields selected as output fields, and fields added as QueryExpressions.

QueryCriteria

The selection criteria for the query. Each QueryCriteria object is made up of the following:

Logical	Any criteria objects after the first must include have a Logical value, either AND or OR.
Expression 1	A field or value that you want to base the selection criterion on, that is, Expression1 is the left-hand side of the criterion's comparison.
Operator	A mathematical or other operator used to specify the relationship between Expression 1 and Expression 2.
Expression 2	A field or other value, also called a comparison value, used with Expression 1, that is, Expression2 is the right-hand side of the criterion's comparison.

In PeopleSoft Query, a QueryCriteria is under the Criteria tab.

Expression can be made up of constant values, fields, subqueries, and so on.

See Also

Chapter 30, "Query Classes," Working With Criteria and Expressions, page 1241

QueryDBRecords and QueryDBRecordFields

A QueryDBRecord is a record in the database that can be used as a QueryRecord. The list of records is controlled by security: the only records displayed as QueryDBRecords are records accessible by the user.

The QueryDBRecordFields are the fields that make up the QueryDBRecords.

In PeopleSoft Query, the QueryDBRecords are under the Record tab. After you click on the plus sign next to a record, the QueryDBRecordFields are displayed.

 Copyright © 1988-2007, Oracle. All rights reserved.

Working With Criteria and Expressions

If you run a query after selecting the QueryFields (which executes a SQL statement, such as SELECT EMPLID, DEPTID from PS_QE_EMPLOYEE), the system retrieves *all* the data in those columns; that is, it retrieves the data from every row in the QueryRecord or records because there is no filter limiting the number of rows.

You can select which *rows* of data you want by adding selection criteria to the query.

This document assumes that you know how to use selection criteria. This section discusses working with the QueryCriteria and QueryExpression objects only in the Query API.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query, "Defining Selection Criteria"

Setting the Type

Before you can add a new expression (either Expression 1 or Expression 2) to your QueryCriteria, you must set the type for the expression.

The following code example adds a new criteria, sets the type for the first expression, adds the first expression to the main select of the query definition, then does the same thing for the second expression.

```
&MyQuerySelect = &MyQuery.QuerySelect;
&MyCriteria = &MyQuerySelect.AddCriteria();

/* make expression 1 a field */
&MyCriteria.Expr1Type = %Query_ExprField;

/* add the ABSENCE_TYPE field */
&MyCriteria.AddExpr1Field("A", "ABSENCE_TYPE");

/* make it not equal to */
&MyCriteria.Operator = %Query_CondNotEqual;

/* Make expression 2 a constant */
&MyCriteria.Expr2Type = %Query_ExprConstant;
&MyCriteriaExpr = &MyCriteria.AddExpr2Expression1();
&MyCriteriaExpr.Text = "VAC";
```

Considerations Adding New Expressions

When you use any of the QueryCriteria methods to add either a new Expression 1 or Expression 2, you destroy the existing value.

In general, you should use the QueryCriteria methods to add a new Expression 1 or Expression 2 only when you're adding a new criteria to a query.

Operator and Expression 2 Dependencies

Which values are valid for the Expression 2 type (Expr2Type property) depend on the value of the Operator property.

Copyright © 1988-2007, Oracle. All rights reserved.

The following table describes which Expr2Type values are valid with which values of Operator.

Operator	Expression 2
equal to not equal to greater than not greater than less than not less than	Constant Field Expression Subquery Prompt
Exists not exists	Subquery
Like not like	Constant (with wildcards) Prompt
is null is not null	
in tree not in tree	Tree Option
Eff Date <= Eff Date >= Eff Date < EffDate >	Field Expression Constant Current Date
First Eff Date Last Eff Date	

Copyright © 1988-2007, Oracle. All rights reserved.

Operator	Expression 2
in list not in list	In list Subquery
Between Not Between	Const-Const Const-Field Const-Expr Field-Const Field-Field Field-Expr Expr-Const Expr-Field Expr-Expr

See Also

Chapter 30, “Query Classes,” Operator, page 1326

Query Monitor

The Query Classes provide many methods and properties for examining the QueryStatistics, which you can use to report on the average execution time, the last date and time the query was run, and so on.

You can view the statistics for a query before you save it to the database.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query, “Getting Started with PeopleSoft Query”

Using Query Metadata

There are two ways of accessing information about a query:

- The Query classes (QueryOutputField, QueryRecord, and so on).
- The Metadata property.

If you use the Metadata property, the information is presented in a different format. It is also read-only. Using this property can be a quick and easy way to access information about a query.

Each Query Metadata object is a name-value pair. *Name* is an indicator of which Query Metadata property you’re accessing, and *Value* is the value of that property.

Copyright © 1988-2007, Oracle. All rights reserved.

While the value of each Query Metadata property is unique, the name may or may not be. For example, there is only one description (Descr) for each query, so there is only one Query Metadata property with *name* equal to Descr and *value* equal to the description for the query.

However, there may be more than one record for a query. A Query Metadata property exists for each record, with *name* equal to Record and *value* equal to one of the records in the query. The same is true for Input Param, Expression, Field, and Heading.

The following is a simple PeopleCode program to get the Query Metadata for a private query, ADDRESS_TEMP:

```
Local ApiObject &MyQuery, &MyMetacol, &MyMetadata;
Local File &MyFile;

&MyFile = GetFile("Metadata.Txt", "A");

&MyQuery = %Session.GetQuery();

&Rlst = &MyQuery.Open("ADDRESS_TEMP", False, 1);

If &Rlst = 0 Then
   &MyMetacol = &MyQuery.metadata;
   &MyFile.WriteLine("Name              Value");
   &MyFile.WriteLine("----------------------");

   For &i = 1 To &MyMetacol.count
      &MyMetadata = &MyMetacol.item(&i);
      &Name = &MyMetadata.name;
      &Value = &MyMetadata.Value;
      &MyFile.WriteLine(&Name | "              " | &Value);
   End-For;

Else
   WinMessage("Open query not successful");
End-If;
```

Here is the sample output:

```
Name              Value
----------------------
Descr             Temporary address query for testing
LongDescr
Public/Private    QEDMO
LastUpdDttm       2001-11-19-15.06.22.930000
LastUpdOprId      QEDMO
Record            QE_PERS_DATA
Field             QE_EMPLID
Field             QE_NAME
Field             ADDRESS1
Field             ADDRESS2
Field             ADDRESS3
Field             ADDRESS4
Field             CITY
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Field           COUNTY
Field           STATE
Field           QE_ZIP
Field           QE_COUNTRY
Heading         ID
Heading         Name
Heading         Address 1
Heading         Address 2
Heading         Address 3
Heading         Address 4
Heading         City
Heading         County
Heading         St
Heading         QE_ZIP
Heading         QE_COUNTRY
```

Running a Query

With Query API, you have the following options for running a query:

- Use the RunToRowset method

  This method takes a Query Prompt Record as input and returns a PeopleCode rowset containing the query result. You should always use a standalone rowset (that is, one that was created using CreateRowset).

- Use the RunToFile method

  This method takes a Query Prompt Record and a file name as input and writes the query result to the file.

Note. For the Query classes, all date, time, and datetime fields that are part of a query are now required fields when running a query.

Considerations Using the RunTo Methods

RunToRowset may require a lot of memory for the rowset object, therefore it also takes more processing time. PeopleSoft recommends that RunToRowset not be used for getting large result sets, of the order of 50000 or more.

Both of the RunTo methods can be called to run a query before saving it, that is, it isn't necessary to first save the query.

The last parameter of both RunToRowset and RunToFile is the maximum number of rows to fetch.

- -1 returns all rows regardless of the setting on the security profile.
- 0 returns the maximum number of rows allowed by the security profile.
- >0 is the limit on the number of rows.

See Chapter 30, "Query Classes," RunToFile, page 1270.

See Chapter 30, "Query Classes," RunToRowset, page 1272.

Copyright © 1988-2007, Oracle. All rights reserved.

Specifying User's Language

For scheduled queries, the system uses the language specified in the user's profile. It does not use the language selected during signon. The system also uses the International and Regional settings the user specified using My Personalizations. If no personal setting have been specified, the system uses the default installation international settings.

Note. Most PeopleSoft components can default to international settings from the browser if the user has not set any user specific settings. However, this is not available for scheduled queries or any Process Scheduler processes.

Query Security

Security is critical for your business data. Typically, you don't want everyone in your company to have access to all the data. All the standard security features used with PeopleSoft applications are integrated in the PeopleSoft Query, as well as the Query classes.

In addition, you can use the QuerySecurityProfile Class to view the current user's security profile for PeopleSoft Query. This class doesn't contain any methods, and all the properties are read-only.

See Also

Chapter 30, "Query Classes," QuerySecurityProfile Class, page 1346

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query, "PeopleSoft Query Security"

Enterprise PeopleTools 8.49 PeopleBook: Security Administration, "Understanding PeopleSoft Security"

Error Handling With Query Classes

All errors for the Query classes, like the other APIs, are logged in the PSMessages collection, instantiated from a session object. In addition, some methods return error codes. See the individual method description to see if it returns anything.

The query classes log errors "interactively", that is, as they happen. For example, suppose you specified an invalid query name. The error would be logged in the PSMessages collection as soon as you executed the GetQuery method.

When to check for errors is application-specific. However, if you check for errors after every assignment, you may see a performance degradation.

PeopleSoft recommends that you check for invalid prompts after using any of the following methods or properties:

- Save, RunToRowset, or RunToFile Query class methods.
- SQL, RuntimePrompts, or Prompts Query class properties.
- Any of the methods or properties in the QueryPrompts collection.

The easiest way to check for errors is to check the number of messages in the PSMessages collection, using the Count property. If the Count is 0, there are no errors.

```
Local ApiObject &MySession;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local ApiObject &ERRORCOL;
Local ApiObject &Query, &QueryList;

&MySession = %Session;

If &MySession Then
   /* connection is good */

   &QueryList = &MySession.SearchPublicQueries(%Query_ListQuery, %Query_FindName,⇒
 "%", True);

   For &I = 1 to &QueryList.Count
      &Query = &QueryList.Item(&I);

      /* Do processing */

      /* Do error checking */

      &ERRORCOL = &MySession.PSMessages;
      If (&ERRORCOL.Count <> 0) Then
         /* errors occurred - do processing */
      Else
         /* no errors */
      End-If;

   End-For;
Else
   /* do processing for no connection */
End-If;
```

See Also

Chapter 36, “Session Class,” Error Handling, page 1512

Understanding QueryOutputFields and QuerySelectedFields

If you select a field from a Query Record, it becomes a QueryOutputField, that is, it becomes one of the columns in the SQL output. QuerySelectedFields consist of displayed fields and Query Expression Fields.

QuerySelectedFields and QueryOutputFields have all the same methods and properties, so in this documentation, they’re described together under the heading QueryField.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the Query Classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

Copyright © 1988-2007, Oracle. All rights reserved.

Understanding Having Criteria

You can have Where (simple) criteria or Having criteria in a query. In most cases, you only have a WHERE clause, that is, simple criteria. Having criteria is only used in some special cases of aggregation queries. They represent the Having clause of the SQL statement, similar to the following:

```
Select A.DEPTID, COUNT(A.EMPLID)

FROM PS_QE_EMPLOYEE A

Group by A.DEPTID

HAVING COUNT(A.EMPLID)>5
```

Having Criteria are separated from simple criteria. They're accessed as follows:

- Having criteria are accessed using either the AddHavingCriteria QuerySelect method or HavingCriteria QuerySelect property
- Simple criteria are accessed using either the AddCriteria QuerySelect method or the Criteria QuerySelect property.
- However, the Having criteria and the simple criteria have all the same methods and properties, so in this documentation, they're described together under the heading QueryCriteria.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query, "Defining Selection Criteria," Defining HAVING Criteria

Data Type of Query Objects

All query objects, like a query collection, a QueryDBRecord, a QueryExpression, and so on, are declared as type ApiObject. For example,

```
Global ApiObject &MyQuery;

Local ApiObject &MyExpression;
```

Scope of the Query Objects

All query objects can be instantiated from PeopleCode only.

You can use this object anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

 Copyright © 1988-2007, Oracle. All rights reserved.

You can only instantiate a query object from a session object. You *must* instantiate the session object before you can instantiate a query object or query collection. Use the %Session system variable to connect to the existing session.

```
Local ApiObject &QueryList;
Local ApiObject &MySession;

&MySession = %Session;

If &MySession <> Null Then
   /* connection is good */
   /* Search for public queries of type QUERY, search both the name and⇒
 description for the pattern MyQ%, and do a case insensitive search */
   &QueryList = &MySession.SearchPublicQueries(1, 3, "MyQ%", False);
Else
   /* do error processing */
End-if;
```

Session Class Methods

The following methods are part of the Query API. However, they're used with a Session object.

See Also

Chapter 36, "Session Class," page 1511

AdvancedSearchQueries

Syntax

AdvancedSearchQueries(*GetFavorites*, *QueryName*, *QueryNameOp*, *Descr*, *DescrOp*, *FolderName*, *FolderNameOp*, *RecordName*, *RecordNameOp*, *FieldName*, *FieldNameOp*, *TreeName*, *TreeNameOp*, *QueryType*, *OwnerType*, *CaseSensitive*)

Description

Use the AdvancedSearchQueries method to do more complex searches for queries.

Using Search Operators

All of the parameters for this method work in pairs, with the value in the first of the paired parameters further distinguished by the search operator used in the second parameter. For example, the value specified in *FieldName* is paired with the value specified in *FieldNameOp*.

All of the search operator parameters use the following values. Note that the format of the value of the first parameter is sometimes affected by the value of the search operator parameter.

Constant	Description
%Query_AdvSrchBegins	Name begins with the values specified.
%Query_AdvSrchContains	Name contains the value specified.

Copyright © 1988-2007, Oracle. All rights reserved.

Constant	Description
%Query_AdvSrchEquals	Name equals the value specified.
%Query_AdvSrchNotEquals	The name does not equal the value specified.
%Query_AdvSrchLessThan	The name is less than the value specified.
%Query_AdvSrchLessEquals	The name is less than or equal to the value specified.
%Query_AdvSrchGreaterThan	The name is greater than the value specified.
%Query_AdvSrchGreaterEquals	The name is greater than or equal to the value specified.
%Query_AdvSrchBetween	The name is between two values specified by a comma. Do not use quotation marks. For example, ACCT1,ACCT9.
%Query_AdvSrchIn	The name is in the list specified list. The values are separated with commas. Do not use quotation marks. For example, ACCT1, ACCT2, ACCT3

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
GetFavorites	Specify whether to return only queries marked as favorites. This parameter takes a Boolean value: true if you only want favorite queries returned, false otherwise. If you specify true for this parameter, all other parameters are ignored.
QueryName	Specify the name of the query you want returned, as a string. Use this parameter with the *QueryNameOp* parameter.
QueryNameOp	Specify the operator to be used with the *QueryName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
Descr	Specify the description you want returned, as a string. Use this parameter with the *DescrOp* parameter.
DescrOp	Specify the operator to be used with the *Descr* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FolderName	Specify the name of the folder of the query or queries you want returned, as a string. Use this parameter with the *FolderNameOp* parameter.
FolderNameOp	Specify the operator to be used with the *FolderName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
RecordName	Specify the name of the record used with the query (queries) you want returned, as a string. Use this parameter with the *RecordNameOp* parameter.
RecordNameOp	Specify the operator to be used with the *RecordName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FieldName	Specify the name of the field associated with the query (queries) you want returned, as a string. Use this parameter with the *FieldNameOp* parameter.
FieldNameOp	Specify the operator to be used with the *FieldName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
TreeName	Specify the name of the tree associated with the query (queries) you want returned, as a string. Use this parameter with the *TreeNameOp* parameter.
TreeNameOp	Specify the operator to be used with the *TreeName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
QueryType	Specify the type of query, as a string. Valid values are:

Numeric Value	Constant Value	Description
1	%Query_Query	Find queries of the type Query.
5	%Query_DBAgent	Find queries of the type Process.
4	%Query_Role	Find queries of the type Role
7	%Query_Archive	Find queries of the type Archive.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameter	Description
OwnerType	Specify the type of owner, whether the query is public or private.
CaseSensitive	This parameter has not yet been implimented.

Returns

A reference to a Query collection containing zero or more queries.

AdvancedSearchRecords

Syntax

AdvancedSearchRecords(*RecordName*, *RecordNameOp*, *Descr*, *DescrOp*, *FieldName*, *FieldNameOp*, *TreeName*, *TreeNameOp*, *CaseSensitive*)

Description

Use the AdvancedSearchRecords method to do more complex searches for records.

Security applies to the results of this list, that is, you have access to all the records your UserID (permission list) allows you to access.

Using Search Operators

All of the parameters for this method work in pairs, with the value in the first of the paired parameters further distinguished by the search operator used in the second parameter. For example, the value specified in *FieldName* is paired with the value specified in *FieldNameOp*.

All of the search operator paramerters use the following values. Note that the format of the value of the first parameter is sometimes affected by the value of the search operator parameter.

Constant	Description
%Query_AdvSrchBegins	Name begins with the values specified.
%Query_AdvSrchContains	Name contains the value specified.
%Query_AdvSrchEquals	Name equals the value specified.
%Query_AdvSrchNotEquals	The name does not equal the value specified.
%Query_AdvSrchLessThan	The name is less than the value specified.
%Query_AdvSrchLessEquals	The name is less than or equal to the value specified.
%Query_AdvSrchGreaterThan	The name is greater than the value specified.
%Query_AdvSrchGreaterEquals	The name is greater than or equal to the value specified.

 Copyright © 1988-2007, Oracle. All rights reserved.

Constant	Description
%Query_AdvSrchBetween	The name is between two values specified by a comma. Do not use quotation marks. For example, ACCT1,ACCT9.
%Query_AdvSrchIn	The name is in the list specified list. The values are separated with commas. Do not use quotation marks. For example, ACCT1, ACCT2, ACCT3

Parameters

Parameter	Description
RecordName	Specify the name of the record you want returned, as a string. Use this parameter with the *RecordNameOp* parameter.
RecordNameOp	Specify the operator to be used with the *RecordName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
Descr	Specify the description you want returned, as a string. Use this parameter with the *DescrOp* parameter.
DescrOp	Specify the operator to be used with the *Descr* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FolderName	Specify the name of the folder of the records you want returned, as a string. Use this parameter with the *FolderNameOp* parameter.
FolderNameOp	Specify the operator to be used with the *FolderName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
FieldName	Specify the name of the field associated with the record you want returned, as a string. Use this parameter with the *FieldNameOp* parameter.
FieldNameOp	Specify the operator to be used with the *FieldName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
TreeName	Specify the name of the tree associated with the record you want returned, as a string. Use this parameter with the *TreeNameOp* parameter.
TreeNameOp	Specify the operator to be used with the *TreeName* parameter. The valid values for this parameter are found in the Using Search Operators section, above.
CaseSensitive	**Note.** This parameter has not been implemented yet. Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

Returns

A reference to a QueryDBRecord collection containing zero or more records.

FindQueryDBRecords

Syntax

```
FindQueryDBRecords()
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The FindQueryDBRecords method returns a reference to a QueryDBRecord collection, filled with zero or more records.

Security applies to the results of this list, that is, you have access to all the records your UserID (permission list) allows you to access.

Parameters

None.

Returns

A reference to a QueryDBRecord collection containing zero or more records.

See Also

Chapter 30, "Query Classes," QueryDBRecord Collection, page 1349 and Chapter 30, "Query Classes," QueryDBRecord Class, page 1352.

FindQueries

Syntax

```
FindQueries()
```

Description

The FindQueries method returns a reference to a Query collection, filled with zero or more queries.

FindQueries Considerations

FindQueries returns both public and private queries. It depends on your database whether the private query is returned first or the public query. If you have two queries with the same name, it depends on your database whether the first use of Item returns the private or the public query.

Parameters

None.

Returns

A reference to a Query collection containing zero or more queries.

Example

In the following example, all available queries are returned:

```
Local ApiObject &MySession;
Local ApiObject &MyList;

&MySession = %Session
&MyList = &MySession.FindQueries();
```

See Also

Chapter 30, "Query Classes," Open, page 1268, Chapter 30, "Query Classes," FindQueriesDateRange, page 1255 and Chapter 30, "Query Classes," Query Collection, page 1260.

 Copyright © 1988-2007, Oracle. All rights reserved.

FindQueriesDateRange

Syntax

FindQueriesDateRange(*StartDateString*, *EndDateString*)

Description

The FindQueriesDateRange method returns a reference to a Query collection, filled with zero or more queries that match the specified date range.

FindQueriesDateRange Considerations

FindQueriesDateRange returns both public and private queries. It depends on your database whether the private query is returned first or the public query. If you have two queries with the same name, it depends on your database whether the first use of Item returns the private or the public query.

Parameters

Parameter	Description
StartDateString	Specify the year, month, and day of the beginning date that you want to look for. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or* YYYY/MM/DD.
EndDateString	Specify the year, month, and day of the end date. This parameter takes a string value. You can specify the date either as YYYY-MM-DD *or* YYYY/MM/DD.

Returns

A reference to a Query collection containing zero or more queries.

Example

```
Local ApiObject &MySession, &QueryList;

&MySession = %Session;

&Start = GetField(VOLUN_ACT_WRK.START_DT_STR).Value;
&End = GetField(VOLUN_ACT_WRK.END_DT_STR).Value;

&QueryList = &MySession.FindQueriesDateRange(&Start, &End);
```

See Also

Chapter 30, “Query Classes,” FindQueries, page 1254, Chapter 30, “Query Classes,” Open, page 1268 and Chapter 30, “Query Classes,” Query Collection, page 1260.

GetQuery

Syntax

GetQuery()

Description

The GetQuery method returns an empty query object. After you have an empty query object, you can use it to open an existing query (using the Open method) or to create a new query definition (using the Create method).

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to an empty query object if successful, NULL otherwise.

Example

```
&MyQuery = &MySession.GetQuery();

If &MyQuery.Open("PHONELIST") Then
```

See Also

Chapter 30, "Query Classes," FindQueries, page 1254, Chapter 30, "Query Classes," FindQueriesDateRange, page 1255, Chapter 30, "Query Classes," Open, page 1268 and Chapter 30, "Query Classes," Create, page 1265.

GetQuerySecurityProfile

Syntax

GetQuerySecurityProfile()

Description

Use GetQuerySecurityProfile to return the current user's security profile for PeopleSoft Query. You can then use the QuerySecurityProfile properties to determine if the user can modify queries, the maximum number of rows to fetch for this user, and so on.

Parameters

None.

Returns

A reference to a QuerySecurityProfile if successful, NULL otherwise.

Example

```
&MySecProfile = %Session.GetQuerySecurityProfile();
If &MySecProfile.CanModifyQuery Then
   /* do some processing */
End-If;
```

See Also

Chapter 30, "Query Classes," QuerySecurityProfile Class, page 1346

SearchQueryDBRecords

Syntax

SearchQueryDBRecords(*SearchType*, *Pattern*, *CaseSensitive*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The SearchQueryDBRecords method returns a reference to a QueryDBRecord collection, filled with zero or more records.

Security applies to the results of this list, that is, you have access to all the records your UserID (permission list) allows you to access.

You can use wildcard characters % and _ when searching. % means find all characters, while _ means find a single character. For example, if you wanted to find all queries that started with the letter M, use "M%" for *Pattern*. To find either DATE or DATA, use "DAT_" for *Pattern*.

These characters can be escaped (that is, ignored) using a \. For example, to search for a query that contains the character %, use \% in *Pattern*.

If *Pattern* is an empty string, this method retrieves all queries of the specified type (that is, specifying "" for *Pattern* is the same as specifying "%").

Parameters

Parameter	Description
SearchType	Specify the type of search to be used with the given pattern. You can use either a constant or a number value for this parameter. Values are:

Numeric Value	Constant Value	Description
1	%Query_FindName	Search for records with the name matching the given pattern.
2	%Query_FindDescr	Search for records with the description matching the given pattern.
3	%Query_FindNameDescr	Search for records with either the name or the description matching the given pattern.

Parameter	Description
Pattern	Specify the pattern to be used when searching for records.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

Returns

A reference to a QueryDBRecord collection containing zero or more records.

Example

```
Local ApiObject &MySession, &DBRecList;

&MySession = %Session;

DBRecList = &MySession.SearchQueryDBRecords(%Query_FindName, "A%", False);
```

See Also

Chapter 30, "Query Classes," FindQueryDBRecords, page 1253

Copyright © 1988-2007, Oracle. All rights reserved.

SearchPrivateQueries

Syntax

SearchPrivateQueries(*QueryType*, *UserId*, *SearchType*, *Pattern*, *CaseSensitive*)

Description

The SearchPrivateQueries method returns a reference to a Query collection, filled with zero or more Private queries that match the specified *SearchType*, *UserId*, *Pattern*, and *CaseSensitive* choice

Parameters

Parameter	Description
QueryType	Specify the type of query to be searched for. You can specify either a constant or number value for this parameter. Values are:

Numeric Value	Constant Value	Description
1	%Query_Query	Find queries of the type Query.
3	%Query_DBAgent	Find queries of the type Process.
4	%Query_Role	Find queries of the type Role
10	N/A	Find queries of the type Archive.

Parameter	Description
UserId	Specify the user Id to be used to find currently signed-on user's private queries.
SearchType	Specify the type of search to be used with the given pattern. You can use either a constant or a number value for this parameter. Values are:

Numeric Value	Constant Value	Description
1	%Query_FindName	Search for queries with the name matching the given pattern.
2	%Query_FindDescr	Search for queries with the description matching the given pattern.
3	%Query_FindNameDescr	Search for queries with either the name or the description matching the given pattern.

Parameter	Description
Pattern	Specify the pattern to be used when searching for queries.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

Returns

A reference to a Query collection containing zero or more queries.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example retrieves all private queries of type Query which start with A and do a case-insensitive search, that is, get all queries starting with A or a.

```
Local ApiObject &MySession, &DBRecList;

&MySession = %Session;

DBRecList = &MySession.SearchPrivateQueries(%Query_ListQuery, %UserId, %Query_Find⇒
Name, "A%", False);
```

See Also

Chapter 30, "Query Classes," FindQueries, page 1254, Chapter 30, "Query Classes," FindQueriesDateRange, page 1255 and Chapter 30, "Query Classes," SearchPublicQueries, page 1259.

SearchPublicQueries

Syntax

SearchPublicQueries(*QueryType, SearchType, Pattern, CaseSensitive*)

Description

The SearchPublicQueries method returns a reference to a Query collection, filled with zero or more Public queries that match the specified *SearchType*, *Pattern,* and *CaseSensitive* choice.

Parameters

Parameter	Description
QueryType	Specify the type of query to be searched for. You can specify either a constant or number value for this parameter. Values are:

Numeric Value	Constant Value	Description
1	%Query_ListQuery	Find queries of the type Query.
3	%Query_ListDBAgent	Find queries of the type Process
4	%Query_ListRole	Find queries of the type Role
10	N/A	Find queries of type Archive

Parameter	Description
SearchType	Specify the type of search to be used with the given pattern. You can use either a constant or a number value for this parameter. Values are:

Numeric Value	Constant Value	Description
1	%Query_FindName	Search for queries with the name matching the given pattern.
2	%Query_FindDescr	Search for queries with the description matching the given pattern.
3	%Query_FindNameDescr	Search for queries with either the name or the description matching the given pattern.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameter	Description
Pattern	Specify the pattern to be used when searching for queries.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it isn't.

Returns

A reference to a Query collection containing zero or more queries.

Example

The following example retrieves all public queries of type Query that start with A and does a case-insensitive search, that is, get all queries starting with A or a.

```
Local ApiObject &MySession, &DBRecList;

&MySession = %Session;

DBRecList = &MySession.SearchPublicQueries(%Query_ListQuery, %Query_FindName,⇒
 "A%", False);
```

See Also

Chapter 30, "Query Classes," FindQueries, page 1254, Chapter 30, "Query Classes," FindQueriesDateRange, page 1255 and Chapter 30, "Query Classes," SearchPrivateQueries, page 1258.

Query Collection

A query collection is returned from the following session methods:

- FindQueries
- FindQueriesDateRange
- SearchPrivateQueries
- SearchPublicQueries

See Also

Chapter 30, "Query Classes," FindQueries, page 1254

Chapter 30, "Query Classes," FindQueriesDateRange, page 1255

Chapter 30, "Query Classes," SearchPublicQueries, page 1259

Chapter 30, "Query Classes," SearchPrivateQueries, page 1258

Query Collection Methods

In this section, we discuss each query collection method.

 Copyright © 1988-2007, Oracle. All rights reserved.

First

Syntax

First()

Description

The First method returns the first Query object in the Query collection.

Parameters

None.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
&MyQuery = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the Query object that exists at the *number* position in the Query collection.

Item Considerations

FindQueries and FindQueriesDateRange return both public and private queries. It depends on your database whether the private query is returned first or the public query. If you have two queries with the same name, it depends on your database whether the first use of Item returns the private or the public query.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the Query object that you want returned.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryColl.Count;
   &MyQuery = &QueryColl.Item(&I);
   /* do processing */
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the Query object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing Query within the Query collection. If you specify an invalid name, the object is NULL. The length of this parameter is 30 characters.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
&MyQuery = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next Query object in the Query collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a Query object if successful, NULL otherwise.

Example

```
&MyQuery = &MyCollection.Next();
```

Query Collection Property

In this section, we discuss the Count query collection property.

 Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of Query objects in the Query Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Query Class

A query object is returned from the following:

- The GetQuery session method.
- The Query Collection methods First, Item, ItemByName, or Next.

See Chapter 30, “Query Classes,” GetQuery, page 1255.

See Chapter 30, “Query Classes,” Query Collection, page 1260.

Query Class Methods

In this section, we discuss each Query class method.

AddQuerySelect

Syntax

AddQuerySelect()

Description

Use the AddQuerySelect method to add the first Query Select statement into the query definition. This method returns the instance of the newly created QuerySelect.

Parameters

None.

Returns

A reference to a QuerySelect object. If the query already contains the Main Select, it returns NULL.

See Also

Chapter 30, “Query Classes,” QuerySelect Collection, page 1293

Copyright © 1988-2007, Oracle. All rights reserved.

AddPrompt

Syntax

AddPrompt(*PromptName*)

Description

The AddPrompt method adds a prompt with the given name to the query definition. This method returns a new QueryPrompt object that you can then use to specify details about the prompt.

Note. Prompt names are *not* checked for uniqueness. Each new prompt is just added to the list of prompts.

Parameters

Parameter	Description
PromptName	Specify the name of the new prompt with a string. The length of this parameter is 30 characters.

Returns

A reference to a QueryPrompt object.

See Also

Chapter 30, "Query Classes," DeletePrompt, page 1266, Chapter 30, "Query Classes," Prompts, page 1280 and Chapter 30, "Query Classes," QueryPrompt Class, page 1363.

Close

Syntax

Close()

Description

The Close method closes the query, freeing the memory associated with that object, and discarding any changes made to the query since the last save. The Close method can be used only on an open query, not a closed query. This means you must have opened the query with the Open or Create methods before you can close it. To save any changes, you must use the Save method before using Close.

It's very important to close your query when you're finished processing. Canceling out of a page does *not* close a query. You may receive error messages every other time you run your program if you haven't closed your queries.

Parameters

None.

Returns

None.

See Also

Chapter 30, "Query Classes," Open, page 1268, Chapter 30, "Query Classes," Save, page 1275 and Chapter 30, "Query Classes," Create, page 1265.

 Copyright © 1988-2007, Oracle. All rights reserved.

Create

Syntax

Create(*queryname, Public, Type, Description, LongDescription*)

Description

The Create method creates a new query, based on the parameters passed with the method. The specified query must be a *new* query.

Warning! If you specify the name of a query that already exists, the existing query is overwritten by the new query.

The Create method can be used only with a closed query, it cannot be used on an open query.

After you create a new query, you don't have to open it with the Open method. The existing query object points to the new query.

Creating a new query doesn't create the query in the database. You must save the query (with the Save method) to commit it to the database.

Parameters

Parameter	Description
Queryname	Name of the query to be created. This parameter takes a string value. This parameter takes 30 characters. A query name can contain only alphabetic and numeric characters, as well as underscores.
Public	Specify if the query is public or private. This parameter takes a Boolean value: True if the query is public, False if it's a private query.
Type	Specify the type of query. This parameter takes either a numeric or constant value. Values are:

Numeric Value	Constant Value	Description
1	%Query_Query	Query
3	%Query_View	View
4	%Query_Role	Role
5	%Query_DBAgent	Process
7	%Query_Archive	Archive

Parameter	Description
Description	Specify a short description for the query. This parameter takes a string value. This parameter takes 30 characters.
LongDescription	Specify a long description for the query. This parameter takes a string value. The length of this parameter depends on your system database limit for LONG fields.

Returns

An integer value: 0 means the query was created successfully.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
/* Use the existing session */

&MySession = %Session;

&MyQry = &MySession.GetQuery();

/* create a new query : public type-User */
&Rslt = &MyQry.Create("MYQUERY", True, %Query_Query, "My Query", "My first⇒
 Query");

If &Rslt = 0 Then
   /* Query created successfully */
Else
   /* do error processing */
End-If;
```

See Also

Chapter 30, “Query Classes,” GetQuery, page 1255, Chapter 30, “Query Classes,” Close, page 1264 and Chapter 30, “Query Classes,” Save, page 1275.

Delete

Syntax

Delete()

Description

The Delete method deletes the specified query from the database. The Delete method can be used only with an open query, it cannot be used on a closed query. Before you use the Delete method, you must explicitly open the query object to be deleted (with either the Open or Create method.)

Parameters

None.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 30, “Query Classes,” Save, page 1275

DeletePrompt

Syntax

DeletePrompt(*PromptNumber*)

Description

The DeletePrompt method deletes a prompt from a query definition.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PromptNumber	Specify the numeric location in the query definition of the prompt that you want to delete.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 30, "Query Classes," AddPrompt, page 1264, Chapter 30, "Query Classes," Prompts, page 1280 and Chapter 30, "Query Classes," QueryPrompt Class, page 1363.

FindExpression

Syntax

```
FindExpression(ExpressionNumber)
```

Description

Use the FindExpression method to search the query definition for an expression with the given expression number. Although expressions are associated with the QuerySelect in which they are defined, PeopleSoft Query doesn't qualify expressions by Select number. Therefore, each expression has a unique number across all QuerySelect objects.

Parameters

Parameter	Description
ExpressionNumber	Specify the numeric value of the expression number to be searched for in the query definition.

Returns

A QueryExpression object if successful, NULL otherwise.

See Also

Chapter 30, "Query Classes," AddPrompt, page 1264, Chapter 30, "Query Classes," Prompts, page 1280 and Chapter 30, "Query Classes," QueryPrompt Class, page 1363.

FormatResultString

Syntax

```
FormatResultString(&Rowset, Output_Format, StartRow, EndRow)
```

Description

Use the FormatResultString method to generate a string containing the content of the input (the rowset) in HTML, XLS, or CSV format. You can specify the range of rows to be in the rowset to be used as input. The rowset object is created using the RunToRowset method.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&Rowset	Specify an already instantiated and populated rowset object containing the query result. This rowset is created using the RunToRowset method.
Output_Format	Specify the format of the output. You can specify either a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
2	%Query_PDF	The output is in PDF format.
5	%Query_HTML	The output is in HTML format.
8	%Query_XLS	The output for Excel in HTML format.
14	%Query_TXT	The output for text in CSV format.

Parameter	Description
Start_Row	Specify the first row in the rowset that you want to use for output. This parameter takes a numeric value.
End_Row	Specify the last row in the rowset that you want to use for output. This parameter takes a numeric value.

Returns

A string containing the formatted output.

Example

/* you want to use RowCount, not ActiveRowCount, to get the total number of rows. */

```
&End = &MyRowset.RowCount;

&FormString = &MyQuery.(&MyRowset, %Query_TXT, 1, &End);
```

See Also

Chapter 30, “Query Classes,” RunToFile, page 1270 and Chapter 30, “Query Classes,” RunToRowset, page 1272.

Open

Syntax

Open(*QueryName*, *Public*, *Update*)

Description

The Open method opens the query object specified by the parameters. The Open method can be used only with a closed query, it cannot be used on an open query. You cannot read or set any properties of a query until after you open it.

Considerations for Opening Different Types of Queries

When you use the Open method, the system tries to open queries according to the type of query. The following is the order used for searching for the query to open:

 Copyright © 1988-2007, Oracle. All rights reserved.

1. Query (User)
2. View
3. Role
4. Process
5. Archive

All query names are unique. You can't have two queries with the same name, just of different types.

Parameters

Parameter	Description
QueryName	Specify the name of the query to be opened. You must specify an existing query. This parameter takes a string value.
Public	Specify if the query is public or private. This parameter takes a Boolean value: True if the query is public, False if it's a private query.
Update	This parameter is required, but is unused in this release. You must specify a either True or False.

Returns

An integer value: 0 means the query was opened successfully.

See Also

Chapter 30, "Query Classes," GetQuery, page 1255, Chapter 30, "Query Classes," Close, page 1264 and Chapter 30, "Query Classes," Save, page 1275.

Rename

Syntax

Rename(*NewQueryName*)

Description

The Rename method renames the existing query definition with *NewQueryName*. The Rename method can be used only with an open query, it cannot be used on a closed query. Before you use the Rename method, you must explicitly open the query object to be renamed (with either the Open or Create method.)

Note. The Rename method takes place immediately. You don't have to save the query for the rename to occur.

Parameters

Parameter	Description
NewQueryName	Specify the new name for the existing query. The maximum length of this parameter is 30 characters.

Returns

An integer value: 0 means the query was renamed successfully.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 30, "Query Classes," Open, page 1268, Chapter 30, "Query Classes," Close, page 1264 and Chapter 30, "Query Classes," Save, page 1275.

RunToFile

Syntax

RunToFile(*&PromptRecord, Destination, OutputFormat, MaxRows*)

Description

Use the RunToFile method to execute the Query and return the result to the file specified with *Destination*. The query should be an existing query in the database, or it should have been saved using the Save method.

Because a Query may have runtime prompts (that is, criteria defined using Prompt), this method requires those values to be passed in when you execute this method. *PromptRecord* is a PeopleCode record object containing the prompt values as fields in the record. You can use the PromptRecord property to obtain this object.

Destination must include the absolute path name of where the file is to be created.

If the specified subdirectory does not exist, this method does *not* automatically create them for you, and you receive an error.

If you specify HTML as the output format, the PeopleSoft Query style sheet PSQUERYSTYLEDEF is used for formatting the output.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Style Sheet Definitions"

Parameters

Parameter	Description
&PromptRecord	Specify an instance of a PeopleCode record that contains the runtime prompts and values required for running the query.
Destination	Specify the absolute path name of where the files are to be created.
OutputFormat	Specify the format of the data being written to the file. You can use either a constant or a numeric value for this parameter. The values are:

Numeric Value	Constant Value	Description
2	%Query_PDF	The output is in PDF format.
5	%Query_HTML	The output is in HTML format.
8	%Query_XLS	The output for Excel in HTML format.
14	%Query_TXT	The output for txt in CSV format.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameter	Description
MaxRows	Specify the maximum number of rows to be fetched. This parameter takes a numeric value. The values are: • -1 returns all rows *regardless* of the setting on the Query security profile (MaxRowsFetched). • 0 returns the maximum number of rows allowed by the Query security profile. • >0 is the limit on the number of rows.

Returns

Returns 0 if successful.

Example

To run a query using the Query API RunToFile method:

1. Open the query.

```
&aRunQry = %Session.GetQuery();

&aRunQry.Open(&sQryName, False, False);
```

2. Obtain the PromptRecord for the query.

```
&aQryPromptRec = &aRunQry.PromptRecord;
```

This instance of the PromptRecord can be passed to the PeopleCode Prompt function to prompt the user for the runtime values, as follows:

```
&nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
```

3. Run the query.

Now that you have the runtime values, the query can be run, as follows:

```
&aRowSet = &aRunQry.RunToFile(&aQryPromptRec, "c:\temp\QueryOutput.html", %Query_⇒
PDF, 0);
```

4. Access the data of the rowset.

You can now manipulate the rowset using the data buffer access methods and properties:

```
&aRowSet(&i).GetRecord(1).GetField(&j).Value
```

The following is a complete sample code example:

```
Local Rowset &aRowSet;
   Local Row &aRow;
   Local Record &aQryPromptRec;
   Local Record &aRec;
   Local ApiObject &aRunQry;
   &strHTML = "";
   &aRunQry = %Session.GetQuery();
   If (&aRunQry.Open(&sQryName, False, False) <> 0) Then
      &strHTML = "Error in opening query";
   Else
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      &aQryPromptRec = &aRunQry.PromptRecord;
      &strQryName = &aRunQry.Name;
   If &aQryPromptRec <> Null Then
      &nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
   End-If;
   If (&aRunQry.RunToFile(&aQryPromptRec, "c:\temp\" | &aRunQry.Name, 3, 0) = 0)⇒
 Then
      &strHTML = "Resultset saved into file successfully.";
   Else
      &strHTML = "Failed to save Resultset into file.";
   End-If;
End-If;
```

See Also

Chapter 30, "Query Classes," RunToRowset, page 1272 and Chapter 30, "Query Classes," PromptRecord, page 1281.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Prompt

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

RunToRowset

Syntax

RunToRowset(*&PromptRecord, MaxRows*)

Description

Use the RunToRowset method to execute the Query and return the result to a rowset. The query should be an existing query in the database, or it should have been saved using the Save method.

Because a Query may have runtime prompts (that is, criteria defined using Prompt), this method requires those values to be passed in when you execute this method. *PromptRecord* is a PeopleCode record object containing the prompt values as fields in the record. The PromptRecord property can be used to obtain this object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&PromptRecord	Specify an instance of a PeopleCode record that contains the runtime prompts and values required for running the query.
MaxRows	Specify the maximum number of rows to be fetched. This parameter takes a numeric value. The total number of rows fetched is the minimum value of this parameter's value, and the application server configuration parameter Max Fetch Size (which is specified in KB.) The values are: • -1 returns all rows *regardless* of the setting on the Query security profile (MaxRowsFetched). • 0 returns the maximum number of rows allowed by the Query security profile. • >0 is the limit on the number of rows.

Returns

If successful, the query result is returned as a populated PeopleCode rowset. If the query wasn't successful, the method returns NULL. If unsuccessful, check the PSMessages collection for errors.

Example

To run a query using the Query API RunToRowset method:

1. Open the Query.

```
&aRunQry = %Session.GetQuery();

&aRunQry.Open(&sQryName, False, False);
```

2. Get the PromptRecord for the Query.

```
&aQryPromptRec = &aRunQry.PromptRecord;
```

   This instance of the PromptRecord can be passed to the PeopleCode Prompt function to prompt the user for the runtime values, as follows:

```
&nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
```

3. Run the query.

   Now that you have the prompt values, you can run the query:

```
&aRowSet = &aRunQry.RunToRowset(&aQryPromptRec, 0);
```

4. Access the data of the rowset.

   You can now manipulate the rowset using the data buffer access methods and properties:

```
&aRowSet(&i).GetRecord(1).GetField(&j).Value
```

The following is a complete sample code example:

```
Local Rowset &aRowSet;
 Local Row &aRow;
 Local Record &aQryPromptRec;
 Local Record &aRec;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   Local ApiObject &aRunQry;
   &strHTML = "";
   &aRunQry = %Session.GetQuery();
   If (&aRunQry.Open(&sQryName, False, False) <> 0) Then
      &strHTML = "Error in opening query";
   Else
      &aQryPromptRec = &aRunQry.PromptRecord;
      &strQryName = &aRunQry.Name;
   If &aQryPromptRec <> Null Then
      &nResult = Prompt(&strQryName | " Prompts", "", &aQryPromptRec);
      &aRowSet = &aRunQry.RunToRowset(&aQryPromptRec, 0);
         If &aRowSet <> Null Then
               &strHTML = &strHTML | "<table cellpadding='2' cellspacing='0' width=⇒
⇒
⇒
⇒
'90%'>";
               &aRec = &aRowSet(1).GetRecord(1);
               &QrySelOutputFldCol = &aRunQry.QuerySelect.QueryOutputFields;
               If &QrySelOutputFldCol <> Null Then
                  &strHTML = &strHTML | "<TR><TD>";
                  &strHTML = &strHTML | "<table border='1' cellpadding='2'⇒
 cellspacing='0' width='100%'>";
                  &strHTML = &strHTML | "<TR><TH> </TH>";
                  For &j = 1 To &QrySelOutputFldCol.count
                     &strHTML = &strHTML | "<TH><B>" | &QrySelOutputFldCol.Item⇒
(&j).LongName | "</B></TH>";
                  End-For;
                  &strHTML = &strHTML | "</TR>";
                  &strHTML = &strHTML | "</TD></TR>";
               End-If;
               &strHTML = &strHTML | "<TR><TD>";
               If &aRowSet.RowCount > 0 Then
                  For &i = 1 To &aRowSet.RowCount
                     &aRow = &aRowSet(&i);
                     &strHTML = &strHTML | "<TR>" | "<TD>" | &i | "</TD>";
                     For &j = 1 To &aRow.GetRecord(1).FieldCount
                        &strHTML = &strHTML | "<TD>" | &aRow.GetRecord(1).GetField⇒
(&j).Value | "</TD>";
                     End-For;
                     &strHTML = &strHTML | "</TR>";
                  End-For;
               Else
                  &strHTML = &strHTML | "No records retrieved.";
               End-If;
               &strHTML = &strHTML | "</TD></TR>";
               &strHTML = &strHTML | "</TABLE>";
               &strHTML = &strHTML | "</TABLE>";
            Else
               &strHTML = "Failed to retrieve result set.";
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
            End-If;
         End-If;
      End-If;
```

See Also

Chapter 30, "Query Classes," RunToFile, page 1270 and Chapter 30, "Query Classes," PromptRecord, page 1281.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Prompt

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Save

Syntax

```
Save()
```

Description

The Save method writes any changes to the existing query to the database.

The Save method can be used only on an open query, not on a closed query. This means you must have opened the query with the Open method before you can save it.

The query object remains open after executing Save. You must execute the Close method on the object before it is closed and the memory freed.

Note. If you're calling the Query API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

Parameters

None.

Returns

An integer value: 0 means the query was saved successfully.

See Also

Chapter 30, "Query Classes," Open, page 1268 and Chapter 30, "Query Classes," Close, page 1264.

Query Class Properties

In the following section, we discuss each Query class property.

Copyright © 1988-2007, Oracle. All rights reserved.

Approved

Description

This property returns a string indicating whether a query is approved, unapproved, or modified. This is useful for query administration. The values are:

Value	Description
U	Query is unapproved. This is the default value. The query administrator can prevent execution of unapproved queries.
A	Query has been approved by the query administrator.
M	Query has been modified.

This property is read-write.

ApproveOprId

Description

Note. This property has been deprecated, and remains for backward compatibility only. Use the ApproveUserId property instead.

This property returns a string containing the UserID of the user who approved the query. This can be useful for query administration.

This property is read-write.

See Also

Chapter 30, "Query Classes," ApproveUserId, page 1276

ApproveUserId

Description

This property returns a string containing the UserID of the user who approved the query. This can be useful for query administration.

This property is read-write.

ApproveDtTm

Description

This property returns the date-time stamp when the query was most recently approved. This can be useful for query administration.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

CreateOprId

Description

Note. This property has been deprecated, and remains for backward compatibility only. Use the CreateUserId property instead.

This property returns a string containing the User Id of the user who created the query. This can be useful for query administration.

This property is read-only.

See Also

Chapter 30, "Query Classes," CreateUserId, page 1277

CreateDtTm

Description

This property returns a string containing the date-time stamp indicating when the query was created. This property is set by the query runtime when a new query is saved. This can be useful for query administration.

This property is read-only.

CreateUserId

Description

This property returns a string containing the User Id of the user who created the query. This can be useful for query administration.

This property is read-only.

Description

Description

This property returns or sets the short description for the query.

The length of this property is 30 characters.

This property is read-write.

Disabled

Description

This property indicates if the query is active or not. This property returns a string value:

Value	Description
Y	This query is not active
Blank or any other value	This query is active

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

ExecAppName

Description

This property specifies the name of the application executing the query. This property isn't required. This means this property is used only if the query will be executed. This name is stored in the query execution log. This property returns a blank value when the query is opened.

When executing a query using the Query API, this property should be set to the Application name that's executing it, so that the Query Monitor can track query execution. Doing this makes this property useful for query administration. If you try to read this property before setting it, you receive a NULL string.

This property should be set before using RunToRowset or RunToFile.

This property is read-write. However, this property is generally used only to set the name, rather than to read it.

ExecLogging

Description

This property indicates whether an execution log should be created when the query is executed. This can be useful for query administration. This property takes a Boolean value: True, create a log, False, don't create one.

The execution log, which is created by setting this property, can be viewed from the Query Monitor. The logging is done in a PeopleSoft Table. It stores, along with other relevant information, the Execution DateTime, Total Execution Time for the query, and Total Fetch Time for the query.

If you try to read this property before setting it, you receive a NULL string.

This property is read-write. However, this property is generally used only to set logging, rather than read.

Folder

Description

This property is used to group queries together. This parameter takes a string value, up to 18 characters.

This property is read-write.

LastSQLErrorCode

Description

This property returns the last SQL error code returned by the database as a number. The session object contains the error text and any other errors encountered while executing the query.

PeopleSoft recommends checking this value after using RunToRowset or RunToFile.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

LastUpdDttm

Description

This property returns the most recent date-time when the query was updated as a string.

This property is read-only.

LastUpdOprId

Description

This property returns the User Id of the user who updated the query most recently as a string.

This property is read-only.

LongDescription

Description

This property returns or sets the long description for the query.

The length of this property is 256 characters.

This property is read-write.

Metadata

Description

This property returns a metadata collection.

This property is read-only.

Example

```
&MetadataList = &MyQuery.Metadata;
```

See Also

Chapter 30, “Query Classes,” Query Metadata Collection, page 1341

MetaSQL

Description

This property returns the SQL that represents the query, unresolved for any platform.

For example, the MetaSQL property returns the following:

```
SELECT %DATEOUT(A.ASOF_DT)
 FROM PS_AEREQUESTTBL A
 WHERE A.ASOF_DT = %DATEIN('1900-01-01')
```

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property returns the name of the query as a string.

The length of this property is 30 characters.

This property is read-only.

PDFFont

Description

This property is used to set the PDF Font number required for generating PDF using RunToFile. This property takes either a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
2	%PDFFont_TraditionalChinese	The output must be written using the Traditional Chinese Font.
3	%PDFFont_SimplifiedChinese	The output must be written using the Simplified Chinese Font.
1	%PDFFont_Japanese	The output must be written using the Japanese Font
4	%PDFFont_Korean	The output must be written using the Korean Font

If you try to read this property before setting it, you receive a NULL string.

This property is read-write.

See Also

Chapter 30, “Query Classes,” RunToFile, page 1270 and Chapter 30, “Query Classes,” RunToRowset, page 1272.

Prompts

Description

This property returns all the prompts defined for the Query in a QueryPrompt Collection. If you just want the prompts used in the criteria, use the RunTimePrompts property.

This property is read-only.

See Also

Chapter 30, “Query Classes,” RunTimePrompts, page 1282 and Chapter 30, “Query Classes,” QueryPrompt Collection, page 1361.

 Copyright © 1988-2007, Oracle. All rights reserved.

PromptRecord

Description

This property returns the runtime prompts of a query as an instance of a PeopleCode record object. This can be used with the Prompt built-in function to prompt for bind values. This record instance can also be used as the first input parameter for the RunToRowset and RunToFile methods.

This property is read-only.

See Also

Chapter 30, "Query Classes," RunToFile, page 1270 and Chapter 30, "Query Classes," RunToRowset, page 1272.

PublicPrivate

Description

This property specifies whether the query is public or private.

You can use either a constant or numeric value for this property. The values are:

Numeric Value	Constant Value	Description
0	%Query_Private	The query is a private query.
1	%Query_Public	The query is a public query.

This property is read-write.

Example

```
If &QryObj.PublicPrivate = %Query_Public Then
   /* code when working with Public Query */
Else
   /* code when working with Private Query */
End-if;
```

QuerySelect

Description

This property returns the Main Select (that is, the first select) of the query definition as a QuerySelect object. For a new query which does not have any selects, it returns NULL.

The Query Records, Query Criteria, QueryOutput fields, and QuerySelected Fields can be obtained using the QuerySelect object only. Hence, after opening a query, this property serves as the starting point for getting the different components of the Select statement.

This property is read-only.

See Also

Chapter 30, "Query Classes," QuerySelect Class, page 1283

Copyright © 1988-2007, Oracle. All rights reserved.

QueryStatistics

Description

This property returns statistical information pertaining to the query's execution as a QueryStatistics object.

This property is read-only.

See Also

Chapter 30, "Query Classes," QueryStatistics Class, page 1345

RunTimePrompts

Description

This property returns the prompts used in the Query Definition's criteria in a QueryPrompt Collection. To return all the prompts defined for the query, use the Prompts property.

This is property is read-only.

See Also

Chapter 30, "Query Classes," Prompts, page 1280 and Chapter 30, "Query Classes," QueryPrompt Collection, page 1361.

SQL

Description

This property returns the SQL statement as generated from the query definition as a character string.

Note. The value of the SQL property is never stored as part of the query definition. Instead, it's generated by the system whenever it's needed for one of the RunTo methods or for the SQL property.

In the PeopleSoft Query, this is located under the SQL tab.

This property is read-only.

Considerations Using the SQL Property

This property returns a basic SQL statement. This statement does not include logic for doing related language record transactions or translate descriptions. PeopleSoft does not recommend using this SQL for selecting data. Instead, PeopleSoft recommends using the RunToRowset method for selecting data.

Type

Description

This property specifies the type of query. This parameter takes either a constant or number value. Values are:

Numeric Value	Constant Value	Description
1	%Query_Query	Query of type Query
3	%Query_View	Query of type View
4	%Query_Role	Role

 Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
5	%Query_DBAgent	Process
7	%Query_Archive	Archive

This property is read-write.

QuerySelect Class

The QuerySelect represents the SELECT statement of the SQL used in the query definition. This can be any of the following:

- The main select statement
- The unions
- The subqueries

Consider the following SQL statement:

```
Select ACCOUNT_NUM from GL_ACCOUNT_TBL_00

Where PRODUCT_ID IN (SELECT DISTINCT PRODUCT_ID FROM ORDER_TBL)

Union

SELECT ACCOUNT_NUM from GL_ACCOUNT_TBL_01
```

In this example there are three select statements. Here, the first statement is the main statement, the second query is the SubQuery, which is the used in the criterion, and the union is the third select statement. There are fields and records associated with each select statement. In the Query API, the first Select is accessible using Query.QuerySelect, which can be obtained as shown:

```
&MainSelect = &Qry.QuerySelect;
```

The Union can be obtained from the main select, as shown:

```
/* There can be more than one union, hence there is collection of selects in the⇒
 main select */

&Union1 = &MainSelect.QuerySelects.Item(1);
```

The subquery is obtained from the main select as shown:

```
/* belongs to first criteria of the select */

&SubQry1 = &MainSelect.Criteria.Item(1). Expr2SubQuery;
```

Therefore, there is one QuerySelect instance for the main select of the Query, one instance per union defined in the query and one instance per sub-query.

In PeopleSoft Query, the first select (that is, the Main Select) is the parent of all the unions and sub-queries. The Main Select is obtained QuerySelect property. The unions and subqueries are obtained from the QuerySelects property of each QuerySelect instance.

Copyright © 1988-2007, Oracle. All rights reserved.

A QuerySelect object is returned from the following:

- The QuerySelect Collection methods AddUnion, First, Item, ItemBySelNum or Next
- The QuerySelect Query property

See Chapter 30, "Query Classes," QuerySelect Collection, page 1293 and Chapter 30, "Query Classes," QuerySelects, page 1292.

QuerySelect Methods

In this section, we discuss each QuerySelect method. The methods are arranged in alphabetical order.

AddAllFields

Syntax

AddAllFields(*QueryRecord*)

Description

The AddAllFields method adds all the fields in the specified query record to the query definition, that is, makes them all QueryOutputFields. The query record must already be a part of the query definition.

Parameters

Parameter	Description
QueryRecord	Specify the instance of an existing record in the query from which you want to add all the fields from to the query definition. This instance can be obtained from the QueryRecords collection of the QuerySelect.

Returns

An integer: 0 if successfully added.

See Also

Chapter 30, "Query Classes," AddQueryOutputField, page 1286, Chapter 30, "Query Classes," AddQueryRecord, page 1286 and Chapter 30, "Query Classes," AddQuerySelectedField, page 1287.

AddCriteria

Syntax

AddCriteria(*Name*)

Description

The AddCriteria method adds new criterion to the query definition. This method returns a reference to a new QueryCriteria object that you can then use to specify details about the criteria.

Note. If you do not specify a unique name for *Name* when adding a criteria, a criteria object referencing the new criteria is returned, not the existing criteria object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify a string containing the name of the criteria that you want to add. The maximum length of this parameter is 30 characters.

Returns

A reference to a QueryCriteria object.

See Also

Chapter 30, "Query Classes," DeleteCriteria, page 1288, Chapter 30, "Query Classes," Criteria, page 1290 and Chapter 30, "Query Classes," QueryCriteria Class, page 1314.

AddExpression

Syntax

AddExpression(*Name*)

Description

The AddExpression method adds an expression to the query definition. It returns a reference to a new QueryExpression object you can use to specify details about the expression.

Parameters

Parameter	Description
Name	Specify a string containing the name of the expression that you want to add. This maximum length of this parameter is 30 characters.

Returns

A reference to a QueryExpression object.

See Also

Chapter 30, "Query Classes," FindExpression, page 1267, Chapter 30, "Query Classes," DeleteExpression, page 1288, Chapter 30, "Query Classes," Expressions, page 1290 and Chapter 30, "Query Classes," QueryExpression Class, page 1331.

AddHavingCriteria

Syntax

AddHavingCriteria(*Name*)

Description

The AddHavingCriteria method adds new criterion to the query definition, which is intended for use in the Having Clause of the Select statement. This method returns a reference to a new QueryCriteria object that you can then use to specify details about the having criteria.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. If you do not specify a unique name for *Name* when adding a criterion, a criteria object referencing the new criteria is returned, not the existing criteria object.

Parameters

Parameter	Description
Name	Specify a string containing the name of the criteria that you want to add. The maximum length of this parameter is 30 characters.

Returns

A reference to a QueryCriteria object.

See Also

Chapter 30, "Query Classes," DeleteHavingCriteria, page 1289, Chapter 30, "Query Classes," HavingCriteria, page 1291 and Chapter 30, "Query Classes," QueryCriteria Class, page 1314.

AddQueryOutputField

Syntax

```
AddQueryOutputField(QueryRecord, index)
```

Description

The AddQueryOutputField method adds a query output field to the query definition. This method returns a reference to a new QueryOutputField object that you can then use to specify attributes for the query definition.

Parameters

Parameter	Description
QueryRecord	Specify the QueryRecord instance that contains the field that you want to include in the query definition. This instance can be obtained from the QueryRecords collection of the QuerySelect.
Index	Specify the numeric position in the QueryRecord of the field that you want to add.

Returns

A reference to a QueryOutputField object.

See Also

Chapter 30, "Query Classes," AddAllFields, page 1284 and Chapter 30, "Query Classes," QueryField Class, page 1304.

AddQueryRecord

Syntax

```
AddQueryRecord(QueryRecordName)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The AddQueryRecord method adds a query record to the query definition. You must specify a valid record name in the string parameter *QueryRecordName*. This method returns a reference to the record as a QueryRecord.

Parameters

Parameter	Description
QueryRecordName	Specify a string containing the name of a record to be added to the query definition. You must specify a valid record name. The maximum allowed length for this parameter is 15 characters.

Returns

A reference to the record as a QueryRecord object.

See Also

Chapter 30, "Query Classes," QueryRecord Class, page 1299

AddQuerySelectedField

Syntax

AddQuerySelectedField(*QueryRecordName, RecordAlias, FieldName, Heading*)

Description

The AddQuerySelectedField method adds a query field to the query definition, as a QuerySelectedField. This method returns a reference to a new QuerySelectedField object that you can then use to specify attributes for the query definition. At the time it's added, it isn't an output field. This can be changed by setting the column number of the field to a value greater than zero.

Parameters

Parameter	Description
QueryRecordName	Specify the name of the query record to which the field that you want to add belongs.
RecordAlias	Specify the alias of the record (such as A, B, C, and so on.) The length of this parameter is 1.
FieldName	Specify the name of the field that's being added. The maximum allowed length for a field name is 18 characters.
Heading	Specify the heading of the field that's being added. The maximum allowed length for a heading is 30 characters.

Returns

A reference to a QuerySelectedField object.

See Also

Chapter 30, "Query Classes," AddAllFields, page 1284, Chapter 30, "Query Classes," DeleteField, page 1288 and Chapter 30, "Query Classes," QueryField Class, page 1304.

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteCriteria

Syntax

DeleteCriteria(*index*)

Description

The DeleteCriteria method deletes the criteria specified by *index* from the query definition.

Parameters

Parameter	Description
Index	Specify the numeric position in the query definition of the criteria that you want to delete.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 30, "Query Classes," AddCriteria, page 1284 and Chapter 30, "Query Classes," Criteria, page 1290.

DeleteExpression

Syntax

DeleteExpression(*index*)

Description

The DeleteExpression method deletes the specified expression from the query definition.

Parameters

Parameter	Description
Index	Specify the numeric position of the expression in the query definition that you want to delete.

Returns

An integer: 0 if successfully deleted.

See Also

Chapter 30, "Query Classes," AddExpression, page 1285, Chapter 30, "Query Classes," Expressions, page 1290 and Chapter 30, "Query Classes," QueryExpression Class, page 1331.

DeleteField

Syntax

DeleteField(*index*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The DeleteField method deletes the selected field from the query definition. This does *not* delete the field from the QueryRecord, just from the query definition, so it's no longer selected.

Parameters

Parameter	Description
Index	Specify the position number of the field that you want deleted from the query definition.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 30, "Query Classes," AddAllFields, page 1284 and Chapter 30, "Query Classes," QueryField Class, page 1304.

DeleteHavingCriteria

Syntax

DeleteHavingCriteria(*index*)

Description

The DeleteHavingCriteria method deletes the Having Criteria specified by *index* from the query definition.

Parameters

Parameter	Description
Index	Specify the numeric position in the query definition of the having criteria that you want to delete.

Returns

An integer value: 0 means the criteria was deleted successfully.

See Also

Chapter 30, "Query Classes," AddHavingCriteria, page 1285 and Chapter 30, "Query Classes," HavingCriteria, page 1291.

DeleteRecord

Syntax

DeleteRecord(*index*)

Description

The DeleteRecord method deletes the selected record from the query definition. This does *not* delete the record from the database or the QueryDBRecord collection, just from the query definition, so it's no longer selected.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Index	Specify the position number of the record you want deleted from the query definition.

Returns

An integer value: 0 means the query was deleted successfully.

See Also

Chapter 30, "Query Classes," AddQueryRecord, page 1286 and Chapter 30, "Query Classes," QueryField Class, page 1304.

QuerySelect Properties

In this section, we discuss the QuerySelect properties. The properties are discussed in alphabetical order.

Criteria

Description

This property returns a reference to a QueryCriteria Collection for the simple criteria of the SELECT (that is, the criteria that are used in the where clause of the select statement.)

This property is read-only.

See Also

Chapter 30, "Query Classes," QueryCriteria Collection, page 1312

Distinct

Description

This property specifies if the Select is distinct or not.

This property takes a Boolean value: True if the query is distinct, False if the query isn't distinct.

This property is read-write.

Expressions

Description

This property returns a reference to a QueryExpression Collection.

This property is read-only.

See Also

Chapter 30, "Query Classes," QueryExpression Collection, page 1328

 Copyright © 1988-2007, Oracle. All rights reserved.

HavingCriteria

Description

This property returns a reference to a QueryCriteria collection populated with Having criteria.

This property is read-only.

See Also

Chapter 30, "Query Classes," QueryCriteria Collection, page 1312

ParentSelectNum

Description

This property returns the select number of the parent select of the current select object. Every QuerySelect is assigned a unique number indicating its place in the hierarchy of select statements. For the Main Select, this number is zero. For unions, this number is 1 (unions aren't allowed in subqueries.)

This property is read-only.

QueryOutputFields

Description

This property returns a reference to a QueryField Collection made up of QueryOutputFields.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the Query Classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

This is property is read-only.

See Also

Chapter 30, "Query Classes," QueryField Collection, page 1301

QueryRecords

Description

This property returns a reference to a QueryRecord Collection for the query.

This is property is read-only.

See Also

Chapter 30, "Query Classes," QueryRecord Collection, page 1296

Copyright © 1988-2007, Oracle. All rights reserved.

QuerySelectedFields

Description

This property returns a reference to a QueryField Collection made up of QuerySelectedFields.

This is property is read-only.

See Also

Chapter 30, "Query Classes," QueryField Collection, page 1301

QuerySelects

Description

This property returns all the Query Selects that are children of the current Select Statement as a QuerySelect Collection. If the query contains Unions, the QuerySelects of the First Select contains Unions and sub-queries (if any criteria contain sub-queries). For all other selects, this property contains only the sub-queries (because PeopleSoft Query doesn't allow unions for sub-queries).

This is property is read-only.

See Also

Chapter 30, "Query Classes," QuerySelect Collection, page 1293

SelectNum

Description

This property returns the select number of the current select. Every QuerySelect is assigned a unique number indicating its place in the hierarchy of select statements. For the Main Select, this number would be 1.

This property is read-only.

SelectType

Description

This property specifies what type of select statement. You can check for either a numeric or a constant value. The values are:

Numeric Value	Constant Value	Description
1	%Query_SelectMain	This is the Main Select of the query definition.
2	%Query_SubQuery	This is a subquery in the query definition.
3	%Query_Union	This is a Union in the query definition.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

QuerySelect Collection

A QuerySelect Collection is returned from the QuerySelects property of each QuerySelect instance. It contains the unions and sub-queries, which are children of the current Select statement. PeopleSoft Query allows Unions only with the Main Select, the QuerySelect instance for any subqueries will not have any Unions.

See Chapter 30, "Query Classes," QuerySelects, page 1292.

QuerySelect Collection Methods

In this section, we discuss the QuerySelect collection methods. The methods are discussed in alphabetical order.

AddUnion

Syntax

AddUnion()

Description

The AddUnion method adds a new QuerySelect object of type Union in the current QuerySelect collection. You can use AddUnion only with the first QuerySelect Collection because PeopleSoft Query doesn't' support Unions for subqueries. All unions are considered children of the Main Select.

Parameters

None.

Returns

A reference to a QuerySelect object if successful. If the current QuerySelect Collection does not belong to the first Select statement, it returns NULL. If it fails, it returns NULL.

Example

```
&QryMainSel = &Query.QuerySelect;
&QryMainSelCol = &QryMainSel.QuerySelects;
&QryUnion = &QryMainSelCol.AddUnion();
```

DeleteQuerySelect

Syntax

DeleteQuerySelect(*SelNumber*)

Description

The DeleteQuerySelect method deletes the QuerySelect instance represented *SelNumber*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
SelNumber	Specify the numeric value containing the Select Number of the QuerySelect that you want to delete.

Returns

Returns 0 if successful.

Example

```
&QryMainSel = &Query.QuerySelect;
&QryMainSelCol = &QryMainSel.QuerySelects;
&QryMainSelCol.DeleteQuerySelect(3);
```

First

Syntax

First()

Description

The First method returns the first child QuerySelect object in the current QuerySelect's collection.

Parameters

None.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
&MyChildQrySel = &MySelCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QuerySelect object that exists at the *number* position in the current QuerySelect collection.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QuerySelect object that you want returned.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
For &I = 1 to &QrySelCol.Count;
   &MyChildQrySel = &QrySelCol.Item(&I);
   /* do processing */
End-For;
```

ItemBySelNum

Syntax

ItemBySelNum(*SelNumber*)

Description

The ItemBySelNum method returns the QuerySelect object specified by *the Select Number*. Each Select Statement in a Query Definition is identified by a unique select number.

Parameters

Parameter	Description
SelNumber	Specify the numeric value of the Select Number.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
&MyChildQuerySelect = &MySelCol.ItemBySelNum(3);
```

Next

Syntax

Next()

Description

The Next method returns the next QuerySelect object in the current QuerySelect collection. You can use this method only after you have used the First method: otherwise the system returns a NULL.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to a QuerySelect object if successful, NULL otherwise.

Example

```
&MyChildQuerySelect = &MySelCol.Next();
```

QuerySelect Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the total number of QuerySelect objects in the current QuerySelect Collection, as a number.

This property is read-only.

Example

```
&COUNTCHILDSELS = &MYSELCOL.Count;
```

QueryRecord Collection

A QueryRecord collection is returned from the QueryRecords QuerySelect class method.

See Chapter 30, “Query Classes,” QueryRecords, page 1291.

QueryRecord Collection Methods

In this section, we discuss each QueryRecord collection method. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The First method returns the first QueryRecord object in the QueryRecord collection.

Parameters

None.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
&MyQueryRecord = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryRecord object that exists at the *number* position in the QueryRecord collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryRecord object that you want returned.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryRecordColl.Count;
   &MyQueryRecord = &QueryRecordColl.Item(&I);
   /* do processing */
End-For;
```

ItemByAlias

Syntax

ItemByAlias(*Alias*)

Description

The ItemByAlias method returns the QueryRecord object specified by *Alias*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Alias	Specify the record alias (for example, A, B, C, and so on), used in the SQL statement as a TableName alias.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
&MyQueryRecord = &MyCollection.ItemByAlias("A");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryRecord object in the QueryRecord collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryRecord object if successful, NULL otherwise.

Example

```
&MyQueryRecord = &MyCollection.Next();
```

QueryRecord Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryRecord objects in the QueryRecord Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

QueryRecord Class

A QueryRecord object is returned from the following:

- The QueryRecord Collection methods First, Item, ItemByName, or Next.
- The AddQueryRecord QuerySelect class method.
- The QueryRecord QueryField class method.

See Chapter 30, "Query Classes," QueryRecord Collection, page 1296; Chapter 30, "Query Classes," AddQueryRecord, page 1286 and Chapter 30, "Query Classes," QueryRecord, page 1310.

QueryRecord Class Method

In this section, we discuss the GetField method.

GetField

Syntax

GetField(*Index*)

Description

The GetField method returns the QueryField object specified by *index* from the QueryRecord collection.

Parameters

Parameter	Description
Index	Specify the number of the field to be accessed.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&QryFieldColl = &QryRec.QueryFields;

For &I = 1 to &QryFieldColl.Count
   &QryField = &QryRec.GetField(&I);
   /* do processing */
End-For;
```

QueryRecord Class Properties

In this section, we discuss each QueryRecord class property. The properties are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Description

This property returns the short description of the record as a string.

The length of this property is 30 characters.

This property is read-only.

JoinAlias

Description

This property returns or sets the join record's alias, that is, A, B, C, and so on.

This is applicable in any type of join. The value is a string. The length of this property is 1 character.

This property is read-write.

JoinFieldName

Description

This property returns or sets the join record's field used in the join criteria. This is applicable in lookup-table joins. The value is a string.

The length of this property is 18 characters.

This property is read-write.

JoinType

Description

This property returns or sets the join type. This is applicable in any type of join. You can use either a constant or numeric value.

The values for the join type are:

Numeric Value	Constant Value	Description
1	%Query_JoinNone	Query record isn't joined with any other record.
2	%Query_JoinHierarchy	Query record has a hierarchy join with another record.
3	%Query_JoinRelated	Query record has a lookup table join with another record.
4	%Query_JoinTree	Query record has a tree join with another record.
5	%Query_JoinLeftOuter	Query record has a left outer join with another record.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property returns the name of the record as a string.

The length of this property is 15 characters.

This property is read-only.

QueryFields

Description

This property returns a reference to a QueryField Collection that contains instances of all the fields belonging to the record definition.

This property is read-only.

See Also

Chapter 30, “Query Classes,” QueryField Collection, page 1301

RecordAlias

Description

This property returns the alias used for the record in the query (that is, A, B, C, and so on.)

The length of this property is 1 character.

This property is read-write.

QueryField Collection

A QueryField collection is returned from the following:

- The QueryOutputFields of QuerySelect class method.
- The QuerySelectedFields of QuerySelect class method.
- The QueryFields QueryRecord class method.

The collection of QueryOutputFields does *not* necessarily include all columns returned in query resultset. This collection only includes columns that have been added to query output collection using the Query Classes or by an end-user designing a query.

PeopleSoft Query can also add additional columns for related language processing and also for translate labels on fields, and so you cannot use the Query Output Collections as a way to discover all of the columns that are returned in the resultset or by executing the SQL generated from a query.

See Chapter 30, “Query Classes,” QueryOutputFields, page 1291; Chapter 30, “Query Classes,” QuerySelectedFields, page 1240 and Chapter 30, “Query Classes,” QueryFields, page 1301.

Copyright © 1988-2007, Oracle. All rights reserved.

QueryField Collection Methods

In this section, we discuss each QueryField method. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryField object in the QueryField collection.

Parameters

None.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&MyQueryField = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryField object that exists at the *number* position in the QueryField collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryField object that you want returned.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryFieldColl.Count;
   &MyQueryField = &QueryFieldColl.Item(&I);
   /* do processing */
End-For;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ItemByNameAndAlias

Syntax

ItemByNameAndAlias(*Name, RecordAlias*)

Description

The ItemByNameAndAlias method returns the QueryField object with the given *Name* and *Record Alias*.

Parameters

Parameter	Description
Name	Specify the name of an existing QueryField within the QueryField collection. If you specify an invalid name, the object is NULL.
RecordAlias	Alias of the record to which the field belongs

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&MyQueryField = &MyCollection.ItemByNameAndAlias("PHONELIST", "A");
```

ItemByExpNum

Syntax

ItemByExpNum(*ExpNum*)

Description

The ItemByExpNum method returns the QueryField object with the given Expression Number, *ExpNum*.

Parameters

Parameter	Description
ExpNum	Expression Number for the given expression field. Non-expression fields have 0 value for the Expression Number.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&QryFld = &QryFldCol.ItemByExpNum(1);
```

Next

Syntax

Next()

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Next method returns the next QueryField object in the QueryField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryField object if successful, NULL otherwise.

Example

```
&MyQueryField = &MyCollection.Next();
```

QueryField Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryField objects in the QueryField Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryField Class

A QueryField object is returned by the following:

- The AddQuerySelectedField of QuerySelect class method.
- The AddQueryOutputField of QuerySelect class method.
- The QueryField Collection methods First, Item, ItemByName or Next.

See Chapter 30, "Query Classes," AddQuerySelectedField, page 1287 and Chapter 30, "Query Classes," AddQueryOutputField, page 1286.

See Chapter 30, "Query Classes," QueryField Collection, page 1301.

 Copyright © 1988-2007, Oracle. All rights reserved.

QueryField Methods

In this section, we discuss each QueryField method. The methods are discussed in alphabetical order.

AddTranslateExpression

Syntax

AddTranslateExpression(*ExpressionName*)

Description

Use the AddTranslateExpression method to add Translate Effective Date Logic expressions for translatable fields.

Parameters

Parameter	Description
ExpressionName	Specify the name of the expression name that you want to add.

Returns

A reference to a QueryExpression object.

See Also

Chapter 30, "Query Classes," Expressions, page 1290, Chapter 30, "Query Classes," DeleteExpression, page 1288, Chapter 30, "Query Classes," AddExpression, page 1285, Chapter 30, "Query Classes," AddTranslateField, page 1305 and Chapter 30, "Query Classes," QueryExpression Class, page 1331.

AddTranslateField

Syntax

AddTranslateField(*FieldName*)

Description

Use the AddTranslateField method to add a Translate Effective Date Logic field for a translatable field.

Parameters

Parameter	Description
FieldName	Specify the name of the translate field that you want to add. The maximum allowed length of this parameter is 18 characters.

Returns

A reference to a QueryField object.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 30, "Query Classes," Expressions, page 1290, Chapter 30, "Query Classes," DeleteExpression, page 1288, Chapter 30, "Query Classes," AddExpression, page 1285, Chapter 30, "Query Classes," AddTranslateExpression, page 1305 and Chapter 30, "Query Classes," QueryExpression Class, page 1331.

QueryField Class Properties

In this section, we discuss the QueryField class properties. The properties are discussed in alphabetical order.

Aggregate

Description

This property returns or sets the aggregate type for the query field.

This property takes either a constant or numeric value. The values are:

Numeric Value	Constant Value	Description
1	%Query_TtlNone	Query field has no total
2	%Query_TtlSum	Query field is for Sum
3	%Query_TtlCount	Query field is for Count
4	%Query_TtlMin	Query field is for Minimum
5	%Query_TtlMax	Query field is for Maximum
6	%Query_TtlAvg	Query field is for Average

This property is read-write.

ColumnNumber

Description

This property returns or sets the column number for the query field as a number. If the column number is greater than zero, the field is an output field.

This property is read-write.

Description

Description

This property returns the long description of the field as a string. This same value is returned by the Description property of the QueryDBRecordField class.

The length of this property is 30 characters.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Decimal

Description

This property returns the decimal positions QueryField as a number. This same value is returned by the Decimal property of the QueryDBRecordField class.

This property is read-only.

ExpNum

Description

This property returns or sets the Expression Number for the field. This value is 0 for non-expression fields. If the field is created for an expression, this value is the same as the expression number of the corresponding expression. This property takes a number value.

This property is read-write.

Flag

Description

This property returns the Use Edit flag for the Query Field. It has the same values as returned by the Flag property of the QueryDBRecordField. This property takes a number value.

This property is read-only.

Format

Description

This property returns the field format for the query field. This property returns a string value. This property is useful for displaying the datatype in a string format.

Values are:

Data Type	Value Returned
Character	“CHAR “
Long Character	"LONG CHAR "
Number	"NUMBER "
Signed number	"SIGNED "
Date	"DATE "
Time	"TIME "

Copyright © 1988-2007, Oracle. All rights reserved.

Data Type	Value Returned
Datetime	"DATE/TIME "
Image	"IMAGE "

This property is read-only.

HeadingText

Description

This property returns or sets the heading text for the query field as a string.

The length of this property is 30 characters.

This property is read-write.

HeadingType

Description

This property returns or sets the heading type for the query field. This property takes either a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
1	%Query_HdgNone	Query field has no heading.
2	%Query_HdgText	Query field has a text heading.
3	%Query_HdgRftShort	Query field uses the short RFT heading.
4	%Query_HdgRftLong	Query fields uses the long RFT heading.

This property is read-write.

HeadingUniqueFieldName

Description

This property returns or sets the unique field name for the heading. The default value for this property is the record alias combined with the field name.

The length of this property is 30 characters.

This property is read-write.

Length

Description

This property returns the length of the Query Field. The same value is returned by the Length property of the QueryDBRecordField. This property takes a numeric value.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

LongName

Description

This property returns the long name of the Query Field. The same value is returned by the Long Name property of the QueryDBRecordField. This property takes a string value.

The length of this property is 30 characters.

This property is read-only.

Name

Description

This property returns the name of the query field as a string.

The length of this property is 18 characters.

This property is read-only.

OrderByDirection

Description

This property returns or sets the order by direction. This property takes a numeric value. The constants are the ASCII codes for space (" ") and "D".

Values are:

Numeric Value	Constant Value	Description
32	Code (" ")	Ascending
68	Code("D")	Descending

This property is read-write.

Example

The following sets the order by direction to be ascending.

```
&QueryField.OrderByDirection = Code(" ");
```

The following sets the order by direction to be descending.

```
&QueryField.OrderByDirection = Code("D");
```

OrderByNumber

Description

Use this property to specify whether a field is used as part of an 'Order By' statement in the SQL. The number value of this property indicates which is the first field in the order by statement, which is the second, and so on.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

To order a query by one field, you must set the OrderByNumber property for the other QueryFields as well, specifying the primary field as 1, the next field as 2, and so on.

```
&QryFld = &MainQrySel.AddQuerySelectedField("PSRECDEFN", "A", "RECNAME", "Record⇒
 Name");
&QryFld.ColumnNumber = 1;
&QryFld.OrderByNumber = 1;

&QryFld = &MainQrySel.AddQuerySelectedField("PSRECDEFN", "A", "RECDESCR", "Record⇒
 Descr");
&QryFld.ColumnNumber = 2;
&QryFld.OrderByNumber = 2;

&QryFld = &MainQrySel.AddQuerySelectedField("PSRECDEFN", "A", "RELLANGRECNAME",⇒
 "Record Lang Rec");
&QryFld.ColumnNumber = 3;
&QryFld.OrderByNumber = 3;
```

QueryRecord

Description

This property returns a reference to the QueryRecord containing this QueryField. If the field has no QueryRecord (that is, that this field is an expression field), this property returns NULL.

This property is read-only.

Example

```
&QryRcd = &QryFld.QueryRecord;
```

RecordAlias

Description

This property returns the record alias for the Query Field as a string. This value is usually a value like "A", "B", and so on. The same value is returned by the RecordAlias property for QueryRecord.

The length of this property is 1 character.

This property is read-only.

ShortName

Description

This property returns the short name of the Query Field. The same value is returned by the ShortName property for QueryDBRecordField. This property takes a string value.

The length of this property is 15 characters.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

TranslateEffDtLogic

Description

This property returns or sets the effective date logic for the QueryField.

The values are:

Numeric Value	Constant Value	Description
1	%Query_ExprCurDt	Current Date
2	%Query_ExprField	Field
3	%Query_Expression	Expression

This property is read-write.

TranslateExpression

Description

This property returns a reference to an expression object based on a translate field if the Translate Effective Date option refers to an Expression

This property is read-only.

See Also

Chapter 30, “Query Classes,” QueryExpression Class, page 1331

TranslateField

Description

This property returns a reference to a QueryField object for a translate field if the Translate Effective Date Option refers to a field.

This property is read-only.

See Also

Chapter 30, “Query Classes,” QueryField Class, page 1304

TranslateOption

Description

This property returns or sets the translate value for the QueryField. This property takes a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
1	%Query_XlatNone	None
2	%Query_XlatShort	Short
3	%Query_XlatLong	Long

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Type

Description

This property returns the type of the field. You can use either a numeric or constant value. The values are:

Numeric Value	Constant Value	Description
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	Datetime
7	%FieldType_Image	Image

This property is read-only.

QueryCriteria Collection

A QueryCriteria collection is returned from the Criteria QuerySelect class property.

See Chapter 30, "Query Classes," Criteria, page 1290.

QueryCriteria Collection Methods

In this section, we discuss the QueryCriteria collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first QueryCriteria object in the QueryCriteria collection.

Parameters

None.

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyQueryCriteria = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryCriteria object that exists at the *number* position in the QueryCriteria collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryCriteria object that you want returned.

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryCriteriaColl.Count;
   &MyQueryCriteria = &QueryCriteriaColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*CriteriaName*)

Description

The ItemByName method returns the QueryCriteria object that exists with the passed CriteriaName in the QueryCriteria collection.

Parameters

Parameter	Description
CriteriaName	Specify the name of the Criteria to be searched. This name is the same as the one used while creating the criteria using QuerySelect.AddCriteria(CriteriaName).

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
For &I = 1 to &QueryCriteriaColl.Count;
   &MyQueryCriteria = &QueryCriteriaColl.Item(&I);
   /* do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next QueryCriteria object in the QueryCriteria collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryCriteria object if successful, NULL otherwise.

Example

```
&MyQueryCriteria = &MyCollection.Next();
```

QueryCriteria Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryCriteria objects in the QueryCriteria Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryCriteria Class

A QueryCriteria object is returned from the following:

 Copyright © 1988-2007, Oracle. All rights reserved.

- The AddCriteria and AddHavingCriteria QuerySelect class method.
- The First, Item, and Next method of the QueryCriteria Collection.

See Chapter 30, "Query Classes," AddCriteria, page 1284 and Chapter 30, "Query Classes," AddHavingCriteria, page 1285.

See Chapter 30, "Query Classes," QueryCriteria Collection, page 1312.

See Chapter 30, "Query Classes," Working With Criteria and Expressions, page 1241.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query*, "Defining Selection Criteria".

QueryCriteria Class Methods

In this section, we discuss the QueryCriteria class methods. The methods are discussed in alphabetical order.

AddExpr1Expression

Syntax

AddExpr1Expression()

Description

The AddExpr1Expression method returns a reference to a new a QueryExpression object to be used as an expression for Expression 1. You can then use this object to specify details about the expression using the methods and properties of the QueryExpression Class.

Note. You must set the type for Expression 1 using the Expr1Type property before you can use this method.

Parameters

None.

Returns

A reference to a blank QueryExpression object.

See Also

Chapter 30, "Query Classes," Expr1Expression, page 1319, Chapter 30, "Query Classes," AddExpr1Field, page 1315, Chapter 30, "Query Classes," Expr1Type, page 1319 and Chapter 30, "Query Classes," QueryExpression Class, page 1331.

AddExpr1Field

Syntax

AddExpr1Field(*QueryRecordAlias*, *FieldName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The AddExpr1Field method returns a reference to a new a QueryField object to be used as a field for Expression 1. You can then use this object to specify details about the expression using the methods and properties of the QueryField Class.

Note. You must set the type for Expression 1 using the Expr1Type property before you can use this method.

Parameters

Parameter	Description
QueryRecordAlias	Specify the alias of the QueryRecord that contains the QueryField that you want to use.
FieldName	Specify the name of the QueryField in the QueryRecord that you want to use.

Returns

A reference to a blank QueryField object.

See Also

Chapter 30, "Query Classes," Expr1Field, page 1319, Chapter 30, "Query Classes," AddExpr1Expression, page 1315, Chapter 30, "Query Classes," Expr1Type, page 1319 and Chapter 30, "Query Classes," QueryField Class, page 1304.

AddExpr2Expression

Syntax

```
AddExpr2Expression()
```

Description

The AddExpr2Expression method returns a reference to a new a QueryExpression object to be used as an expression for Expression 2. You can then use this object to specify details about the expression using the methods and properties of the QueryExpression Class.

Parameters

None.

Returns

A reference to a blank QueryExpression object.

See Also

Chapter 30, "Query Classes," QueryExpression Class, page 1331

AddExpr2Field1

Syntax

```
AddExpr2Field1(QueryRecordAlias, FieldName)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Expression 2 has two field expressions because when the Between relational operator is used in a criteria, there can be two expression fields. The AddExpr2Field1 method returns a reference to a new QueryField object to be used as a *field 1* for Expression 2. You can then use this object to specify details about the expression using the methods and properties of the QueryField Class.

Parameters

Parameter	Description
QueryRecordAlias	Specify the alias of the QueryRecord that contains the QueryField that you want to use.
FieldName	Specify the name of the QueryField in the QueryRecord that you want to use.

Returns

A reference to a blank QueryField object.

See Also

Chapter 30, "Query Classes," QueryRecord Class, page 1299 and Chapter 30, "Query Classes," QueryField Class, page 1304.

AddExpr2Field2

Syntax

```
AddExpr2Field2(QueryRecordAlias, FieldName)
```

Description

Expression 2 has two field expressions because when the Between relational operator is used in a criteria, there can be two expression fields. The AddExpr2Field2 method returns a reference to a new a QueryField object to be used as a *field 2* for Expression 2. You can then use this object to specify details about the expression using the methods and properties of the QueryField Class.

Parameters

Parameter	Description
QueryRecordAlias	Specify the alias of the QueryRecord that contains the QueryField that you want to use.
FieldName	Specify the name of the QueryField in the QueryRecord that you want to use.

Returns

A reference to a blank QueryField object.

See Also

Chapter 30, "Query Classes," QueryRecord Class, page 1299 and Chapter 30, "Query Classes," QueryField Class, page 1304.

Copyright © 1988-2007, Oracle. All rights reserved.

AddExpr2List

Syntax

AddExpr2List()

Description

Expression 2 can have a list of values when the Operator is of type In List (or Not In List). The AddExpr2List method returns a reference to a new QueryList, which can be used to add a list of values to the criteria.

Parameters

None.

Returns

A reference to a blank QueryList object.

See Also

Chapter 30, "Query Classes," QueryList Class, page 1334

AddExpr2Subquery

Syntax

AddExpr2Subquery()

Description

The AddExpr2Subquery method is used to create a subquery for Expression2. This method returns a new QuerySelect object you can use to specify details about the new subquery.

Warning! The new subquery created with this method replaces any existing subquery (for this criteria), destroying any existing properties or values.

Parameters

None.

Returns

A reference to a new QuerySelect object.

See Also

Chapter 30, "Query Classes," QuerySelect Class, page 1283

QueryCriteria Class Properties

In this section, we discuss the QueryCriteria class properties. The properties are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Expr1Expression

Description

This property returns a reference to the QueryExpression object that's used as Expression 1.

This property is valid only when Expression 1 exists as an expression. If you want to add an expression for Expression 1, use the AddExpr1Expression method.

This property is read-only.

See Also

Chapter 30, "Query Classes," AddExpr1Expression, page 1315

Expr1Field

Description

This property returns a reference to the QueryField object that's used as Expression 1.

This property is valid only when Expression 1 exists as a field. You can then use the QueryField Class methods and property to manipulate this object.

If you want to add a field for Expression 1, use the AddExpr1Field method.

This property is read-only.

See Also

Chapter 30, "Query Classes," QueryField Class, page 1304 and Chapter 30, "Query Classes," AddExpr1Field, page 1315.

Expr1Type

Description

This property returns or sets the type for Expression 1.

Note. You *must* set the type of expression for every new criteria.

The values are:

Numeric Value	Constant Value	Description
2	%Query_ExprField	Field
3	%Query_Expression	Expression

This property is read-write.

Example

The following is used to test the expression to determine the property to use to retrieve it:

```
&MyExpr1 = &MyQueryColl.Next();
If &MyExpr1.Expr1Type = %Query_ExprField Then /* Expression is a Field */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   &OldValue = &MyExpr1.Expr1Field;
   /* do processing */

Else  /* Expression 1 is an expression */

   &OldValue = &MyExpr1.Expr1Expression;
   /* Do processing */

End-if;
```

The following is an example showing how to add a field for Expression 1.

```
/* add a new criteria */
&MyCriteria = &MyQuery.AddCriteria();

/* set the type of the first expression to be a field */
&MyCriteria.Expr1Type = %Query_ExprField;

/* add the field EMPLID from the ABSENCE_HIST record */
&MyField = &MyCriteria.AddExpr1Field("A", "EMPLID");
```

Expr2Constant1

Description

If the Between relational operator is used in the criteria, there can be two constants for Expression 2. This property returns or sets the constant value for the first constant for Expression 2. This property takes a string value.

This property is valid only when Expression 2 is defined as a constant.

This property is read-write.

Expr2Constant2

Description

If the Between relational operator is used in the criteria, there can be two constants for Expression 2. This property returns or sets the constant value for the second constant for Expression 2. This property takes a string value.

This property is valid only when Expression 2 is defined as a constant.

This property is read-write.

Expr2Expression1

Description

If the Between relational operator is used in the criteria, there can be two expressions for Expression 2. This property returns a reference to the first QueryExpression object that's used as Expression 2.

This property is valid only when Expression 2 exists as an expression. To add an expression for Expression 2, use the AddExpr2Expression method.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

See Also
Chapter 30, “Query Classes,” AddExpr2Expression, page 1316

Expr2Expression2

Description

If the Between relational operator is used in the criteria, there can be two expressions for Expression 2. This property returns a reference to the second QueryExpression object that’s used as Expression 2.

This property is valid only when Expression 2 exists as an expression. To add an expression for Expression 2, use the AddExpr2Expression method.

This property is read-only.

See Also
Chapter 30, “Query Classes,” AddExpr2Expression, page 1316

Expr2Field1

Description

If the Between relational operator is used in the criteria, there can be two fields for Expression 2. This property returns a reference to the first QueryField object that’s used as Expression 2.

This property is only valid when Expression 2 exists as a field. You can then use the QueryField Class methods and property to manipulate this object.

To add a field for Expression 2, use the AddExpr2Field1 method.

This property is read-only.

See Also
Chapter 30, “Query Classes,” AddExpr2Field1, page 1316 and Chapter 30, “Query Classes,” QueryField Class, page 1304.

Expr2Field2

Description

If the Between relational operator is used in the criteria, there can be two fields for Expression 2. This property returns a reference to the second QueryField object that’s used as Expression 2.

This property is valid only when Expression 2 exists as a field. You can then use the QueryField Class methods and property to manipulate this object.

To add a field for Expression 2, use the AddExpr2Field2 method.

This property is read-only.

See Also
Chapter 30, “Query Classes,” AddExpr2Field2, page 1317 and Chapter 30, “Query Classes,” QueryField Class, page 1304.

Copyright © 1988-2007, Oracle. All rights reserved.

Expr2List

Description

This property returns a reference to the QueryList object of Expression 2 that's used when the Operator is of type In List (or Not In List).

This property is read-only.

Expr2Subquery

Description

This property returns a reference to the QuerySelect object that's used as a subquery for Expression 2.

This property is valid only when Expression 2 exists as a subquery. To add a subquery for Expression 2, use the AddExpr2Subquery method.

This property is read-only.

See Also

Chapter 30, "Query Classes," AddExpr2Subquery, page 1318

Expr2Type

Description

This property returns or sets the type for Expression 2. The following table lists all of possible values for this property. However, the values for this property are dependent upon the Operator property.

This property is read-write.

See Chapter 30, "Query Classes," Operator, page 1326 and Chapter 30, "Query Classes," Working With Criteria and Expressions, page 1241.

You can use either a constant or numeric value for this property. The values are:

Numeric Value	Constant Value	Description
1	%Query_ExprConstant	Constant
2	%Query_ExprField	Field
3	%Query_Exprression	Expression
4	%Query_ExprSubQuery	Subquery
5	%Query_ExprList	List
6	%Query_ExprCurDt	Current date
7	%Query_ExprTree	Tree
8	%Query_ExprBind	Bind
9	%Query_ExprBothConst	The criterion's operator is Between and both values on the right-hand side are constants. (Const-Const)
10	%Query_ExprConstFld	The criterion's operator is Between, the first value on right-hand side is a constant and the second value is a field. (Const-Field)

 Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
11	%Query_ExprConstExpr	The criterion's operator is Between, the first value on right-hand side is a constant and the second value is an expression. (Const-Expr)
12	%Query_ExprFieldConst	The criterion's operator is Between, the first value on right-hand side is a field and the second value is a constant. (Field-Const)
13	%Query_ExprBothFld	The criterion's operator is Between and both values on the right-hand side are constants. (Field-Field)
14	%Query_ExprFldExpr	The criterion's operator is Between, the first value on right-hand side is a field and the second value is an expression. (Field-Expr)
15	%Query_ExprExprConst	The criterion's operator is Between, the first value on right-hand side is an expression and the second value is a constant. (Expr-Const)
16	%Query_ExprExprFld	The criterion's operator is Between, the first value on right-hand side is an expression and the second value is a field. (Expr-Field)
17	%Query_ExprBothExpr	The criterion's operator is Between and both values on the right-hand side are expressions. (Expr-Expr)

The following table describes how to access or change Expression 2 depending on the Expression2 Type.

Expression2 Type	Method or Property for Changing the Expression	Method or Property for Accessing the Expression
Constant	Expr2Constant	Expr2Constant
Field	AddExpr2Field()	Expr2Field
Expression	AddExpr2Expression()	Expr2Expression
In List	AddExpr2List	Expr2List
In Tree	AddExpr2Expression	Expr2Expression
Subquery	AddExpr2Subquery()	Expr2Subquery
Const-Const	Expr2Constant, Expr2Constant	Expr2Constant, Expr2Constant

Copyright © 1988-2007, Oracle. All rights reserved.

Expression2 Type	Method or Property for Changing the Expression	Method or Property for Accessing the Expression
Const-Field	Expr2Constant, AddExpr2Field()	Expr2Constant, Expr2Field
Const-Expr	Expr2Constant, AddExpr2Expression()	Expr2Constant, Expr2Expression
Field-Const	AddExpr2Field(), Expr2Constant	Expr2Field, Expr2Constant
Field-Field	AddExpr2Field(), AddExpr2Field()	Expr2Field, Expr2Field
Field-Expr	AddExpr2Field(), AddExpr2Expression()	Expr2Field, Expr2Expression
Expr-Const	AddExpr2Expression(), Expr2Constant	Expr2Expression, Expr2Constant
Expr-Field	AddExpr2Expression(), AddExpr2Field()	Expr2Expression, Expr2Field
Expr-Expr	AddExpr2Expression(), AddExpr2Expression()	Expr2Expression, Expr2Expression

Example

The following is used to test the expression to determine the property to use to retrieve it:

```
&MyExpr2 = &MyQueryColl.Next();

If &MyExpr2.Expr2Type = %Query_ExprConstant Then /* Expression is a constant */

   &OldValue = &MyExpr2.Expr2Constant;
   /* do processing */

End-if;
```

The following is an example showing how to add a field for Expression 1.

```
/* add a new criteria */
&MyCriteria = &MyQuery.AddCriteria();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/* set the type of the first expression to be a field */
&MyCriteria.Expr1Type = %Query_ExprField;

/* add the EMPLID field from the ABSENCE_HIST record whose record alias is A */
&MyField = &MyCriteria.AddExpr1Field("A", "EMPLID" );
```

Logical

Description

This property returns or sets the logical portion of a criteria.

Note. This property is valid only when there are more than one criteria for a query. Also, this property is required when there is more than one criteria for a query.

The values are:

Numeric Value	Constant Value	Description
1	%Query_CombAnd	Logical And for non-having criteria
2	%Query_CombOr	Or for non-having criteria
3	%Query_CombNotUsed	No logical operator. Used for the first non-having criteria.
4	%Query_CombHaveAnd	Logical And for having criteria
5	%Query_CombHaveOr	Logical Or for having Criteria
6	%Query_CombHaveNotUsed	No logical operator. Used for the first having criteria.

This property is read-write.

LParenLvl

Description

This property returns or sets the left parenthesis level used for grouping criteria. This property takes a numeric value.

This property is read-write.

Name

Description

This property returns the name of the Query Criteria, as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Negation

Description

This property returns a Boolean value, indicating whether a criterion is negated: True if the criterion is negated, False if it isn't.

This property is read-only.

Operator

Description

This property returns or sets the operator for the criteria.

The value of this property determines the valid types of the Expression 2, set with the Expr2Type property.

This property is read-write.

See Chapter 30, "Query Classes," Expr2Type, page 1322 and Chapter 30, "Query Classes," Working With Criteria and Expressions, page 1241.

You can use either a constant or numeric value for this property. The values are:

Numeric Value	Constant Value	Description
1	%Query_CondNone	None (used for initializing a new criteria.)
2	%Query_CondEqual	Criteria's left-hand side is equal to right-hand side (= operator)
3	%Query_CondNotEqual	Criteria's left-hand side is not equal to right-hand side (<> operator)
4	%Query_CondGreaterThan	Criteria's left-hand side is greater than right-hand side (> operator)
5	%Query_CondNotGreaterThan	Criteria's left-hand side is not greater than right-hand side (<= operator)
6	%Query_CondLessThan	Criteria's left-hand side is less than right-hand side (< operator)
7	%Query_CondNotLessThan	Criteria's left-hand side is not less than right-hand side (>= operator)
8	%Query_CondInList	Criteria's left-hand side is in the given list (IN operator)
9	%Query_CondNotInList	Criteria's left-hand side is not in the given list (Not IN operator)
10	%Query_CondBetween	Criteria's left-hand side is between the two values of right-hand side (BETWEEN operator)
11	%Query_CondNotBetween	Criteria's left-hand side is not between the two values of right-hand side (BETWEEN operator)
12	%Query_CondExists	Criteria's left-hand side is the output of the subquery of right-hand side (EXISTS operator)

 Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
13	%Query_CondNotExists	Criteria's left-hand side doesn't exist in the output of the subquery of right-hand side (NOT EXISTS operator)
14	%Query_CondLike	Criteria's left-hand side is like (wildcard search) the right-hand side (LIKE operation)
15	%Query_CondNotLike	Criteria's left-hand side is not like (wildcard search) the right-hand side (NOT LIKE operation)
16	%Query_CondNull	Criteria's left-hand side is NULL (NULL operation)
17	%Query_CondNotNull	Criteria's left-hand side is not NULL (IS NOT NULL operation)
18	%Query_CondInTree	Criteria's left-hand side is from a list of nodes in Tree (IN operation)
19	%Query_CondNotInTree	Criteria's left-hand side is not from a list of nodes in Tree (NOT IN operation)
20	%Query_CondEffDtLessEqual	Criteria's left-hand side is an Effective Date and is less than or equal to the date on the right-hand side (<= operation)
21	%Query_CondEffDtGreaterEqual	Criteria's left-hand side is an Effective Date and is greater than or equal to the date on the right-hand side (>= operation)
22	%Query_CondEffDtLess	Criteria's left-hand side is an Effective Date and is less than the date on the right-hand side (< operation)
23	%Query_CondEffDtGreater	Criteria's left-hand side is an Effective Date and is greater than the date on the right-hand side (> operation)
24	%Query_CondFirstEffDt	Criteria's left-hand side is the first effective date (Function MIN())
25	%Query_CondLastEffDt	Criteria's left-hand side is the last effective date (Function MAX())
26	%Query_CondInTreeJoin	Criteria's left-hand side is an In Tree Join.

R1ExprNum

Description

This property returns or sets the expression number for the first expression of Expression 2. This property takes a numeric value.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

R2ExprNum

Description

This property returns or sets the expression number for the second expression of Expression 2. This property takes a numeric value.

This property is read-write.

R1ExprType

Description

This property returns the expression type for the first expression of Expression 2. This property takes a numeric value and is the same range of values as the Expr2Type. It helps distinguish the type of an expression based on the value of Expr2Type. For instance, if the Expr2Type is Field-Expr, R1Expr2Type is of type Field and R2Expr2Type is of type Expression.

This property is read-only.

R2ExprType

Description

This property returns the expression type for the second expression of Expression 2. This property takes a numeric value and is the same range of values as the Expr2Type. It helps distinguish the type of an expression based on the value of Expr2Type. For instance, if the Expr2Type is Field-Expr, R1Expr2Type is of type Field and R2Expr2Type is of type Expression.

This property is read-only.

RParenLvl

Description

This property returns or sets the right parenthesis level used for grouping criteria. This property takes a numeric value.

This property is read-write.

QueryExpression Collection

A QueryExpression Collection is returned from the Expressions Query class property.

See Chapter 30, “Query Classes,” Expressions, page 1290.

 Copyright © 1988-2007, Oracle. All rights reserved.

QueryExpression Collection Methods

In this section, we discuss the QueryExpression collection methods. The methods are described in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryExpression object in the QueryExpression collection.

Parameters

None.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
&MyQueryExpression = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryExpression object that exists at the *number* position in the QueryExpression collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryExpression object that you want returned.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryExpressionColl.Count;
   &MyQueryExpression = &QueryExpressionColl.Item(&I);
   /* do processing */
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

ItemByName

Syntax

ItemByName(*ExpressionName*)

Description

The ItemByName method returns the QueryExpression object in the QueryExpressionCollection with the given expression name.

Parameters

Parameter	Description
ExpressionName	The name of the required expression.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
&QryExpr = &QryExprCol.Item("Exp-6");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryExpression object in the QueryExpression collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryExpression object if successful, NULL otherwise.

Example

```
&MyQueryExpression = &MyCollection.Next();
```

QueryExpression Collection Property

In this section, we discuss the Count property.

 Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of QueryExpression objects in the QueryExpression Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryExpression Class

A QueryExpression object is return by the following:

- The AddExpression Query class method.
- The First, Item and Next methods of the QueryExpression Collection.
- All of the QueryCriteria Class Methods.
- The Expr1Expression and Expr2Expression1 QueryCriteria properties.

See Chapter 30, “Query Classes,” AddExpression, page 1285; Chapter 30, “Query Classes,” QueryExpression Collection, page 1328; Chapter 30, “Query Classes,” QueryCriteria Class Methods, page 1315; Chapter 30, “Query Classes,” Expr1Expression, page 1319 and Chapter 30, “Query Classes,” Expr2Expression1, page 1320.

See Chapter 30, “Query Classes,” Working With Criteria and Expressions, page 1241.

QueryExpression Class Properties

In this section, we discuss the QueryExpression class properties. The properties are discussed in alphabetical order.

Aggregate

Description

This property specifies whether the expression is an aggregate function.

This property takes a Boolean value: True if the expression is an aggregate function, False, otherwise.

This property is read-write.

BindFlag

Description

This property specifies whether the expression contains a bind value, as Boolean value.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is mainly used while reading a query, to determine whether a Query Expression contains a bind value. It isn't necessary to set this value while saving a query to the database.

Values are:

Value	Description
False	Expression doesn't have a bind value
True	Expression has a bind value

This property is read-write.

Decimal

Description

This property returns or sets the decimal value of the expression.

This property is only valid with numeric fields.

This property is read-write.

ExpNum

Description

This property returns or sets the unique expression number, as a numeric value.

This property is read-write.

IsXlatExpression

Description

This property indicates whether the expression is for a translate field. This property takes a Boolean value: True, the expression is based on a translated field.

This property is read-only.

Length

Description

This property returns or sets the length of the expression, as a number.

This property is read-write.

Name

Description

This property returns the name of the expression, as a string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

OutputField

Description

This property returns the instance of the displayed expression field, as a QueryField. This property returns a NULL when the expression isn't used in a displayed (output) field.

This property is read-write.

RightExprFlag

Description

This property specifies whether an expression is used in the right-hand side of a criteria (that is, is it an Expr2 expression), as a Boolean value.

This property is typically used while reading a query to determine whether a Query Expression is used in the right-hand side of criteria. It's not necessary to set this value while saving a query to the database.

Values are:

Value	Description
False	Expression is not used in the right-hand side of the criteria
True	Expression is used in the right-hand side of the criteria

This property is read-write.

SelectedField

Description

This property returns the instance of the expression field, as a QueryField. This property returns a NULL when the expression is used only in the right-hand side of a criteria.

This property is read-write.

Text

Description

This property returns or sets the text of the expression, as a character string.

This property is read-write.

Using ORACLE Hints in Expressions

Oracle hints can be included in expressions using the following considerations:

- Expression containing a hint must begin with /*+.
- Expression can only contain one hint. For example, only one set of /* */ is allowed in each expression.
- Each /* must precede an */.

Copyright © 1988-2007, Oracle. All rights reserved.

- Each expression must contain a complete hint. For example, an expression can't have only /* or */. Both must be in same expression.

Type

Description

This property returns or sets the field type of the expression.

You can specify either a constant or a numeric value. The values are:

Numeric Value	Constant Value	Description
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	Datetime
7	%FieldType_Image	Image

This property is read-write.

QueryList Class

A QueryList Class is returned from the Expr2List property and AddExpr2List method of the QueryCriteria Class.

See Chapter 30, "Query Classes," Expr2List, page 1322 and Chapter 30, "Query Classes," AddExpr2List, page 1318.

QueryList Class Methods

In this section, we discuss the QueryList class methods. The methods are discussed in alphabetical order.

AddListValue

Syntax

AddListValue(*Value, IsPrompt*)

Description

The AddListValue method adds a new List Value into the QueryList instance.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Value	Specify the string value to be added to the list
IsPrompt	Specify whether the string specified by *Value* is a bind variable. This parameter takes a Boolean value: True, *Value* is a bind variable.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
&MyListValue = &MyList.AddListValue("1", False);
```

First

Syntax

First()

Description

The First method returns the first QueryListValue object in the QueryList instance.

Parameters

None.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
&MyListValue = &MyList.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryListValue object that exists at the *number* position in the QueryList instance.

Parameters

Parameter	Description
Number	Specify the position number in the QueryList object that you want returned.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryList.Count;
   &QryListVal = & QueryList.Item(&I);
   /* do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next QueryListValue object in the QueryList instance. This method should be called after calling First, else it returns NULL.

Parameters

None.

Returns

A reference to a QueryListValue object if successful, NULL otherwise.

Example

```
&MyNextListValue = &MyList.Next();
```

QueryList Class Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryListValue objects in the QueryList Collection, as a number.

This property is read-only.

Example

```
&ListValueCount = &MYList.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

QueryListValue Class

A QueryListValue instance is returned from the AddListValue, First, Item, or Next QueryList Class methods.

See Chapter 30, "Query Classes," QueryList Class Methods, page 1334.

QueryListValue Class Properties

In this section, we discuss the QueryListValue class properties. These properties are discussed in alphabetical order.

IsPrompt

Description

This property indicates whether the value is a bind variable (such as :1). This property takes a Boolean value: True, this property is a bind variable.

This property is read-only.

Value

Description

This property returns a string for the value stored in the list.

This property is read-only.

QueryRecordHierarchy Collection

A QueryRecordHierarchy collection is returned from the RecordHierarchy QueryDBRecord property.

The order of each QueryRecordHierarchy object in the collection maps to the order of each tree node as it appears in the tree hierarchy from top to bottom.

See Chapter 30, "Query Classes," RecordHierachy, page 1354.

QueryRecordHierarchy Collection Methods

In this section, we discuss the QueryRecordHierarchy collection methods. The methods are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

First

Syntax

First()

Description

The First method returns the first QueryRecordHierarchy object in the QueryRecordHierarchy collection.

Parameters

None.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
&MyQueryRecordHierarchy = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryRecordHierarchy object that exists at the *number* position in the QueryRecordHierarchy collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryRecordHierarchy object that you want returned.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryRecordHierarchyColl.Count;
   &MyQueryRecordHierarchy = &QueryRecordHierarchyColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*Name*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The ItemByName method returns the QueryRecordHierarchy object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing QueryRecordHierarchy within the QueryRecordHierarchy collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
&MyQueryRecordHierarchy = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryRecordHierarchy object in the QueryRecordHierarchy collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryRecordHierarchy object if successful, NULL otherwise.

Example

```
&MyQueryRecordHierarchy = &MyCollection.Next();
```

QueryRecordHierarchy Collection Property

In this section, we discuss the QueryRecordHierarchy properties. These properties are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of QueryRecordHierarchy objects in the QueryRecordHierarchy Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryRecordHierarchy Class

A reference to a QueryRecordHierarchy object is returned by the First, Item, ItemByName, and Next QueryRecordHierarchy Collection methods.

Every QueryRecordHierarchy object returned from a collection represents a record node in the Record Hierarchy tree in Query tab of PeopleSoft Query.

The QueryRecordHierarchy object returned from the QueryField represents the prompt table for the record field.

See Chapter 30, "Query Classes," QueryRecordHierarchy Collection, page 1337.

QueryRecordHierarchy Class Properties

In this section, we discuss the QueryRecordHierarchy class properties. The properties are discussed in alphabetical order.

Description

Description

This property returns a description of the record node as a string.

This property is read-only.

Level

Description

This property returns the level of the record node in the record hierarchy. 1 is the root node, 2 is a node beneath the root node, 3 is a child of that, and so on.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Name

Description

This property returns the name of the record node as a string.

This property is read-only.

ParentFlag

Description

This property returns the parent flag. The values are:

Value	Description
0	Record node contains no children nodes.
1	Record node contains children nodes

This property is read-only.

Query Metadata Collection

A Query Metadata collection is returned by the Metadata Query class property.

See Chapter 30, "Query Classes," Metadata, page 1279.

See Chapter 30, "Query Classes," Using Query Metadata, page 1243.

Query Metadata Collection Methods

In this section, we discuss the Query Metadata collection methods. These methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first Query Metadata object in the Query Metadata collection.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
&MyMetadata = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the Query Metadata object that exists at the *number* position in the Query Metadata collection.

Parameters

Parameter	Description
number	Specify the position number in the collection of the Query Metadata object that you want returned.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
For &I = 1 to &MetadataColl.Count;
   &MyMetadata = &MetadataColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the Query Metadata object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing Query Metadata within the Query Metadata collection. If you specify an invalid name, the object is NULL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
&MyMetadata = &MyQuery.Metadata.ItemByName("Descr");
```

Next

Syntax

Next()

Description

The Next method returns the next Query Metadata object in the Query Metadata collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a Query Metadata object if successful, NULL otherwise.

Example

```
&MyMetadata = &MyCollection.Next();
```

Query Metadata Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of Query Metadata objects in the Query Metadata Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Query Metadata Class

A Query Metadata object is returned from the Query Metadata Collection methods First, Item, ItemByName, or Next.

See Chapter 30, "Query Classes," Query Metadata Collection, page 1341.

See Chapter 30, "Query Classes," Using Query Metadata, page 1243.

Query Metadata Class Properties

In this section, we discuss the Query Metadata class properties. The properties are discussed in alphabetical order.

Name

Description

This property returns the name of each Query Metadata property as a string.

Values for this property are:

Value	Description
Descr	Description
LongDescr	Long description
Public/Private	Specifies if the query is public or private. If the query is private, the name of the owner is listed in Value.
LastUpdDttm	Last updated date and time
LastUpdOprId	The UserId of the user who updated the value last
Record	Record name. May be more than one.
Input Param	Input parameter. May be more than one.
Expression	Expression. May be more than one.
Field	Record field name for the output column. May be more than one.
Heading	Heading name for the output column. May be more than one.

This property is read-only.

Value

Description

This property returns the value for the Query Metadata property as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

QueryStatistics Class

The QueryStatistics class is used to view statistical information about a query's execution. It can be useful for query administration. You can view query statistics for a query before you save it to the database.

A QueryStatistics Class is returned from the QueryStatistics Query property.

See Chapter 30, "Query Classes," QueryStatistics, page 1282.

QueryStatistics Class Properties

In this section, we discuss the QueryStatistics properties. The properties are discussed in alphabetical order.

AvgExecTime

Description

This property returns the average execution time for the query as a string.

This property is read-only.

AvgFetchTime

Description

This property returns the average fetch time for the query as a string.

This property is read-only.

AvgNumRows

Description

This property returns the average number of rows fetched for the query.

This property is read-only.

ExecCount

Description

This property returns the total number of times the query has been executed.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

LastExecDtTm

Description

This property returns the last date and time the query was executed, as a string.

This property is read-only.

QuerySecurityProfile Class

The QuerySecurityProfile class is used to view the current user's security profile for PeopleSoft Query. This class doesn't contain any methods, and all the properties are read-only. An instance of this class is returned by the GetQuerySecurityProfile Session method.

See Chapter 30, "Query Classes," GetQuerySecurityProfile, page 1256.

QuerySecurityProfile Class Properties

In this section, we discuss the QuerySecurityProfile properties. The properties are discussed in alphabetical order.

AllowAnyJoin

Description

This property indicates whether the user is allowed to define queries with any join. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowDistinct

Description

This property indicates whether the user can define queries with a Distinct clause in a SELECT statement. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowExpressions

Description

This property indicates whether the user is allowed to define expressions in queries. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

AllowSubqueries

Description

This property indicates whether the user is allowed to define criteria containing subqueries. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

AllowUnions

Description

This property indicates whether the user is allowed to define queries containing unions. This property takes a Boolean value: True, the user can define such queries.

This property is read-only.

ApprovePrivateQuery

Description

This property indicates whether the user is allowed to approve private queries. This property takes a Boolean value: True, the user can approve such queries. This property is meant for query administration.

This property is read-only.

ApprovePublicQuery

Description

This property indicates whether the user is allowed to approve public queries. This property takes a Boolean value: True, the user can approve such queries. This property is meant for query administration.

This property is read-only.

CanCreatePublic

Description

This property indicates whether the user can create public queries. This property takes a Boolean value: True, the user can create such queries.

This property is read-only.

CanCreateWorkFlow

Description

This property indicates whether the user can create or run any workflow queries. Workflow queries are of the following types:

- Archive
- Process

Copyright © 1988-2007, Oracle. All rights reserved.

- Role

This property takes a Boolean value: True, the user can create such queries.

This property is read-only.

CanModifyQuery

Description

This property indicates whether the user can modify queries. This property takes a Boolean value: True, the user can modify queries.

This property is read-only.

CanRunQuery

Description

This property indicates whether the user can run queries. This property takes a Boolean value: True, the user can run queries.

This property is read-only.

CanRunToCrystal

Description

This property indicates whether the user can run queries to a Crystal report. This property takes a Boolean value: True, the user can run queries.

This property is read-only.

CanRunToExcel

Description

This property indicates whether the user can run queries to an Excel spreadsheet. This property takes a Boolean value: True, the user can run queries.

This property is read-only.

LimitUnapproved

Description

This property indicates whether there is a limit on the number of rows returned for unapproved queries. This property is meant for query administration. This property takes a Boolean value: True, limit the number of rows. The MaxUnapprovedRows property is active only if this property is specified as True.

This property is read-only.

See Also

Chapter 30, “Query Classes,” MaxUnapprovedRows, page 1349

 Copyright © 1988-2007, Oracle. All rights reserved.

MaxInTreeCriteria

Description

This property indicates the maximum number of In Tree Criteria that can be used in the queries defined by the current user. This property takes a numeric value.

This property is read-only.

MaxJoins

Description

This property indicates the maximum number of joins allowed in the queries defined by the current user. This property takes a numeric value.

This property is read-only.

MaxRowsToFetch

Description

This property indicates the maximum number of rows to fetch for the current user when a query is executed. This property takes a numeric value.

This property is read-only.

MaxUnapprovedRows

Description

This property indicates the maximum number of rows that can be returned for unapproved queries. This property takes a numeric value. This property is meant for query administration. This property is active only if the property LimitUnapproved is True.

This property is read-only.

See Also

Chapter 30, “Query Classes,” LimitUnapproved, page 1348

QueryDBRecord Collection

A QueryDBRecord collection is returned from the FindQueryDBRecords session method.

See Chapter 30, “Query Classes,” FindQueryDBRecords, page 1253.

Copyright © 1988-2007, Oracle. All rights reserved.

QueryDBRecord Collection Methods

In this section, we discuss the QueryDBRecord collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryDBRecord object in the QueryDBRecord collection.

Parameters

None.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
&MyQueryDBRecord = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryDBRecord object that exists at the *number* position in the QueryDBRecord collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryDBRecord object that you want returned.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryDBRecordColl.Count;
   &MyQueryDBRecord = &QueryDBRecordColl.Item(&I);
   /* do processing */
End-For;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the QueryDBRecord object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing QueryDBRecord within the QueryDBRecord collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
&MyQueryDBRecord = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryDBRecord object in the QueryDBRecord collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryDBRecord object if successful, NULL otherwise.

Example

```
&MyQueryDBRecord = &MyCollection.Next();
```

QueryDBRecord Collection Property

In this section, we discuss the Count property.

Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of QueryDBRecord objects in the QueryDBRecord Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryDBRecord Class

A QueryDBRecord object is returned from the QueryDBRecord Collection methods First, Item, ItemByName, or Next.

See Chapter 30, "Query Classes," QueryDBRecord Collection, page 1349.

QueryDBRecord Class Methods

In this section, we discuss the QueryDBRecord class methods. The methods are discussed in alphabetical order.

QueryDBRecordFieldByIndex

Syntax

QueryDBRecordFieldByIndex(*Index*)

Description

The QueryDBRecordFieldByIndex method returns the QueryDBRecordField object that exists at the *index* position in the QueryDBRecordField collection.

Parameters

Parameter	Description
Index	Specify the position number in the collection of the QueryDBRecordField object that you want returned.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

See Also

Chapter 30, "Query Classes," QueryDBRecordFieldByName, page 1353 and Chapter 30, "Query Classes," QueryDBRecordField Class, page 1357.

 Copyright © 1988-2007, Oracle. All rights reserved.

QueryDBRecordFieldByName

Syntax

QueryDBRecordFieldByName(*Name*)

Description

The QueryDBRecordFieldByName method returns the QueryDBRecordField object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of the QueryDBRecordField object that you want returned.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

See Also

Chapter 30, "Query Classes," QueryDBRecordFieldByIndex, page 1352 and Chapter 30, "Query Classes," QueryDBRecordField Class, page 1357.

QueryDBRecord Class Properties

In this section, we discuss the QueryDBRecord class properties. The properties are discussed in alphabetical order.

Description

Description

This property returns the description of the QueryDBRecord as a string.

This property is read-only.

Name

Description

This property returns the name of the QueryDBRecord as a string.

This property is read-only.

QueryDBRecordFields

Description

This property returns a reference to a QueryDBRecordField Collection.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 30, "Query Classes," QueryDBRecordField Collection, page 1354

RecordHierachy

Description

This property returns a reference to a QueryRecordHierarchy Collection.

The record hierarchy is not related to the query tree hierarchy shown when viewing access groups. Instead, it reflects an actual relationship between the record components, as defined in PeopleSoft Application Designer using the Parent Record Name feature.

This property is read-only.

See Also
Chapter 30, "Query Classes," QueryRecordHierarchy Collection, page 1337

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query, "PeopleSoft Query Security"

QueryDBRecordField Collection

A QueryDBRecordField collection is returned from the QueryDBRecordFields QueryDBRecord property.

See Chapter 30, "Query Classes," QueryDBRecordFields, page 1353.

QueryDBRecordField Collection Methods

In this section, we discuss the QueryDBRecordField collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Description

The First method returns the first QueryDBRecordField object in the QueryDBRecordField collection.

Parameters

None.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyQueryDBRecordField = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the QueryDBRecordField object that exists at the *number* position in the QueryDBRecordField collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryDBRecordField object that you want returned.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Example

```
For &I = 1 to &Coll.Count;
   &MyQueryDBRecordField = &Coll.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the QueryDBRecordField object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing QueryDBRecordField within the QueryDBRecordField collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyQueryDBRecordField = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

Description

The Next method returns the next QueryDBRecordField object in the QueryDBRecordField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryDBRecordField object if successful, NULL otherwise.

Example

```
&MyQueryDBRecordField = &MyCollection.Next();
```

Sort

Syntax

Sort(*SortCriteria*)

Description

Use the Sort method to sort the fields within the QueryDBRecordField collection, based on the sort criteria specified with the method.

Parameters

Parameter	Description
SortCriteria	Specify the sort order for the list. This parameter can take either a constant or numeric value. The values are:

Numeric Value	Constant Value	Description
1	%Query_SortNameAsc	Sort database fields in ascending order based on field name.
2	%Query_SortNameDesc	Sort database fields in descending order based on field name.
3	%Query_SoftFldNumAsc	Sort database fields in ascending order based on field number.
4	%Query_SortFldNumDesc	Sort database fields in descending order based on field number.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

QueryDBRecordField Collection Property

In this section, we discuss the Count property.

Count

Description

This property returns the number of QueryDBRecordField objects in the QueryDBRecordField Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryDBRecordField Class

A QueryDBRecordField object is from the QueryDBRecord Collection methods First, Item, ItemByName or Next.

See Chapter 30, “Query Classes,” QueryDBRecord Collection, page 1349.

QueryDBRecordField Class Properties

In this section, we discuss the QueryDBRecordField class properties. The properties are discussed in alphabetical order.

Decimal

Description

This property returns the decimal positions for a field. This indicates how many numbers are allowed on the right side of the decimal.

This property is read-only.

Example

```
&FldDecs = &QryDBRcdFlds.Decimal;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Description

This property returns the long description of the field as a string.

This is property is read-only.

Flag

Description

This property returns the Use and Edit Flags of the field, which are set when the Field is defined in PeopleSoft Application Designer, as a numeric value.

Values for Use Flags are:

Value	Description
1	Key
2	Duplicate Key
4	System
8	Audit Field Add
16	Alternate Key
32	List Item
64	Descending Key
128	Audit Field Change
1024	Audit Field Delete
2048	Search Item
32768	Auto Update

Values for Edit Flags are:

Value	Description
256	Required
512	Edit Translate
4096	Date Range Edit
8192	Yes/No Table Edit
16384	Edit Table
262144	From Search Field
524288	Through Search Field

 Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
8388608	Use Default Label Flag
16777216	Default Search Field

This is a read-only property.

Format

Description

This property returns the field format for a field. Values are:

Value	Description
1	No format
2	Name
3	Phone Number North America
4	Zip/Postal Code North America
5	Social Security Number (SSN)
6	Uppercase
7	Mixed case
8	Raw binary
9	Numbers only
10	Canadian Social Insurance Number (SIN)
11	Phone Number International
12	Zip/Postal Code International
13	Seconds
14	Microseconds
15	Custom

This property is read-only.

Length

Description

This property returns the length of the field as a number.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

LongName

Description

This property returns the long name of the field as a string.

This property is read-only.

LookupTableName

Description

If the field has a lookup table associated with it, this property returns the name of that Lookup Table, else it returns an empty string.

This is a read-only property.

LookupTableRecord

Description

If the field has a lookup table associated with it, this property returns the instance of the QueryDBRecord for that Lookup Table, else it returns NULL.

This is a read-only property.

Name

Description

This property returns the name of the field as a string.

This property is read-only.

Shortname

Description

This property returns the short name of the field as a string.

This property is read-only.

Type

Description

This property returns the type of the field. You can specify either a constant or a numeric value for this property. The values are:

Numeric Value	Constant Value	Description
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Numeric Value	Constant Value	Description
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	Datetime
7	%FieldType_Image	Image

This property is read-only.

QueryPrompt Collection

A QueryPrompt collection is returned from the Prompts and RuntimePrompts Query class properties.

See Chapter 30, "Query Classes," Prompts, page 1280 and Chapter 30, "Query Classes," RunTimePrompts, page 1282.

QueryPrompt Collection Methods

In this section, we discuss the QueryPrompt collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first QueryPrompt object in the QueryPrompt collection.

Parameters

None.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
&MyQueryPrompt = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Item method returns the QueryPrompt object that exists at the *number* position in the QueryPrompt collection.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the QueryPrompt object that you want returned.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
For &I = 1 to &QueryPromptColl.Count;
   &MyQueryPrompt = &QueryPromptColl.Item(&I);
   /* do processing */
End-For;
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the QueryPrompt object with the name *Name*.

Parameters

Parameter	Description
Name	Specify the name of an existing QueryPrompt within the QueryPrompt collection. If you specify an invalid name, the object is NULL.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
&MyQueryPrompt = &MyCollection.ItemByName("PHONELIST");
```

Next

Syntax

Next()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Next method returns the next QueryPrompt object in the QueryPrompt collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Parameters

None.

Returns

A reference to a QueryPrompt object if successful, NULL otherwise.

Example

```
&MyQueryPrompt = &MyCollection.Next();
```

QueryPrompt Collection Property

In this section, we discuss the QueryPrompt collection properties. The properties are discussed in alphabetical order.

Count

Description

This property returns the number of QueryPrompt objects in the QueryPrompt Collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

QueryPrompt Class

A reference to a QueryPrompt is returned by the following:

- The First, Item, ItemByName, and Next methods of a QueryPrompt Collection.
- The AddPrompt Query class method.

See Chapter 30, "Query Classes," QueryPrompt Collection, page 1361 and Chapter 30, "Query Classes," AddPrompt, page 1264.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Query*, "Defining Selection Criteria," Defining Prompts.

Copyright © 1988-2007, Oracle. All rights reserved.

QueryPrompt Properties

In this section, we discuss the QueryPrompt properties. The properties are discussed in alphabetical order.

EditType

Description

This property returns or sets the edit type for the field. This property takes a number value. The values are:

Value	Description
0	No table edit
16384	Prompt table
512	Translate table
8192	Yes/No

This property is read-write.

FieldDecimal

Description

This property returns or sets the decimal value for the field.

This property is only valid with number fields.

This property is read-write.

FieldFormat

Description

This property returns the field format for a field. This property takes a number value. Values are:

Value	Description
1	No format
2	Name
3	Phone Number North America
4	Zip/Postal Code North America
5	Social Security Number (SSN)
6	UpperCase
7	Mixed case
8	Raw binary

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
9	Numbers only
10	Canadian Social Insurance Number (SIN)
11	Phone Number International
12	Zip/Postal Code International
13	Seconds
14	Microseconds
15	Custom

This property is read-only.

FieldLength

Description

This property returns or sets the field length.

This property is read-write.

FieldName

Description

This property returns or sets the field name used with the prompt.

This property is read-write.

FieldType

Description

This property returns or sets the field type of the field used with the prompt.

This property returns the type of the field. You can specify either a constant or a numeric value. The values are:

Numeric Value	Constant Value	Description
0	%FieldType_Char	Character
1	%FieldType_LongChar	Long Character
2	%FieldType_Number	Number
3	%FieldType_SignedNumber	Signed number
4	%FieldType_Date	Date
5	%FieldType_Time	Time
6	%FieldType_DateTime	Datetime
7	%FieldType_Image	Image

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

HeadingText

Description

This property returns or sets the heading text for the prompt field.

This property is read-write.

HeadingType

Description

This property returns or sets the heading type for the query field. This property takes either a constant or numeric value. The values are:

Numeric Value	Constant Value	Description
1	%Query_HdgNone	Query field has no heading.
2	%Query_HdgText	Query field has a text heading.
3	%Query_HdgRftShort	Query field uses the short RFT heading.
4	%Query_HdgRftLong	Query fields uses the long RFT heading.

This property is read-write.

LangCount

Description

This property returns the total count of the language records for the current prompt.

This property is read-only.

Name

Description

This property returns a string containing the Prompt name. When an existing query is read, this name is the same as the Field Name. When a new prompt is added using AddPrompt, this is the *Name* parameter used with that method.

This property is read-only.

PromptRecordFieldName

Description

When you have more than one field with the same name in the prompt collection, the system generates a unique prompt field name for each repeated field. The generated names are of the form Bind1, Bind2, and so on. This property returns the unique name for the prompt record field.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

PromptTable

Description

This property returns or sets the prompt table name for the prompt field.

This property is read-write.

UniquePromptName

Description

This property returns or sets the unique prompt name for the prompt field.

This property is read-write.

UseCount

Description

This property returns the total count of the usage of the current prompt.

This property is read-only.

The following are examples of the usual actions that you perform using the Query classes.

Query Classes Examples

The following are examples of the usual actions that you perform using the Query classes.

Creating a New Query

In this example, you are creating a new query, adding a record and two fields. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &aQuery, &aQrySelCol;
Local ApiObject &COLL, &ERROR;
Local String &TEXT;
Local Session &MySession;
Local Record &aQryRcd;
Local Field &aQryFld;

&MySession = %Session;

If &MySession <> Null Then

   &aQuery = &MySession.GetQuery();

   &aQuery.Create("TEST1", False, %Query_Query, "PIA Test 1", "Creating Test Query⇒
 1 from PIA Page");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   &aQrySel = &aQuery.AddQuerySelect();

   &aQryRcd = &aQrySel.AddQueryRecord("ABSENCE_HIST");
   &aQryFld = &aQrySel.AddQuerySelectedField("ABSENCE_HIST", "A", "EMPLID", "ID");

   If &aQryFld <> Null Then
         &aQryFld.ColumnNumber = 1;
         &aQryFld.HeadingType = %Query_HdgRftShort;
      End-If;

   &Rslt = &aQuery.Save();

   If &Rslt <> 0 Then
      /* save didn&rsquo;t complete */
      &COLL = &MySession.PSMessages;

      For &I = 1 to &COLL.Count
         &ERROR = &COLL.Item(&I);
         &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;

      &COLL.DeleteAll();
   End-if;
    /* error processing for not getting a session */
End-if;
```

To create a new query:

1. Get a session object.

   Before you can create a query, you have to get a session object. The session controls access to the query, provides error tracing, enables you to set the runtime environment, and so on. Then this program checks to verify that the session object is valid.

   ```
   &MySession = %Session;

   &aQuery = &MySession.GetQuery();

   If &MySession <> Null Then
   ```

2. Create the query.

   Use the Create method to create the query. This query is a private query, of type query.

   ```
   &aQuery.Create("TEST1", False, %Query_Query, "Test 1", "Creating Test Query");
   ```

3. Add a QuerySelect.

 Copyright © 1988-2007, Oracle. All rights reserved.

The QuerySelect contains the main query statement for the query. There can be multiple QuerySelect objects for Queries that involve unions or subqueries. Each select (or union or subquery) consists of QueryRecords, QueryOutputFields, QuerySelectedFields, and QueryCriteria and is treated as a child of the MAIN select statement.

```
&aQrySel = &aQuery.AddQuerySelect();
```

4. Add a record and a field.

   The AddQueryRecord method adds a query record to the query. The AddQuerySelectedField adds a field, using the record alias "A". The ID is what gets displayed in the heading for the query.

```
&aQryRcd = &aQrySel.AddQueryRecord("ABSENCE_HIST");
&aQryFld = &aQrySel.AddQuerySelectedField("ABSENCE_HIST", "A", "EMPLID", "ID");
```

5. Make the field an output field.

   The field was added as a selected field. By setting the ColumnNumber to a number greater than one, the field is now an output field. The text that's displayed in the heading comes from the RFT short description of the field.

```
If &aQryFld <> Null Then
    &aQryFld.ColumnNumber = 1;
    &aQryFld.HeadingType = %Query_HdgRftShort;
 End-If;
```

6. Save the data.

   When you execute the Save method, the new query is saved to the database.

```
&Rslt = &aQuery.Save();
If &Rslt <> 0 Then
```

   The Save method returns a numeric value: 0 if successful. You can use this value to do error checking.

7. Check Errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
   /* save didn&rsquo;t complete */
   &COLL = &MySession.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, you may want to delete it from the PSMessages collection.

Adding Criteria

In this example, you are accessing an existing query, then adding criteria both as part of the query as well as part of a subquery. The SQL statement created by this subquery is as follows:

Copyright © 1988-2007, Oracle. All rights reserved.

```
SELECT RECNAME, RECDESCR, RELLANGRECNAME, PARENTRECNAME, DESCRLONG from PSRECDEFN ⇒
where RECNAME IN (select OBJECTVALUE1 from PSPROJECTITEM where PROJECTNAME =⇒
 'PPLTOOLS') AND RECTYPE = 0 order by RECNAME
```

The following is the complete code sample: the steps explain each line.

```
Local ApiObject &MyQuery, &MainQrySel, &Criteria1, &MyCrit2Expr2, &MyCriteria2;
Local ApiObject &SubQrySel, &SubQryCrit1, &SubQryExpr1, &MyCrit2Expr2;
Local Record &SubQryRec;
Local Session = &MySession;
Local ApiObject &COLL, &ERROR;
Local String &TEXT;

&MySession = %Session;

If &MySession <> Null Then

&MyQuery = &MySession.GetQuery();

&MyQuery.Open("Table", False, True);

&MainQrySel = &MyQuery.QuerySelect;

/* Adding query recory, adding fields, then making selected fields output fields *⇒
/

&MainRec = &MainQrySel.AddQueryRecord("PSRECDEFN");

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RECNAME", "Record Name");
&QryFld.ColumnNumber = 1;
&QryFld.OrderByNumber = 1;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RECDESCR", "Record Descr");
&QryFld.ColumnNumber = 2;
&QryFld.OrderByNumber = 2;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RELLANGRECNAME", "Record Lang Rec");
&QryFld.ColumnNumber = 3;
&QryFld.OrderByNumber = 3;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "PARENTRECNAME", "Parent Record Name");
&QryFld.ColumnNumber = 4;

&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "DESCRLONG", "Long Descr");
&QryFld.ColumnNumber = 5;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "EMPLID", "ID");
&QryFld.ColumnNumber = 6;

/* adding first criteria */
&Criteria1 = &MainQrySel.AddCriteria("FirstCriteria");

/* First criteria will not have any logical AND/OR */
&Criteria1.Logical = %Query_CombNotUsed;

&Criteria1.Expr1Type = %Query_ExprField;
&Criteria1.AddExpr1Field(&MainRec.RecordAlias, "RECNAME");

/* So that the criteria is constructed as - RECNAME IN (...)*/
&Criteria1.Operator = %Query_CondInList;
&Criteria1.Expr2Type = %Query_ExprSubQuery;

&SubQrySel = &Criteria1.AddExpr2SubQuery();
&SubQryRec = AddQueryRecord("PSPROJECTITEM");
&SubQryFld1 = &SubQrySel.AddQuerySelectedField(&SubQryRec.Name, &SubQryRec.Record⇒
Alias, "OBJECTVALUE1", "Join Object")
&SubQryFld1.ColumnNumber = 1;

/* Need criteria - PROJECTNAME = 'PPLTOOLS'  - in the subquery */
&SubQryCrit1 = &SubQrySel.AddCriteria("FirstSubCrit");

/* First criteria will not have any logical AND/OR */
&SubQryCrit1.Logical = %Query_CombNotUsed;

&SubQryCrit1.Expr1Type = %Query_ExprField;
&SubQryCrit1.AddExpr1Field(&SubQryRec.RecordAlias, "PROJECTNAME");
&SubQryCrit1.Operator = %Query_CondEqual;

/* So that the criteria is constructed as - PROJECTNAME = 'PPLTOOLS'*/
&SubQryCrit1.Expr2Type = %Query_ExprConstant;
&SubQryExpr1 = &SubQryCrit1.AddExpr2Expression();
&SubQryExpr1.Text = "PPLTOOLS";
&SubQryCrit1.Expr2Expression1 = &SubQryExpr1;

/* Second Criteria, which is for RECTYPE = 0 */
&MyCriteria2 = &MainQrySel.AddCriteria("SecondCriteria");

&MyCriteria2.Expr1Type = %Query_ExprField;
&MyCriteria2.AddExpr1Field(&MainRec.RecordAlias, "RECTYPE");

/* Since this is second criteria, we need a logical AND to state that - AND⇒
 RECTYPE = 0 */

&MyCriteria2.Logical = %Query_CombAnd;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&MyCriteria2.Operator = %Query_CondEqual;
&MyCriteria2.Expr2Type = %Query_ExprConstant;
&MyCrit2Expr2 = &MyCriteria2.AddExpr2Expression();
&MyCriteria2.Expr2Expression1 = &MyCrit2Expr2;
&MyCrit2Expr2.Text = "0";

&Rslt = &MyQuery.Save();

If &Rslt <> 0 Then
   /* save didn&rsquo;t complete */
   &COLL = &MySession.PSMessages;

      For &I = 1 to &COLL.Count
         &ERROR = &COLL.Item(&I);
         &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;

      &COLL.DeleteAll();
   End-if;

Else

/* do error processing for not getting session */

End-if;
```

To add criteria to a query:

1. Get a session object.

   Before you can create a query, you have to get a session object. The session controls access to the query, provides error tracing, enables you to set the runtime environment, and so on. Then this program checks to verify that the session object is valid.

   ```
   &MySession = %Session;

   &MyQuery = &MySession.GetQuery();

   If &MySession <> Null Then
   ```

2. Access an existing query and get the main query select statement.

   Use the Open method to get the existing query. Then access the main select statement with the QuerySelect property.

   ```
   &MyQuery.Open("Table", False, True);

   &MainQrySel = &MyQuery.QuerySelect;
   ```

3. Add Query Record in the Main Select.

   Add the query record that you want to use.

   ```
   &MainRec = &MainQrySel.AddQueryRecord("PSRECDEFN");
   ```

 Copyright © 1988-2007, Oracle. All rights reserved.

4. Add the displayed fields.

   You want to add the selected fields. Note instead of hardcoding the name of the record, this code example uses the Name property. Also, the code uses the RecordAlias property instead of hardcoding the alias. This makes the code easier to read, as well as easier to maintain. Specifying a column number also makes this an output field.

```
&QryFld = &MainQrySel.AddQuerySelectedField(&MainRec.Name, &MainRec.RecordAlias,⇒
 "RECNAME", "Record Name");
&QryFld.ColumnNumber = 1;
```

5. Specify the OrderBy value.

   Because we need to order by this field, the OrderByNumber of that field must be set also.

```
&QryFld.OrderByNumber = 1;
```

6. Add the first criteria.

   Add the first criteria. You don't want it added with any kind of operator, like an AND or an OR, so the Logical property of the first criteria is set with %Query_CombNotUsed. This is also used because it's the first non-having criteria of a query.

```
/* adding first criteria */
&Criteria1 = &MainQrySel.AddCriteria("FirstCriteria");

/* First criteria will not have any logical AND/OR */
   &Criteria1.Logical = %Query_CombNotUsed;
```

7. Add the first criteria field.

   The first field for the criteria is a QueryExpression type field. The type of the field *must* be set before the field is added.

```
   &Criteria1.Expr1Type = %Query_ExprField;
   &Criteria1.AddExpr1Field(&MainRec.RecordAlias, "RECNAME");
```

8. Add the condition for the first criteria and the subquery.

   The first criteria is the WHERE RECNAME IN portion of the SQL statement. The condition is considered 'in list', where the list is the result of the subquery. The expression is a subquery. Again, you have to set the type again before adding the subquery.

```
&Criteria1.Operator = %Query_CondInList;
&Criteria1.Expr2Type = %Query_ExprSubQuery;
&SubQrySel = &Criteria1.AddExpr2SubQuery();
```

9. Add the records for the subquery.

   Add the query record and the field from the query field, and make it an output field.

```
&SubQryRec = AddQueryRecord("PSPROJECTITEM");
&SubQryFld1 = &SubQrySel.AddQuerySelectedField(&SubQryRecName.Name, &SubQry⇒
Rec.RecordAlias, "OBJECTVALUE1", "Join Object");
&SubQryFld1.ColumnNumber = 1;
```

10. Add the criteria in the subquery.

Copyright © 1988-2007, Oracle. All rights reserved.

The following code adds the criteria for the subquery. Because this is the first non-having criteria in a select statement, the Logical property is set as %Query_CombNotUsed. Then the first expression is added as a field, and set to be equal to the second expression, PPLTOOLS. This is the where PROJECTNAME = 'PPLTOOLS' portion of the SQL statement.

```
/* Need criteria - PROJECTNAME = 'PPLTOOLS'  - in the subquery */
&SubQryCrit1 = &SubQrySel.AddCriteria("FirstSubQryCrit");

/* First criteria will not have any logical AND/OR */
   &SubQryCrit1.Logical = %Query_CombNotUsed;

   &SubQryCrit1.Expr1Type = %Query_ExprField;
   &SubQryCrit1.AddExpr1Field(&SubQryRec.RecordAlias, "PROJECTNAME");
&SubQryCrit1.Operator = %Query_CondEqual;

/* So that the criteria is constructed as - PROJECTNAME = 'PPLTOOLS'*/
&SubQryCrit1.Expr2Type = %Query_ExprConstant;
&SubQryExpr1 = &SubQryCrit1.AddExpr2Expression1();
&SubQryExpr1.Text = "PPLTOOLS";
   &SubQryCrit1.Expr2Expression1 = &SubQryExpr1;
```

11. Add the second criteria to the main select.

    Add the second criteria. Remember to set the type for the expression field first. Because this is the second criteria, we need a logical AND to state that this criteria is used with the first criteria.

```
/* Second Criteria, which is for RECTYPE = 0 */
&MyCriteria2 = &MainQrySel.AddCriteria("SecondCriteria");

&MyCriteria2.Expr1Type = %Query_ExprField;
&MyCriteria2.AddExpr1Field(&MainRec.RecordAlias, "RECTYPE");

/* Since this is second criteria, we need a logical AND to state that - AND⇒
 RECTYPE = 0 */

&MyCriteria2.Logical = %Query_CombAnd;

&MyCriteria2.Operator = %Query_CondEqual;
&MyCriteria2.Expr2Type = %Query_ExprConstant;

&MyCrit2Expr2 = &MyCriteria2.AddExpr2Expression();
&MyCriteria2.Expr2Expression1 = &MyCrit2Expr2;
&MyCrit2Expr2.Text = "0";
```

12. Save the data.

    When you execute the Save method, the new query is saved to the database.

```
&Rslt = &MyQuery.Save();
If &Rslt <> 0 Then
```

    The Save method returns a numeric value: 0 if successful. You can use this value to do error checking.

13. Check Errors

 Copyright © 1988-2007, Oracle. All rights reserved.

You can check if there were any errors using the PSMessages property on the session object.

```
   /* save didn&rsquo;t complete */
   &COLL = &MySession.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
      /* do error processing */
   End-For;
   &COLL.DeleteAll();
End-if;
```

If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error. As you correct each error, you may want to delete it from the PSMessages collection.

Using Outer Joins

The following PeopleCode query uses outer joins:

```
 Local ApiObject &aQuery, &aQrySelCol;
  Local ApiObject &aQryRcd, &aQryRcd2;
  Local ApiObject &aQryFld, &aQryFld2;
  Local ApiObject &aQrySel, &Criteria1;
  Local number &Rslt;

  &aQuery = %Session.GetQuery();
  &aQuery.Create("TEST1", False, %Query_Query, "PIA Test 1", "Creating Test⇒
Query1 from PIA Page");

  &aQrySel = &aQuery.AddQuerySelect();

  &aQryRcd = &aQrySel.AddQueryRecord("JOB");
  &aQryRcd.RecordAlias = "A";

  &aQryRcd2 = &aQrySel.AddQueryRecord("PERSONAL_DATA");
  &aQryRcd2.RecordAlias = "B";
  &aQryRcd2.JoinType = %Query_JoinLeftOuter;
  &aQryRcd2.JoinAlias = "A";

  &aQryFld = &aQrySel.AddQuerySelectedField("JOB", "A", "EMPLID", "EMPLID");
  &aQryFld.ColumnNumber = 1;
  &aQryFld.HeadingType = %Query_HdgRftShort;

  &aQryFld2 = &aQrySel.AddQuerySelectedField("PERSONAL_DATA", "B", "NAME",⇒
"NAME");
  &aQryFld2.ColumnNumber = 2;
  &aQryFld.HeadingType = %Query_HdgRftShort;

  &Criteria1 = &aQrySel.AddCriteria("JoinCriteria");
  &Criteria1.Logical = %Query_CombNotUsed;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&Criteria1.Expr1Type = %Query_ExprField;
&Criteria1.AddExpr1Field("A", "EMPLID");
&Criteria1.Operator = %Query_CondEqual;
&Criteria1.Expr2Type = %Query_ExprField;
&Criteria1.AddExpr2Field1("B", "EMPLID");
&Criteria1.OJAlias = "B";

&Rslt = &aQuery.Save();
```

The above PeopleCode program produces the following SQL:

```
SELECT A.EMPLID, B.NAME
FROM (PS_JOB A LEFT OUTER JOIN PS_PERSONAL_DATA B ON A.EMPLID=B.EMPLID)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 31

Record Class

This chapter provides an overview of Record class and discusses the following topics:

- Shortcut considerations.
- Record methods used to create SQL statements.
- Record object declaration.
- Scope of a record object.
- Record class built-in functions.
- Record class reference.

Understanding Record Class

A record object, instantiated from the Record class, is a single instance of a data within a row and is based on a record definition. A record object consists of one to *n* fields.

If a record object is instantiated using GetRecord (either the function or the method), the record object that is instantiated references the record from the current row in the data buffers, and its associated data.

If a record object is instantiated using the CreateRecord function, the record object that's instantiated is a freestanding record definition with its component set of field objects in the data buffer. The fields created by this function are initialized to null values, that is, they do *not* contain any data. Default processing is not performed. You can select into this record object using the SelectByKey method. You can also select into this record object using SQLExec.

CopyFieldsTo is a commonly used method for the record class. Name, IsChanged, and FieldCount are commonly used properties.

The record class is one of the data buffer access classes.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Shortcut Considerations

An expression of the form

```
RECORD.recname.property
```

Copyright © 1988-2007, Oracle. All rights reserved.

or

```
RECORD.recname.method(.  .  .)
```

is converted to an object expression by using **GetRecord**(**RECORD.***recname*). For example, the following two lines of code are equivalent:

```
&MyRow = RECORD.EMPL_CHKLST_ITEM.ParentRow;
&MyRow = GetRecord(RECORD.EMPL_CHKLST_ITEM).ParentRow;
```

In addition, the default method for the Record class is the GetField method. This means you can access a field by just specifying the field name, instead of using GetField. For example, the following two lines of code, which both disable the EMPLID field, are equivalent:

```
&MyRec.EMPLID.Enabled = False;
&MyRec.GetField(FIELD.EMPLID).Enabled = False;
```

Note. If the field you're accessing has the same name as a record property (such as, Name) you can't use this method for accessing the field. You must use the GetField method.

Record Methods Used to Create SQL Statements

You can use the following record object methods to build and execute SQL statements:

- Delete
- SelectByKey
- Insert
- Save
- Update

The methods that result in a database update (specifically, UPDATE, INSERT, and DELETE) can only be issued in the following events:

- SavePreChange
- WorkFlow
- SavePostChange
- FieldChange

Remember that record UPDATEs, INSERTs, and DELETEs go directly to the database server, not to the Component Processor (although you can *view* data in the buffer using the PeopleCode debugger and other data buffer access classes). If a record method assumes that the database has been updated based on changes made in the component, that record method can be issued only in the SavePostChange event, because before SavePostChange none of the changes made to page data has actually been written back to the database.

If your application is repeating the same instruction many times, such as doing a million INSERTs, you should use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 31, “Record Class,” Delete, page 1385

Chapter 31, “Record Class,” SelectByKey, page 1394

Chapter 31, “Record Class,” Insert, page 1390

Chapter 31, “Record Class,” Update, page 1400

Chapter 38, “SQL Class,” page 1563

Data Type of a Record Object

Record objects are declared as type Record. For example,

```
Local Record &MYRECORD;
```

Scope of a Record Object

A record can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, Component Interface PeopleCode, and so on.

Record Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateRecord

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetRecord

Record Class Methods

In this section, we discuss each Record class method. The methods are discussed in alphabetical order.

CompareFields

Syntax

CompareFields(*recordobject*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The CompareFields method compares all like-named fields of the record object executing the method with the specified record object *recordobject*.

Parameters

Parameter	Description
recordobject	Specify a record object for use in the comparison. The specified record object does not have to refer to the same record as the record object executing the method.

Returns

A Boolean value; True if all like-named fields have the same value.

Example

```
&REC = GetRecord(RECORD.OP_METH_VW);
&REC2 = GetRecord(RECORD.OPC_METH);
If &REC2.CompareFields(&REC) Then
    WinMessage("All liked named fields have the same value");
End-If;
```

See Also

Chapter 31, "Record Class," CopyChangedFieldsTo, page 1380 and Chapter 31, "Record Class," CopyFieldsTo, page 1381.

CopyChangedFieldsTo

Syntax

CopyChangedFieldsTo(*recordobject*)

Description

The CopyChangedFieldsTo method copies all like-named field values that have changed from the record object executing the method to the specified record object *recordobject.* This copies only changed field values. To copy all field values, use the CopyFieldsTo method.

Note. This method works only with database records. The Component Processor doesn't track the contents of work records, so there is no changed value to use for copying changed fields.

Parameters

Parameter	Description
recordobject	Specify a record object to be copied to. The specified record object does not have to refer to the same record as the record object executing the method.

Returns

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local Record &REC, &REC2;

&REC = GetRecord(RECORD.OPC_METH);

/* make changes to the values of fields in the record */

&REC2 = CreateRecord(RECORD.OPC_METH_WORK);

&REC.CopyChangedFieldsTo(&REC2);
```

See Also

Chapter 31, "Record Class," CompareFields, page 1379 and Chapter 31, "Record Class," CopyFieldsTo, page 1381.

CopyFieldsTo

Syntax

CopyFieldsTo(*recordobject*[, *DontCopyUnusedInSource* [, *DontCopyUnusedInDestination*]] [,IsBatch])

Description

The CopyFieldsTo method copies all like-named field values from the record object executing the method to the specified record object *recordobject.* This copies all field values.

To copy only changed field values, use the CopyChangedFieldsTo method.

To restrict the copy to fields that have been marked as unused (set using the SetDBFieldNotUsed function) you can specify either *DontCopyUnusedInSource* or *DontCopyUnusedInDestination.*

Note. If you are copying to a derived work record, the IsChanged flag for the record is not set. Copying fields to a database record does set the IsChanged flag to True.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
recordobject	Specify a record object to be copied to. The specified record object does not have to refer to the same record as the record object executing the method.
DontCopyUnusedInSource	Specify this parameter to be True if you want to restrict the copy of fields to not include fields marked as unused in the record of the object performing the CopyFieldsTo method. This is the source record. The default value for this parameter is False, which means that there are no restrictions on the copy of unused fields as defined in the source record.
DontCopyUnusedInDestination	Specify this parameter to be True if you want to restrict the copy of fields to not include fields marked as unused in the recordobject record which is the destination record. **Note.** To specify this record to be true you must specify a value for *DontCopyUnusedInSource*. The default value for this parameter is False meaning that there are no restrictions on the copy of unused fields as defined in the destination record.
IsBatch	Specify whether this method is being called from an online program or from a batch program (such as an Application Engine program.) The default is false. Using this method may improve batch performance. When set to true, the system uses the FieldNotUsed information directly from the field properties already in the memory

Returns

None.

Example

```
Local Record &REC, &REC2;

&REC = GetRecord(RECORD.OPC_METH);
&REC2 = CreateRecord(RECORD.OPC_METH_WORK);
&REC.CopyFieldsTo(&REC2);
```

In the following example, records are copied into a record after being fetched.

```
Component number &displayNum;
Local SQL &sql;
Local Rowset &rs, &rs2;

&level0 = GetLevel0();
&displayNum = WS_NUM_ORDERS;

&rs = GetRowset(Record.WS_ORD_HDR_VW);
&rs.Flush();
WinMessage("1");
&sql = CreateSQL("%selectall(:1) where BUSINESS_UNIT=:2", Record.WS_ORD_HDR_VW,⇒
 "M04");
WinMessage("2");
&rec = CreateRecord(Record.WS_ORD_HDR_VW);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
For &i = 1 To &displayNum

   If &sql.Fetch(&rec) Then

      &rs.InsertRow(&i);

      &rec.copyfieldsto(&rs.GetRow(&i).WS_ORD_HDR_VW);

      &rs2 = &rs.GetRow(&i).GetRowset(1);
      &rs2.Select(Record.WS_ORD_LINE_VW, "where BUSINESS_UNIT=:1 and ORDER_NO=:2",⇒
 &rec.BUSINESS_UNIT.value, &rec.ORDER_NO.value);

      /* Hide rows that do not contain Products */
      If &rs2 = Null Or None(&rs2.GetRow(1).WS_ORD_LINE_VW.ORDER_INT_LINE_⇒
NO.Value) Then
         For &j = 1 To &rs2.ActiveRowCount
            &rs2.GetRow(&j).Visible = False;
         End-For;
      End-If;
   End-If;
End-For;

&rs.GetRow(&i).Visible = False;
```

The following example is for unused record fields:

```
Local Record &From = CreateRecord(Record.QE_UPS_TIME);

/* setup the initial values */

&From.QE_FROM_ZIP.Value = "12345";
&From.QE_TO_ZIP.Value = "67890";
&From.QE_UPS_TIME_BUTTON.Value = "A";
&From.DESCRLONG.Value = "This is the from record.";

/* Now make one of the fields unused */
Local string &ToZip_Value = "77777";

/* start out clean with &To */
Local Record &To = CreateRecord(Record.QE_UPS_TIME);
&To.QE_TO_ZIP.Value = &ToZip_Value;
If SetDBFieldNotUsed(Field.QE_TO_ZIP, True) <> %MetaDataChange_Success Then
   MessageBox(0, "", 0, 0, "SSetDBFieldNotUsed(Field.QE_TO_ZIP, TRUE) fails??");
End-If;
/* Copy no unused fields from the source record */
&From.CopyFieldsTo(&To, True /* no unused fields from source */);

/* Copy to no unused fields in the destination */
&From.CopyFieldsTo(&To, False /* all fields from source */, True /* no unused⇒
 fields in dest */);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Copy no unused fields either in source or destination */

&From.CopyFieldsTo(&To, True /* no unused fields from source */, True /* no unused⇒
 fields in dest */);

/* Now finally make that field good again */

If SetDBFieldNotUsed(Field.QE_TO_ZIP, False) <> %MetaDataChange_Success Then
   MessageBox(0, "", 0, 0, "SSetDBFieldNotUsed(Field.QE_TO_ZIP, False) fails??");
End-If;
```

See Also

Chapter 31, "Record Class," CompareFields, page 1379, Chapter 31, "Record Class," CopyChangedFieldsTo, page 1380, Chapter 31, "Record Class," IsChanged, page 1402 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," SetDBFieldNotUsed.

DBPatternMatch

Syntax

DBPatternMatch(*Value*, *Pattern*, *CaseSensitive*)

Description

Use the DBPatternMatch function to match the string in *Value* to the given pattern.

You can use wildcard characters % and _ when searching. % means find all characters, while _ means find a single character. For example, to find if the string in *Value* started with the letter M, use "M%" for *Pattern*. To find either DATE or DATA, use "DAT_" for *Pattern*.

These characters can be escaped (that is, ignored) using a \. For example, to search for a value that contains the character %, use \% in *Pattern*.

If *Pattern* is an empty string, this function retrieves the value based only on the specified case sensitivity (that is, specifying "" for *Pattern* is the same as specifying "%").

Parameters

Parameter	Description
Value	Specify the string to be searched.
Pattern	Specify the pattern to be used when searching.
CaseSensitive	Specify whether the search is case-sensitive. This parameter takes a Boolean value: True, the search is case-sensitive, False, it is not.

Returns

A Boolean value: True if the string matches the pattern, False otherwise.

 Copyright © 1988-2007, Oracle. All rights reserved.

Delete

Syntax

Delete()

Description

The Delete method uses the key fields of the record and their values to build and execute a DELETE SQL statement which deletes the record (row of data) from the SQL data table.

This method, like the DeleteRow Rowset class method, initially marks the record or row as needing to be deleted. At save time the row is actually deleted from the database and cleared from the buffer.

Because this method results in a database change, it can be issued only in the following events:

- SavePreChange
- WorkFlow
- SavePostChange

If your application is repeating the same instruction multiple times, such as doing a million DELETEs, use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

For every record deleted by the Delete method, if the set language is not the base language and the record has related language records, the Delete method tries to do related language processing.

Parameters

None.

Returns

The result is True on successful completion, False if the record was not found. Any other conditions cause termination.

Example

Suppose that KEYF1 and KEYF2 are the two key fields of record definition MYRECORD. The following code deletes the database record that has KEYF1 equal to "A" and KEYF2 equal to "X":

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
&REC.Insert();
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 31, "Record Class," SelectByKey, page 1394, Chapter 31, "Record Class," Insert, page 1390 and Chapter 31, "Record Class," Update, page 1400.

Chapter 33, "Rowset Class," DeleteRow, page 1426

Chapter 38, "SQL Class," page 1563

Enterprise PeopleTools 8.49 PeopleBook: Global Technology, "Using Related Language Tables"

ExecuteEdits

Syntax

```
ExecuteEdits([editlevel]);
```

where *editlevels* is a list of values in the form:

```
editlevel1 [+ editlevel2] .  .  .]);
```

and where *editleveln* is one of the following constants:

%Edit_DateRange

%Edit_OneZero

%Edit_PromptTable

%Edit_Required

%Edit_TranslateTable

%Edit_YesNo

Description

The ExecuteEdits method executes the standard system edits on every field in the record. The types of edits performed depends on the *editlevel*. If no *editlevel* is specified, all system edits are executed. All *editlevels* are already defined for the record definition or for the field definition, that is:

- Reasonable Date Range (Is the date contained within the specified reasonable date range?)
- 1/0 (Do all 1/0 fields contain only a 1 or 0?)
- Prompt Table (Is field data contained in the specified prompt table?)
- Required Field (Do all required fields contain data? For numeric or signed fields, it checks that they do not contain NULL or 0 values.)
- Translate Table (Is field data contained in the specified translate table?)
- Yes/No (Do all yes/no fields only contain only yes or no data?)

If any of the edits fail, the status of the property IsEditError is set to False for the record. The field property EditError is set to True for any fields that are in error. In addition, the Field class properties MessageNumber and MessageSetNumber are set to the number of the returned message and message set number of the returned message, for each field in error.

You must use the SetEditTable method to set the prompt tables for fields that are defined with %EditTable in the record definition.

 Copyright © 1988-2007, Oracle. All rights reserved.

If you're running an Application Engine program, and you want to do set based ExecuteEdits (as opposed to row-by-row) consider using the meta-SQL construct %ExecuteEdits.

See Chapter 31, "Record Class," SetEditTable, page 1398; Chapter 15, "Field Class," EditError, page 504; Chapter 15, "Field Class," MessageNumber, page 512; Chapter 15, "Field Class," MessageSetNumber, page 513 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Engine*, "Using Meta-SQL and PeopleCode," %ExecuteEdits.

Considerations for ExecuteEdits and SetEditTable

If an effective date is a key on the prompt table, and the record being edited doesn't contain an EFFDT field, the current datetime is used as the key value.

If a SETID is a key on the prompt table, and the record being edited doesn't contain a SETID field, the system looks for SETID on the other records in the current row first, then searches parent rows.

For all other keys, the value used to "select" the correct row in the prompt table record comes only from the record executing the method. No other records (or key field values) are used. You may get unexpected results if not all the keys for the prompt table are filled in (or filled in correctly.)

Considerations for ExecuteEdits and Numeric Fields

A zero may or may not be a valid value for a numeric field. ExecuteEdits handles numeric fields in different ways, depending on whether the field is required and a prompt table is used:

- If the numeric field is required and there is no prompt edit: 0 is considered valid, so numeric fields never fail the required test.
- If the numeric field is required and there is a prompt edit: Prompt edit is run and the value must be valid.
- If the numeric field is not required and there is a prompt edit: 0 is considered valid, so the prompt edit is skipped.

Parameters

Parameter	Description
editlevel	Specifies the standard system edits to be performed against every field on every record. If *editlevel* isn't specified, all system edits are performed. *editlevel* can be any of the following system variables. • %Edit_DateRange • %Edit_OneZero • %Edit_PromptTable • %Edit_Required • %Edit_TranslateTable • %Edit_YesNo

Returns

None.

Example

The following is an example of a call to execute Required Field and Prompt Table edits:

Copyright © 1988-2007, Oracle. All rights reserved.

```
&REC.ExecuteEdits(%Edit_Required + %Edit_PromptTable);
```

The following is an example showing how ExecuteEdits() could be used:

```
&REC.ExecuteEdits();
If &REC.IsEditError Then
   LogError();  /*application specific call */
End-If;
```

See Also

Chapter 31, "Record Class," SetEditTable, page 1398, Chapter 31, "Record Class," IsEditError, page 1403, Chapter 15, "Field Class," EditError, page 504, Chapter 15, "Field Class," MessageNumber, page 512 and Chapter 15, "Field Class," MessageSetNumber, page 513.

GetField

Syntax

GetField({*n* | FIELD.*fieldname*})

Description

The GetField method instantiates a field object for the specified field associated with the record. This is the default method for the record object. This means that any record object, followed by a parameter list, acts as if GetField is specified.

Note. If the field you're accessing has the same name as a record property (that is, Name or FieldCount) you can't use the default method for accessing the field. You must specify GetField.

For example, the following is invalid for accessing the value of a field called NAME:

```
&NUMBER = &REC.NAME.Value;
```

You must use the following code to get the value of a field called NAME:

```
&NUMBER= &REC.GetField(FIELD.NAME).Value;
```

Parameters

Parameter	Description
n | **FIELD**.*fieldname*	Specify a field to be used for instantiating the field object. You can specify either *n* or **FIELD.***fieldname*. Specifying *n* creates a field object for the *nth* field in the record. This might be used if writing code that needs to examine all fields in a record without being aware of the name. Specifying **FIELD.***fieldname* creates a field object for the field *fieldname*.

Note. There is no way to predict the order the fields will be accessed. Use the *n* option only if the order in which the fields are processed doesn't matter.

Returns

A field object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&REC.GetField(FIELD.CHARACTER).Value = "Hello";
```

As GetField is the default method for a record object, the following code is identical to the previous code:

```
&REC.CHARACTER.VALUE = "Hello";
```

The following code creates an array containing all the names of all the fields in a related language record.

```
Local array of string &FIELD_LIST_ARRAY;
&FIELD_LIST_ARRAY = CreateArray();
For &I = 1 to &REC_RELATED_LANG.FieldCount
   &FIELD_LIST_ARRAY.Push(&REC_RELATED_LANG.GetField(&I).Name);
End-For;
```

The GetField method requires either FIELD.fieldname or a number. It won't take a string field name. However, you can use the @ operator to convert the string field name into a **FIELD**.*fieldname* reference, as follows:

```
&REC = GetRecord();
&REC2 = GetLevel0(1).EMPL_CHECKLIST;
&FIELD2 = &REC.GetField(@ "FIELD." | &REC2.getfield(&I).Name);
```

The following code converts field name strings (using the @ symbol) to component names to get the value of all the fields in a subrecord. Note the code in **Keyword style**.

```
Function get_draft_dst_codes
   /* Load dst id codes into record structures */
   &DSTCODES = CreateRecord(RECORD.DR_DST_CODE_SBR);
   &DRAFTITEM = GetRecord(RECORD.DRAFT_ITEM);
   &DRAFTITEM.CopyFieldsTo(&DSTCODES);
   /* If any missing get from AR dist code, then Draft Type-BU, then BU */
   For &I = 1 To &DSTCODES.FieldCount
      If None(&DSTCODES.GetField(&I).Value) Then
         get_ar_dst_code();
         &NAME = &DSTCODES.GetField(&I).Name;
         If All(&DST.GetField(@ ("FIELD." | &NAME)).Value) Then
            &DSTCODES.GetField(&I).Value = &DST.GetField(@("FIELD." |⇒
 &NAME)).Value;
         Else
            If All(&R_DRAFTBU.GetField(@("FIELD." | &NAME)).Value) Then
               &DSTCODES.GetField(&I).Value = &R_DRAFTBU.GetField(@("FIELD." |⇒
 &NAME)).Value;
            Else
               &DSTCODES.GetField(&I).Value = &R_ARBU.GetField(@("FIELD." |⇒
 &NAME)).Value;
<!--### 3 line break(s) inserted ###-->           End-If;
         End-If;
      End-If;
   End-For;
   /* Copy the defaulted values back to draft item */
   &DSTCODES.CopyFieldsTo(&DRAFTITEM);
End-Function;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetField

Insert

Syntax

```
Insert()
```

Description

The Insert method uses the field names of the record and their values to build and execute an Insert SQL statement which adds the given record (row of data) to the SQL table.

Because this method results in a database change, it can only be issued in the following events:

- SavePreChange
- WorkFlow
- SavePostChange

If your application is repeating the same instruction many times, such as doing a million INSERTs, use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

If you're using a record created using CreateRecord, all fields are initially set to "", 0, or NULL, depending on the type of field. If you don't specify values for these fields, these initial values are written to the database when the insert is executed. If you want the default values for the fields inserted instead, use the SetDefault method prior to using the Insert method.

For every record inserted with the Insert method, if the set language is not the base language and the record has related language records, the Insert method tries to do related language processing.

See Also

Chapter 38, "SQL Class," page 1563, *Enterprise PeopleTools 8.49 PeopleBook: Global Technology*, "Using Related Language Tables" and Chapter 31, "Record Class," SetDefault, page 1398.

Parameters

None.

Returns

The result is True on successful completion, False if there was a record with those keys already in the database, that is, if a duplicate record was found. Any other conditions cause termination.

Example

Suppose that KEYF1 and KEYF2 are the key fields of record definition MYRECORD, and that this record definition also contains the field definitions MYRF3, MYRF4. The following code adds a MYRECORD record (row of data) to the database, with KEYF1 set to "A", KEYF2 set to "B", MYRF3 set to "X", and MYRF4 set to "Y":

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
&REC.Insert();
```

See Also

Chapter 31, "Record Class," Delete, page 1385, Chapter 31, "Record Class," SelectByKey, page 1394 and Chapter 31, "Record Class," Update, page 1400.

Save

Syntax

Save([*CopyToOriginal*])

Description

The Save method saves the record to the database in two steps.

1. Check to see that it is safe to save this record to the database.
2. Either insert this record or update an existing record.

To accomplish the first step and determine that it is safe to save the record, this method uses similar logic to that used in the component processor when it saves a record. To do that a "select for update" SQL statement is executed. If, as a result of this statement, the record does not exist in the database, this method simply inserts the record into the database. All the processing that occurs for the insert method then occurs.

If the result of the "select for update" statement returns a record, the values of all the fields in the database record are compared to the original values of the record executing this method. If they are equal, indicating that no other user has updated the record, the record in the database is updated. All the processing that occurs for the update method then occurs. If the result of the above comparison fails, indicating that there has been an update of the database record in the meantime, perhaps by some other user, the save method fails and returns a false result.

For the update process to succeed, the record object must contain both original and changed values of the data. It is crucial to get data loaded into the record object correctly. To do this, do one of the following:

- Derive the record object from the component buffers.
- Derive the record object from a rowset filled using the Fill method.
- Load the data using the record class SelectByKey and SelectByKeyEffdt methods specifying the optional Boolean parameter to be False.

See the note and example below.

Of course when you use Save to simply insert data you do not need load data into the record object this way.

Use this method when you are writing applications that do not use the facilities that come predelivered with the component processor and you must deal with issues of data contention.

For example, suppose you are writing an application that deals with course registration. You have to handle changing the number of students who are registered carefully if it is possible for more than one user to be registering for the same course at the same time. The normal way is to use the component processor to handle data contention by placing the record that deals with the number of students registering on some page. Then if two users try to update the same record one of them receives a message indicating that the page has been updated by another user.

Copyright © 1988-2007, Oracle. All rights reserved.

If you write an application, outside the component processor, that relies on the record class to manage the number of students enrolled, it is not sufficient to use the record class Insert or Update methods to maintain data integrity. Suppose one user registers for the course and sees there are 10 other registrants. At the same time another user registers for the course and sees that there are also 10 registrants. Suppose the maximum is 11 students. Both users at this point believe they can register since at the point they accessed the data from the database there were only 10 students. So far, so good. The trouble comes when they save their registrations. If the application uses the update method, after simply adding one to the number of registrants, both updates will succeed indicating there are now 11 students in the course when in fact there are now 12. It is for cases such as these that the Save method applies since it guarantees that one of the save calls will fail. It is up to the application then to let the user know that while they were thinking about it someone else registered for the course and took the last spot.

The Save method is important to maintain data integrity. However, it can execute more slowly than saving within the component processor.

Since this method results in a database change, and does either insert or update processing, all the considerations and implications of using the Insert or Update methods apply. See the description of those methods for more detail.

The parameter *CopyToOriginal* indicates what action should occur on a successful save. If this parameter is true, the system copies the current changed buffers for the record into the original buffers for the record. This means that subsequent Saves, without reloading data from the database will run correctly. However if there are database specific updates to this record occurring outside of this application's control (such as, using database specific triggers,) it is always advisable to reload the data from the database.

The default value for this optional parameter is false, indicating that no update of the original buffers for this record occurs. If you are certain that there are no database updates occuring outside of your application's on save processing, you can specify this optional parameter to be true and save the overhead of having to reload the data for this record from the database. If you are unsure you should always reload the record data after calling the Save method.

Note. This method does *not* update fields marked as System maintained in the record definition in PeopleSoft Application Designer.

Since this save is done outside the component processor save processing, no save processing PeopleCode that is associated with this record is executed.

Use of this method that results in an update, and requires that the data in the record object is such that both a before and after image of the data is kept. When the system loads data into the component buffers there is a concept of the "original data" (data loaded from the database) and "changed data" (data changed by the user or the application). This allows the system to determine when saving, whether the data has been changed by another user. If you use this method with record objects not derived from the component buffers you need to make sure that the data in the record object has both a before and after image otherwise the save method fails. Record objects derived from rowsets that have been populated using the Fill method will not produce an error. However if you load data into a record object using either the SelectByKey or SelectByKeyEffdt record class methods, you need to specify the *CopyToOriginal* parameter so the system will load the data into the original data buffers, not the changed data buffers. To do that specify the optional parameter to be False. See the example below.

See Also

Chapter 31, "Record Class," Insert, page 1390 and Chapter 31, "Record Class," Update, page 1400.

Enterprise PeopleTools 8.49 PeopleBook: Global Technology, "Using Related Language Tables"

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
CopyToOriginal	Specify what action should occur on a successful save. This parameter takes a Boolean value: true, then the system copies the current changed buffers for the record into the original buffers for the record, false, the buffers are not copied. The default value is false.

Returns

The result is True on successful completion, False if there were no changed fields in the record or it was not safe to save this record (see above for details). Any other conditions cause termination.

Note. It is not possible to ignore the return code from the Save method in your PeopleCode.

Example

Suppose that KEYF1 and KEYF2 are the key fields of record definition MYRECORD, and that this record definition also contains the record field definitions MYRF3, MYRF4. The following code tries to save MYRECORD record in the database with KEYF1 set to "A", KEYF2 set to "B", setting MYRF3 to "X" and MYRF4 to "Y". This example works with a brand new record object and results in an insert if successful.

```
/* First -- create a brand new record */
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";

/* Second -- update values as needed */
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
/* Finally save it */
if &REC.Save() then
    /* handle the successful return */
else
    /* handle the unsuccessful return message to user? Try again? */
end-if;
```

This example works with an existing record and results in an update if successful.

```
/* First - create the record and get the data from the db */
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.SelectByKey(False);
/* False retrieves data into the original, not changed buffers. */

/* Second - update values as needed */
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Finally save it */
if &REC.Save() then
    /* handle the successful return */
else
    /* handle the unsuccessful return - message to user? Try again? */
end-if;
```

See Also

Chapter 31, "Record Class," Insert, page 1390, Chapter 31, "Record Class," Delete, page 1385, Chapter 31, "Record Class," SelectByKey, page 1394 and Chapter 31, "Record Class," Update, page 1400.

Chapter 31, "Record Class," SelectByKeyEffDt, page 1396

SearchClear

Syntax

SearchClear()

Description

The SearchClear method clears the field values for all search keys for a record.

Considerations Using SearchClear and SearchDefault

The field property SearchDefault sets a field to its default value (if there is one) immediately after SearchInit PeopleCode finishes. SearchDefault overrides SearchClear. If you call SearchClear for a record, then use SearchDefault for a field, the field is set to its default value and the search key values for the rest of the record are cleared.

Parameters

None.

Returns

None.

Example

```
Local Record &REC;

&REC = GetRecord();
&REC.SearchClear();
```

See Also

Chapter 15, "Field Class," SearchDefault, page 514 and Chapter 15, "Field Class," SearchClear, page 498.

SelectByKey

Syntax

SelectByKey([*UseChangedBuffers*])

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The SelectByKey method uses the key field names of the record and their values to build and execute a Select SQL statement. The field values are then fetched from the database SQL table into the record object executing the method, and the Select statement is closed.

Note. You can't use this method in dynamic views.

If you don't specify all the key fields for a record, those you exclude are added to the Where clause with the condition equal to a blank value. If not all keys are set and more than one row is retrieved, you won't receive an error and SelectByKey won't fetch any data.

SelectByKey returns a single row of data. To fetch more than one row, use the SQL object.

For every record read by the SelectByKey method, if the set language is not the base language and the record has related language records, the SelectByKey method tries to do related language processing.

You can also select into a record using SQLExec and meta-SQL. For example, if you need to use SelectByKey with an effective date, you can use the meta-SQL %SelectByKeyEffDt in a SQLExec statement. In the following example, &RECOBJ is the name of a record created using the CreateRecord function.

```
SQLExec("%SelectByKeyEffDt(:1, :2)", &RECOBJ, %Date, &RECOBJ);
```

See Also

Chapter 38, "SQL Class," page 1563
Enterprise PeopleTools 8.49 PeopleBook: Global Technology, "Using Related Language Tables"
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SQLExec

Parameters

Parameter	Description
UseChangedBuffers	Specify whether to retrieve the data into the changed or original buffers. The system keeps track of the original and changed values. This parameter takes a Boolean value: true, retrieve into the changed buffers, False, use the original buffers. The default value is true. is to retrieve the data into the changed buffers. To use this parameter with the Save method, it is important that data be loaded into the original buffers in order for the save logic to work. To do that specify a value of false for this parameter. See Chapter 31, "Record Class," Save, page 1391.

Returns

The result is true on successful completion, false if the record was not found. Any other conditions cause termination.

If false is returned, the system resets the record values to their default values for scrolls that are not derived. If you do a manual insert afterwards (using the Insert method) you may insert a record with null keys.

Example

Suppose that KEYF1 and KEYF2 are the two key fields of record definition MYRECORD. The following code reads the database record that has KEYF1 equal to "A" and KEYF2 equal to "X" into the &REC record object:

```
Local record &REC;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "X";
&REC.SelectByKey();
```

The following example verifies if the method completes successfully before using the Insert method.

```
if not &Rec.SelectByKey() then

     /* Must duplicate this key field population before the Insert - else can get⇒
 'blank' row */

      &Rec.KEY1.Value = ...;
      &Rec.KEY2.Value = ....;
      ....

     &Rec.Insert();

end-if;
```

See Also

Chapter 31, "Record Class," Delete, page 1385, Chapter 31, "Record Class," Insert, page 1390, Chapter 31, "Record Class," Update, page 1400 and Chapter 31, "Record Class," Save, page 1391.

SelectByKeyEffDt

Syntax

SelectByKeyEffDt(*Date*[, *UseChangedBuffers*])

Description

Use the SelectKeyByEffDt to build and execute a SQL Select statement to get the current effective row based on an "as of date". The field values are then fetched from the database SQL table into the record object executing the method, and the Select statement is closed.

Note. You can't use this method in dynamic views.

SelectByKeyEffDt returns a single row of data. To fetch more than one row, use the SQL object.

For every record read by the SelectByKeyEffDt method, if the set language is not the base language and the record has related language records, the SelectByKeyEffDt method tries to do related language processing. The system assumes that both the base table and the related language table are effective-dated and that the effective-date keys are the same in each.

To use other keys, more than just the effective date, use the SelectByKey Record class method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Date	Specify the "as of" date you want to use as the effective date for the Select statement.
UseChangedBuffer	Specify whether to retrieve the data into the changed or original buffers. The system keeps track of the original and changed values. This parameter takes a Boolean value: true, retrieve into the changed buffers, False, use the original buffers. The default value is true. is to retrieve the data into the changed buffers. To use this parameter with the Save method, it is important that data be loaded into the original buffers in order for the save logic to work. To do that specify a value of false for this parameter. See Chapter 31, "Record Class," Save, page 1391.

Returns

The result is True on successful completion, False if the record was not found. Any other conditions cause termination.

If false is returned, the system resets the record values to their default values for scrolls that are not derived. If you do a manul insert afterwards (using the Insert method) you may insert a record with null keys.

Example

```
Local Record &Rec;
Local any &Date;

&Date = %Date;
&Rec = CreateRecord(Record.COMPANY_TBL);
&Rec.COMPANY.Value = "CCB";

If &Rec.SelectByKeyEffDt(&Date) Then
   &Out = &Rec.DESCR.Value | " as of " | &Rec.EFFDT.Value | "->" |⇒
 &Rec.STREET1.Value;
   WinMessage("Found this company: " | &Out);
Else
   WinMessage("Not there!");
End-If;
```

See Also

Chapter 31, "Record Class," Delete, page 1385, Chapter 31, "Record Class," Insert, page 1390, Chapter 31, "Record Class," Update, page 1400, Chapter 31, "Record Class," Save, page 1391 and Chapter 31, "Record Class," SelectByKey, page 1394.

Chapter 38, "SQL Class," page 1563

Enterprise PeopleTools 8.49 PeopleBook: Global Technology, "Using Related Language Tables"

Copyright © 1988-2007, Oracle. All rights reserved.

SetDefault

Syntax

```
SetDefault()
```

Description

SetDefault sets the value of every field in the record to a null value, or to a default, depending on the type of field.

- If this method is used against data from the Component buffers, the next time default processing occurs, it is set to its default value: either a default specified in its record field definition or one set programmatically by PeopleCode located in a FieldDefault event. If neither of these defaults exist, the Component Processor leaves the field blank.
- If this method is used with a field that isn't part of the data buffer (for example, a field in a record object instantiated with CreateRecord) the field is automatically set to its default value if one is set for the field, not for the record field. Any FieldDefault PeopleCode does not run on these types of fields. To set the default values for just one field, use the SetDefault field class method.

Blank numbers correspond to zero on the database. Blank characters correspond to a space on the database. Blank dates and long characters correspond to NULL on the database. SetDefault gives each field data type its proper value.

Parameters

None.

Returns

None.

Example

```
&CHARACTER.SetDefault();
```

See Also

Chapter 15, "Field Class," SetDefault, page 499

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Default Processing

SetEditTable

Syntax

```
SetEditTable(%PromptField, RECORD.recordname)
```

Description

The SetEditTable method works with the ExecuteEdits method. It sets the value of a field on a record that has its prompt table defined as *%PromptField* value. %PromptField values are used to dynamically change the prompt record for a field.

There are several steps to setting up a field so that you can dynamically change its prompt table.

To set up a field with a dynamic prompt table:

 Copyright © 1988-2007, Oracle. All rights reserved.

1. Define a field in the DERIVED record called *fieldname*.
2. In the record definition for the field you want to have a dynamic prompt table, define the prompt table for the field as %*PromptField*, where *PromptField* is the name of the field you created in the DERIVED record.

   Note. When you use SetEditTable, you don't have to add a hidden field to the page.

3. Use SetEditTable to dynamically set the prompt table.

%PromptField is the name of the field on the DERIVED work record. RECORD.*recordname* is the name of the record to be used as the prompt table.

```
&REC.SetEditTable("%EDITTABLE1", Record.DEPARTMENT);
&REC.SetEditTable("%EDITTABLE2", Record.JOB_ID);
&REC.SetEditTable("%EDITTABLE3", Record.EMPL_DATA);
&REC.ExecuteEdits();
```

Every field on a record that has the prompt table field %EDITTABLE1 will have the same prompt table, that is, DEPARTMENT.

See Chapter 31, "Record Class," ExecuteEdits, page 1386 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer*, "Creating Record Definitions".

Considerations for SetEditTable and ExecuteEdits

The keys used to "select" the correct row in the prompt table record come only from the record object executing the method. All the keys from the component are not used. You may get unexpected results if not all the keys for the prompt table are filled in (or filled in correctly.)

Parameters

Parameter	Description
%PromptField	Specifies the name of the field in the DERIVED record that has been defined as the prompt table for a field in the record definition.
RECORD.*recordname*	Specifies the name of the record definition that contains the correct standard system edits to be performed for that field in the record.

Returns

None.

Example

```
/* To Set the Prompt Table */
&Der_EditRec = &TSKPRF_Det_Row.GetRecord(Record.DERIVED);
&Prompt_Editname = &Der_EditRec.EDITTABLE.Value;
&Prompt_Edittable = "Record." | &Prompt_Editname;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE", @&Prompt_Edittable);

&Prompt_Editname2 = &Der_EditRec.EDITTABLE2.Value;
&Prompt_Edittable2 = "Record." | &Prompt_Editname2;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE2", @&Prompt_Edittable2);

&Prompt_Editname3 = &Der_EditRec.EDITTABLE3.Value;
&Prompt_Edittable3 = "Record." | &Prompt_Editname3;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE3", @&Prompt_Edittable3);

&Prompt_Editname4 = &Der_EditRec.EDITTABLE4.Value;
&Prompt_Edittable4 = "Record." | &Prompt_Editname4;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE4", @&Prompt_Edittable4);

&Prompt_Editname5 = &Der_EditRec.EDITTABLE5.Value;
&Prompt_Edittable5 = "Record." | &Prompt_Editname5;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE5", @&Prompt_Edittable5);

&Prompt_Editname6 = &Der_EditRec.EDITTABLE6.Value;
&Prompt_Edittable6 = "Record." | &Prompt_Editname6;
&TSKPRF_Det_Rec.SetEditTable("%EDITTABLE6", @&Prompt_Edittable6);

&TSKPRF_Det_Rec.ExecuteEdits(%Edit_Required + %Edit_PromptTable);
```

See Also

Chapter 31, “Record Class,” ExecuteEdits, page 1386, Chapter 31, “Record Class,” IsEditError, page 1403, Chapter 15, “Field Class,” EditError, page 504, Chapter 15, “Field Class,” MessageNumber, page 512 and Chapter 15, “Field Class,” MessageSetNumber, page 513.

Update

Syntax

Update([*KeyRecord*])

Description

The Update method uses the changed fields of the record and their values to build and execute an Update SQL statement, updating the SQL table. If you are updating the key fields of the record, you must supply a record object *KeyRecord* that contains the old values of the key record fields.

Because this method results in a database change, it can only be issued in the following events:

- SavePreChange
- WorkFlow
- SavePostChange

If your application is repeating the same instruction many times, such as doing a million UPDATEs, use the SQL object with the BulkMode property set to True, rather than the record SQL methods.

For every record updated by the Update method, if the set language is not the base language and the record has related language records, the Update method tries to do related language processing.

Note. This method does not update fields marked as system maintained in PeopleSoft Application Designer.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 38, "SQL Class," page 1563 and *Enterprise PeopleTools 8.49 PeopleBook: Global Technology*, "Using Related Language Tables".

Parameters

Parameter	Description
KeyRecord	If you're updating the key fields of the record, you must supply the old key field values in the record object *KeyRecord*.

Returns

The result is True on successful completion, False if the record was not found or (when updating the key fields) a duplicate record was found. Any other conditions cause termination.

Example

Suppose that KEYF1 and KEYF2 are the key fields of record definition MYRECORD, and that this record definition also contains the record field definitions MYRF3, MYRF4. The following code updates the MYRECORD record in the database with KEYF1 set to "A", KEYF2 set to "B", setting MYRF3 to "X" and MYRF4 to "Y":

```
Local record &REC;
&REC = CreateRecord(RECORD.MYRECORD);
&REC.KEYF1.Value = "A";
&REC.KEYF2.Value = "B";
&REC.MYRF3.Value = "X";
&REC.MYRF4.Value = "Y";
&REC.Update();
```

This code updates the MYRECORD record in the database with KEYF1 of "A" and KEYF2 of "B", setting KEYF2 to "C", MYRF3 to "M" and MYRF4 to "N":

```
Local record &REC1, &REC2;
&REC1 = CreateRecord(RECORD.MYRECORD);
&REC2 = CreateRecord(RECORD.MYRECORD);
&REC1.KEYF1.Value = "A";
&REC1.KEYF2.Value = "B";
&REC2.KEYF1.Value = "A";
&REC2.KEYF2.Value = "C";
&REC2.MYRF3.Value = "M";
&REC2.MYRF4.Value = "N";
&REC2.Update(&REC1);
```

See Also

Chapter 31, "Record Class," Delete, page 1385, Chapter 31, "Record Class," SelectByKey, page 1394 and Chapter 31, "Record Class," Insert, page 1390.

Copyright © 1988-2007, Oracle. All rights reserved.

Record Class Properties

In this section, we discuss the Record class properties. The properties are listed in alphabetical order.

FieldCount

Description

This property returns the total number of fields contained in the record. This value is a number.

This property is read-only.

Example

```
WinMessage("This record has this many fields : " | &REC.FieldCount);
```

fieldname

Description

If a field name is used as a property, it accesses the field object with that name. This means the following code:

```
&REC.fieldname
```

acts the same as

```
&REC.GetField(FIELD.fieldname);
```

Note. If the field you're accessing has the same name as a record property (such as, Name) you can't use this method for accessing the field. You must use the GetField method.

If you combine the *fieldname* property with the *recname* property, you obtain the specified field object associated with that record from the row.

```
&ROW.recname.fieldname
```

This property is read-only.

Example

```
&REC.CHARACTER.Enabled = True;
```

IsChanged

Description

This property returns True if any field value on the primary database record of the row has been changed.

Note. This property is for use only with the primary database record. It does not return valid results if used with a work record.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Considerations Using IsChanged

This property and the IsChanged row property do not always return identical values.

If a row in a scroll contains multiple records (such as a primary database record and one or more work records), the IsChanged row property returns True if any of the records in the row are changed. The IsChanged record property returns True only if a specific record, namely, the primary database record, is changed.

If a row is deleted from a scroll, only the primary database record has its IsChanged property marked to False (since the row has been deleted.) Any work records in the row still have their IsChanged properties set to True.

Example

```
If &REC.IsChanged Then
    Warning("This Record has been changed");
End-if;
```

IsDeleted

Description

This property is True if the record has been deleted. For a level zero record, it is possible for a record to be deleted without the whole row being deleted. For other levels, this property is the same as the row property IsDeleted.

This property is read-only.

Example

```
&tmp = &REC.IsDeleted;
```

See Also

Chapter 32, “Row Class,” IsDeleted, page 1415

IsEditError

Description

This property is True if an error has been found for any field associated with the record after executing the ExecuteEdits method. This property can be used with the Field class properties EditError (to find out which field is in error), and MessageSetNumber and MessageNumber to find the error message set number and error message number.

This property is read-only.

Example

The following is an example showing how IsEditError, along with the ExecuteEdits method could be used:

```
&REC.ExecuteEdits();
If &REC.IsEditError Then
 For &I = 1 to &REC.FieldCount
   If &REC.GetField(&I).EditError Then
      LOG_ERROR();  /* application specific call */
   End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
 End-For;
End-If;
```

See Also

Chapter 31, “Record Class,” ExecuteEdits, page 1386, Chapter 31, “Record Class,” IsEditError, page 1403, Chapter 15, “Field Class,” EditError, page 504, Chapter 15, “Field Class,” MessageNumber, page 512 and Chapter 15, “Field Class,” MessageSetNumber, page 513.

Name

Description

This property returns the name of the record definition of the record as a string.

To access an SQL table name for a record, use the %Table meta-SQL construct.

This property is read-only.

Example

```
WinMessage("The name of this record is : " | &REC.Name);
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “Meta-SQL,” %Table

ParentRow

Description

This property returns the row object for the row containing the record. If the record object was created with the CreateRecord built-in function, the value for ParentRow is null because the record object isn’t part of a row.

This property is read-only.

Example

```
&ROWOBJECT = &REC.ParentRow;
    &TMP = "The row number of the parent row is " | &ROWOBJECT.RowNumber;
/* note that RowNumber is a property of the row class  */
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateRecord

RelLangRecName

Description

This property returns the name of the related language record as a string. The value will be a null string ("") if there is no related language record.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local Record &REC;

&REC = GetRecord();
&RECNAME_BASE = &REC.Name;
&RELLANGRECNAME = &REC.RelLangRecName;
SQLExec("select RELLANGRECNAME from PSRECDEFN where RECNAME = :1", &RECNAME_BASE,⇒
 &RELLANGRECNAME);
```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 32

Row Class

This chapter provides an overview of Row class and discusses the following topics:

- Shortcut considerations.
- Data type of a row object.
- Scope of a row object.
- Row class built-in function.
- Row class methods.
- Row class properties.

Understanding Row Class

A *row* object, instantiated from the Row class, is a single row of data that consists of one to *n* records of data. A single row in a component scroll is a row.

A row object is always associated with a rowset object, that is, you cannot instantiate a row object without first either explicitly or implicitly instantiating a rowset object.

A row may have one to *n* child rowsets. For example, a row in a level two scroll may have *n* level three child rowsets.

CopyTo and GetRecord are two commonly use methods for this class. Visible and IsChanged are two commonly used properties for this class.

The row class is one of the data buffer access classes.

If a rowset object is instantiated using the CreateRowset function, the rowset object that's instantiated is a standalone rowset. Delete and insert activity on these types of rowsets aren't automatically applied at save time. Use standalone rowsets for work records.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Copyright © 1988-2007, Oracle. All rights reserved.

Shortcut Considerations

The default method for the row class is GetRecord. This means you can specify just a record name to access a record on the row. For example, the following two lines of code are equivalent:

```
&Count = &MyRow.EMPL_CHECKLIST.FieldCount;

&Count = &MyRow.GetRecord(RECORD.EMPL_CHECKLIST).FieldCount;
```

In addition, the row class has a method *scrollname*. This enables you to access a specific child rowset *and* row within that rowset. This isn't a default method: you're not accessing a child object, but rather, something within that object. For example, the following two lines of code are equivalent:

```
&ChildRow = &MyRow.EMPL_CHKLST_ITM(&I);

&ChildRow = &MyRow.GetRowset(SCROLL.EMPL_CHKLST_ITM).GetRow(&I);
```

Data Type of a Row Object

Row objects are declared as type Row. For example,

```
Local Row &MYROW;
```

Scope of a Row Object

A Row can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, Component Interface PeopleCode, and so on.

Row Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

Row Class Methods

In this section, we discuss each Row class method. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

CopyTo

Syntax

CopyTo(*row*)

Description

The CopyTo method copies *from*the row executing the method *to*the specified target row, copying *like-named* record fields and subscrolls at corresponding levels. The target row object must be from a "related" rowset, that is, built from the same records, but it can be under a different parent at higher levels.

This method copies *all* the records from the row object executing the method to the target row, not just the primary data record associated with the rowset.

Parameters

Parameter	Description
row	Specify a target row. This must be a row object, not a row number.

Returns

None

Example

The following example copies rows from a child rowset (DERIVED_HR) to the current rows.

```
Local Row &ROW;
Local Rowset &RS1;

&RS1 = GetRowset();
For &I = 1 to &RS1.ActiveRowCount
   &ROW = GetRowset(SCROLL.DERIVED_HR).GetRow(&I);
   &ROW.CopyTo(GetRow(&I));
End-For;
```

See Also

Chapter 32, “Row Class,” ParentRowset, page 1416, Chapter 31, “Record Class,” CopyFieldsTo, page 1381 and Chapter 31, “Record Class,” CopyChangedFieldsTo, page 1380.

GetNextEffRow

Syntax

GetNextEffRow()

Description

The GetNextEffRow method finds the row, in the same rowset as the row executing the method, that has the next effective date (when compared to the row executing the method.)

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None

Returns

A row object. If there is no row with a later effective date, this method returns a null object.

Example

```
&ROW2 = &ROW.GetNextEff();

If &ROW2 <> NULL Then
   &TMP = &ROW2.RowNumber;
   /* other processing */
Else
   /* no effective dated row - do error processing */
End-if;
```

See Also

Chapter 32, "Row Class," GetPriorEffRow, page 1410, Chapter 33, "Rowset Class," GetCurrEffRow, page 1434 and Chapter 33, "Rowset Class," DeleteEnabled, page 1446.

GetPriorEffRow

Syntax

```
GetPriorEffRow()
```

Description

The GetPriorEffRow method finds the row, from the same rowset as the row executing the method, that has the prior effective date to the row executing the method.

Parameters

None

Returns

A row object. If there is no row with a prior effective date, this method returns a null object.

Example

```
&TMP = &ROW.GetPriorEffrow().RowNumber;
```

See Also

Chapter 32, "Row Class," GetNextEffRow, page 1409, Chapter 33, "Rowset Class," GetCurrEffRow, page 1434 and Chapter 33, "Rowset Class," DeleteEnabled, page 1446.

GetRecord

Syntax

```
GetRecord({n | RECORD.recname})
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The GetRecord method creates a record object that references the specified record within the current row object. This is the *default method* for a row object. This means that any row object, followed by a parameter list, acts as if GetRecord is specified.

Parameters

Parameter	Description
n | RECORD.*recname*	Specify a record to be used for instantiating the record object. You can specify either *n* or RECORD.*recname*, Specifying *n* creates a record object for the *nth* record in the row. This might be used if writing code that needs to examine all records in a row without being aware of the record name. Specifying RECORD.*recname* creates a record object for the record *recname*.

Note. There is no way to predict the order the records will be accessed. Use the *n* option only if the order in which the records are processed doesn't matter.

Returns

A record object.

Example

The following example gets a record, then assigns the name of the record to the variable &TMP:

```
&REC = &ROW.GetRecord(RECORD.QEPC_LEVEL1_REC);

&TMP = &REC.NAME;  /* note that NAME is a property of the record class */
```

Because GetRecord is the default method for a row, the following code is identical to the previous:

```
&REC = &ROW.QEPC_LEVEL1_REC;

&TMP = &REC.NAME;
```

The following example loops through all the records for a row:

```
Local rowset &LEVEL1;
Local row &ROW;
Local record &REC;

&LEVEL1 = GetRowset(SCROLL.EMPL_CHECKLIST);
For &I = 1 to &LEVEL1.ActiveRowCount
   &ROW = &LEVEL1.GetRow(&I);
   For &J = 1 to &ROW.RecordCount
      &REC = &ROW.GetRecord(&J);
         /* do processing */
   End-For;
End-For;
```

The following function takes a rowset and a record, passed in from another program. GetRecord won't take a variable for the record, however, using the @ symbol you can convert the string to a component record name.

```
Function Get_My_Row(&PASSED_ROWSET, &PASSED_RECORD)
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   For &ROWSET_ROW = 1 To &PASSED_ROWSET.RowCount
      &UNDERLYINGREC = "RECORD." | &PASSED_ROWSET.DBRecordName;
      &ROW_RECORD = &PASSED_ROWSET.GetRow(&ROWSET_ROW).GetRecord(@&UNDERLYINGREC);

      /* Do other processing */

   End-For;

End-Function;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRecord and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," CreateRecord.

GetRowset

Syntax

GetRowset({*n* | SCROLL.*scrollname*})

Description

The GetRowset method creates a rowset object that references a child rowset of the current row. *scrollname* must specify the primary record for the child rowset.

Parameters

Parameter	Description
n | **SCROLL**.*scrollname*	Specify a rowset to be used for instantiating the rowset object. You can specify either *n* or SCROLL.*scrollname.* Specifying *n* creates a rowset object for the *nth* child rowset in the row. This might be used if you are writing code that examines all rowsets in a row without being aware of the name. Specifying SCROLL.*scrollname* creates a rowset object for the record *scrollname*. *scrollname* must specify the primary record for the child rowset.

Note. There is no way to predict the order the rowsets will be accessed. Use the *n* option only if the order in which the rowsets are processed doesn't matter.

Returns

A rowset object.

Example

```
&CHILDROWSET = &ROW.GetRowset(SCROLL.QEPC_LEVEL1_REC);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 32, "Row Class," scrollname, page 1413

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

scrollname

Syntax
scrollname(*n*)

Description

If *scrollname* is used as a method name for a row, it is used to select a child rowset and a row within that rowset.

```
&ROW.scrollname(n)
```

acts the same as

```
&ROW.GetRowset(SCROLL.scrollname).GetRow(n)
```

Parameters

Parameter	Description
n	Must be a valid row number for the selected child rowset.

Returns

A row object.

Example

```
&SUBROW1 = &ROW.QEPC_LEVEL1_REC(1);
```

See Also
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

Row Class Properties

In this section, we discuss the Row class properties. The properties are discussed in alphabetical order.

ChildCount

Description

This property returns the number of child rowsets of the row. It is defined by the "structure" of the scroll, so it is the same for all rows of the rowset.

It might be used, in conjunction with the GetRowset method, to write code that examines all child rowsets.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
For &I = 1 to &ROW.ChildCount
   &ROWSET = &ROW.GetRowset(&I);
      /* do processing */
End-For;
```

DeleteEnabled

Description

This property determines whether a row can be deleted (the equivalent of the user pressing ALT+8 and ENTER). This property takes a Boolean value.

Note. This property controls only whether an end-user can delete a row. Rows can still be deleted using PeopleCode.

This property is typically used to prevent deletion of an individual row that the other properties controlling deletion would allow to be deleted. The following table goes through the Boolean logic.

Will delete be allowed by user?	Others enable	Others disable
DeleteEnabled = True	Yes	No
DeleteEnabled = False	No	No

The initial value of this property is True.

Note. If No Row Delete is selected in PeopleSoft Application Designer, setting the DeleteEnabled row property to True will *not* override this value. If you want to control whether a row can be deleted at runtime, you should *not* select No Row Delete at design time. Likewise, if the DeleteEnabled Rowset property is False, setting the DeleteEnabled Row property to True will *not* override the rowset property. If you want to control whether an individual row can be deleted at runtime, you should not set the DeleteEnabled rowset property to False.

For consistency, PeopleSoft recommends that either all rows at a level should disable deletions, or they should all allow deletions.

For rows created with non-Component Processor data (such as message rowsets, Application Engine rowsets, and so on) this property has no effect.

Note. Don’t use this property with rows from rowsets created using CreateRowset. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren’t automatically applied at save time.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 33, "Rowset Class," DeleteEnabled, page 1446

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Using Page Controls," Using Scroll Areas and Scroll Bars

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

IsChanged

Description

This property returns True if any field value on the primary database record of the row has been changed.

Note. Use this property only with the primary database record. It doesn't return valid results if used with a work record. This property returns True only if the existing row object has a field that has changed. If a field in a lower level rowset has changed, this property returns False.

This property is read-only.

Considerations Using IsChanged

This property and the IsChanged record property do *not* always return identical values.

If a row in a scroll contains multiple records (such as a primary database record and one or more work records), the IsChanged row property returns True if *any* of the records in the row are changed. The IsChanged record property returns True *only* if a specific record, namely, the primary database record, is changed.

If a row is deleted from a scroll, *only* the primary database record has its IsChanged property marked to False (since the row has been deleted.) Any work records in the row *still* have their IsChanged properties set to True.

Example

```
&tmp =  &ROW.IsChanged;
if &tmp = True then
   Warning("A Field on this row has been changed");
End-If;
```

IsDeleted

Description

This property returns True if the row has been deleted.

This property is read-only.

Note. This property also returns True if the row is new and hasn't been changed.

Example

```
&tmp = &ROW.IsDeleted;
```

Copyright © 1988-2007, Oracle. All rights reserved.

IsEditError

Description

This property is True if an error has been found for any field associated with any record in the current row or on any row in any child rowset of the current row after executing the ExecuteEdits method. This property can be used with the Field class properties EditError (to find out which field is in error), and MessageSetNumber and MessageNumber to find the error message set number and error message number.

This property is read-only.

Example

The following is an example showing how IsEditError, along with the ExecuteEdits method could be used:

```
&MSG.ExecuteEdits();
If &MSG.IsEditError Then
&RS = &MSG.GetRowset();
   For &J = 1 to &RS.RowCount
      &ROW = &RS.GetRow(&J);
         If &ROW.IsEditError Then
            &REC = &ROW.GetRecord();
            For &I = 1 to &REC.FieldCount
               If &REC.GetField(&I).EditError Then
                  LOG_ERROR();  /* application specific call */
               End-If;
            End-For;
      End-If;
   End-For;
End-If;
```

See Also

Chapter 31, “Record Class,” ExecuteEdits, page 1386, Chapter 31, “Record Class,” IsEditError, page 1403, Chapter 15, “Field Class,” EditError, page 504, Chapter 15, “Field Class,” MessageNumber, page 512 and Chapter 15, “Field Class,” MessageSetNumber, page 513.

IsNew

Description

This property is True if the row is a new (inserted) row.

This property is read-only.

Example

```
&tmp = &ROW.IsNew;
```

ParentRowset

Description

This property returns a rowset object referencing the rowset containing the row.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

```
&tmp = &ROW.ParentRowset.RowCount;

/* note that rowcount is a property of the Rowset class */
```

recname

Description

If a record name is used as a property, it accesses the record object with that name. This means the following code:

```
&ROW.somerecname
```

acts the same as

```
&ROW.GetRecord(RECORD.somerecname);
```

This property is read-only.

Example

```
&REC = &ROW.QEPC_LEVEL1_REC;
```

RecordCount

Description

This property returns the number of records in the row. It is defined by the "structure" of the scroll, so it is the same for all rows of the rowset.

This property might be used in conjunction with the GetRecord method to write code that examines all records in some way.

This property is read-only.

Example

```
For &I = 1 to &ROW.RecordCount
   &REC = &ROW.GetRecord(&I);
   /* do processing */
End-For;
```

RowNumber

Description

This property returns the row number (starting from 1) of the row within its rowset.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following returns the row number of the current effective-date row.

```
&NUMBER = GetRowset(SCROLL.MYRECORD).GetCurrEffRow().RowNumber;
```

Selected

Description

This property returns True if the row is part of a grid and has been selected either by the user or programmatically.

This property is meaningful in PIA only if the grid or scroll area attribute has been set to provide the selection control. In this case, the property reflects the state of the checkbox or radio button used for selection.

This property is read-write.

Example

The following example determines how many rows in a grid are selected.

```
Function Count_Child_Assets(&GRIDLEVEL1 As Rowset, &NUM_CHILDREN)
   REM *** How many child assets are selected?                  *;
   &GRIDLEVEL1 = GetRowset(SCROLL.PARENT_CHILD_VW);
   For &CHILDROW_COUNT = 1 To &GRIDLEVEL1.ActiveRowCount
      If &GRIDLEVEL1.GetRow(&CHILDROW_COUNT).Selected = True Then
         &NUM_CHILDREN = &NUM_CHILDREN + 1;
      End-If;
   End-For;
End-Function;
```

Style

Description

This property returns the name of the style class if one has been set for the row. You can also use this property to change the style of a row. This property takes a string value.

Note. This property *overrides* only the existing style. It doesn't *change* it. The next time the page is accessed the original style is used.

The PSSTYLE style sheet does *not* contain a default style class for a row. The Style property for all rows is initially NULL (that is, two quotation marks with no space between them ("")). The row assumes the style of the grid alternate row style. Fields in the row assume field styles if set.

Setting the property for a row overrides the grid alternate row style and any page field styles (field styles that were set in PeopleSoft Application Designer). It does *not* override any field styles set (using the Field class Style property) for fields in the row.

Setting the Style property back to NULL (that is, two quotation marks with no space between them ("")) turns off the override.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. This property is *not* valid for Windows Client applications.

Example

```
&Scroll = GetLevel0().GetRow(1).GetRowset(Scroll.IC_PC_STYLE_2);

&row_class = &Scroll.GetRow(&row);

&row_class.Style = &style;
```

See Also

Chapter 15, "Field Class," Style, page 518

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Creating Field Definitions"

Visible

Description

If this property is True, the row is visible if displayed on the current page. This property can be set False to hide the row.

When Visible is set to False to hide a row, the row moves to the end of the rowset. This means that the row number of the row being hidden, and any subsequent rows, *changes* as a result of hiding a row. If the row is later made visible again, it is *not* moved back to its original position, but remains at the end of the rowset. It is just moved to the start of all the other hidden rows.

For example, if there are 4 rows in a rowset and row 2 is hidden (that is, Visible = False), that row now becomes row 4. In order to make that row visible again, row 4 must be set to Visible = True. Alternatively, you can use a row object reference: this remains valid even though the row number of the row may have been changed.

Note. If you use the Sort method on a rowset with hidden rows, the hidden rows aren't sorted. Only visible rows are sorted.

You cannot hide rows in the current context of the executing program. This means Visible cannot hide the row containing the executing program, or in a child rowset and be executed against its parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Example

```
&ROW.Visible = False;
```

The following code uses the Visible property to determine if a row is visible or not before updating the value of the row.

```
&ROWSET = GetRowset(SCROLL.LD_SHP_INV_VW)

For &I = 1 To &ROWSET.ActiveRowCount
   If &ROWSET(&I).Visible Then
      ITEM_SELECTED.value = "N";
   End-If;
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

The following code starts with the last row and works forward to the first. If any changes are made they don't impact the position of rows that still have to be processed.

```
For &I = &ACTIVE_STREAMS to 1 Step -1
   &Row = &STREAM_ROWSET(&I);
   If &Row.QS_STREAM8.STREAM_ROOT_ID.Value <> &STREAM_ROOT_ID Then
      &Row.Visible = False;
   End-if;
End-For;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 33

Rowset Class

This chapter provides an overview of Rowset class and discusses the following topics:

- Shortcut considerations.
- Rowset Object declaration.
- Scope of a Rowset object.
- Rowset class built-in functions.
- Rowset class methods.
- Rowset class properties.

Understanding Rowset Class

A *rowset* object, instantiated from a Rowset class, is a collection of rows associated with buffer data. A component scroll is a rowset. You can also have a level zero rowset.

If a rowset object is instantiated using GetRowset (either the function or one of the methods) the rowset object that is instantiated is populated with data according to the context in which it was instantiated.

If a rowset object is instantiated using the CreateRowset function, the rowset object that's instantiated is a standalone rowset. Any records and field references created by this function are initialized to null values, that is, they do *not* contain any data. You can populate this rowset object using the CopyTo, Fill, or FillAppend methods.

Default processing isn't performed on a standalone rowset. In addition, a standalone rowset isn't tied to the Component Processor. When you fill it with data, no PeopleCode runs (for example, RowInsert, FieldDefault, and so on.) Delete and insert activity on these types of rowsets aren't automatically applied at save time. Use standalone rowsets for work records.

You might use a rowset object and the ActiveRowCount property to iterate over all the rows of the rowset, or to access a specific row (using the GetRow method) or to find the name of the primary record associated with the scroll (the DBName property).

The Rowset class is one of the data buffer access classes.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Copyright © 1988-2007, Oracle. All rights reserved.

Shortcut Considerations

The default method for the Rowset class is GetRow. This means you can specify just a row number, and not use the GetRow method. For example, the following two lines of code are equivalent:

```
&MyRow = GetRowset()(5);

&MyRow = GetRowset().GetRow(5);
```

Data Type of a Rowset Object

Rowset objects are declared as type Rowset. For example,

```
Local Rowset &MYROWSET;
```

Scope of a Rowset Object

A Rowset can be instantiated from PeopleCode or using Java.

This object can be used anywhere you have PeopleCode, that is, in an Application Class, record field PeopleCode, and so on.

You can't pass a rowset object in as part of a Component Interface user-defined method. (Rowsets aren't common data structures outside of a PeopleSoft system.) However, within a user-defined method for a Component Interface you can use rowset objects.

In an Application Engine program, a rowset object is the equivalent of a record object that contains one row and a single record, that is, the State Record. PeopleSoft suggests using the Record object instead of a rowset object to obtain access to the State Record.

Messages have the same structure as rowsets, that is, hierarchical data structures composed of rows, records, and fields.

File objects can have the same structure as rowsets, that is, hierarchical data structures composed of rows, records, and fields.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Rowset Class Built-In Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

 Copyright © 1988-2007, Oracle. All rights reserved.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowset

Rowset Class Methods

In this section, we discuss the Rowset class methods. The methods are discussed in alphabetical order.

ClearDeletesChanges

Syntax

```
ClearDeletesChanges()
```

Description

Use the ClearDeletesChanges method to clear deleted rows and changed from a standalone rowset.

Note. This method works only with standalone rowsets, that is, rowsets created using the CreateRowset function.

This method differs from Flush in a number of ways:

- it does not remove *all* rows from the rowset, only deleted rows
- it only applies to standalone rowsets

This method first clears the deleted rows, that is, all rows that have been deleted using DeleteRow are removed from the rowset and their associated buffers are freed.

This method then clears changed rows. That means any changes done on a row (such as field values changed) or newly inserted rows are now propagated into their original state and the changed buffers, if any, are freed.

After executing this method on a standalone rowset, any row that was previously new or changed no longer has that state. The IsNew and IsChanged properties of a row return false.

This method does *not* do any database updates.

How would you use this method? Suppose you use a standalone rowset to track changes you need to make to some business process or object. After doing the appropriate database updates to reflect changes recorded in the rowset (that is, inserts or deletes or changes), you call this method to clean up the rowset in preparation for further processing. Without this method, newly inserted rows and changed rows preserve their IsNew and IsChanged status indefinitely, complicating program logic and potentially leading to duplicate inserts or deletes.

Parameters

None.

Returns

None.

Example

```
REM +-------------------------------------------------+;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
REM | Function to Update the DB for a Standalone Rowset|;
REM +-------------------------------------------------+;
Function ProcessDatabaseUpdateforRowset(&rsIn As Rowset)

   For &i = 1 To &rsIn.RowCount

      &rwTMP = &rsIn.GetRow(&i);
      If &rwIn.IsDeleted And
            Not &rwIn.IsNew Then
         &rTMP = &rwIn.GetRecord(1);
         &rTMP.Delete();
      End-If;

      If &rwTMP.IsNew And
            &rwTMP.IsChanged And
            Not &rwTMP.IsDeleted Then

         &rTMP = &rwTMP.GetRecord(1);
         &rTMP.Insert()
      End-If;

      If Not &rwTMP.IsNew And
            &rwTMP.IsChanged And
            Not &rwTMP.IsDeleted Then

         &rTMP = &rwTMP.GetRecord(1);
         &rTMP.Update();
      End-If;

   End-For;

   REM +-------------------------------------------------+;
   REM | Now we need to reset the Rowset flags and     |;
   REM | remove deleted rows                           |;
   REM +-------------------------------------------------+;
   &rsIn.ClearDeletesChanges();

End-Function;
```

See Also

Chapter 33, “Rowset Class,” Flush, page 1432, Chapter 33, “Rowset Class,” FlushRow, page 1433, Chapter 32, “Row Class,” IsChanged, page 1415 and Chapter 32, “Row Class,” IsNew, page 1416.

CopyTo

Syntax

CopyTo(*&DestRowset* [, *record_list*])

Where *record_list* is a list of record names in the form:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
[RECORD.source_recname1, RECORD.target_recname1
[, RECORD.source_recname2, RECORD.target_recname2]].  .  .
```

Description

The CopyTo method copies *from* the rowset executing the method *to* the specified destination rowset, copying *like-named* record fields and subscrolls at corresponding levels.

The CopyTo method uses the current data in the rowset. This might be different from the original data values if the rowset was retrieved from the database and values in it have been changed either by an end-user or a PeopleCode program

If pairs of source and destination record names are given, these are used to pair up the records and subscrolls *before* checking for like-named record fields. Then, after copying the named records pairs, this method copies *all* identically named records.

Note. This method does not work for Application Engine state records. If you don't specify *record_list*, *both* the record name *and* the field name have to match exactly for data to be copied from one record field to another. If you specify *record_list*, after the records have been paired up, the field names have to match before any data is copied.

If the rowset you are copying *from* has the field level EditError, as well as the MessageNumber and MessageSetNumber properties set, these values are copied to the rowset you are copying to. For example, suppose you had an application message that had errors. Using the GetSubContractInstance function, these errors would be copied into the message object. From there, you could instantiate a message rowset, copy the message rowset into a work rowset, and use the work rowset to populate Component buffers. (After the field properties are set, the Record, Row, and Rowset properties IsEditError also get set.)

Parameters

Parameter	Description
&DestRowset	Specify the rowset to be copied to. This rowset object must have already been instantiated.
SourceRecname	Specify a record to be copied from, in the rowset object being copied from.
DestRecname	Specify a record to be copied to, in the rowset object to be copied to.

Returns

None.

Example

If you set one rowset equal to another, you haven't made a copy of the rowset. Instead, you have two variables pointing to the *same* data.

To make a clone of an existing rowset, that is, to make two distinct copies, you can do the following:

```
&RS2 = CreateRowset(&RS);
&RS.CopyTo(&RS2);
```

Copyright © 1988-2007, Oracle. All rights reserved.

The following example copies data from one rowset object to another. Because no like-named records exist between the two rowsets, the record names are specified. Only the like-named fields are copied from one rowset to the other:

```
Local Rowset &RS1, &RS2;
Local String &EMPLID;

&RS1 = CreateRowset(RECORD.PERSONAL_DATA);
&RS2 = CreateRowset(RECORD.PER_VENDOR_DATA);
&EMPLID = "8001";
&RS1.Fill("WHERE EMPLID =: 1", &EMPLID);
&RS1.CopyTo(&RS2, RECORD.PERSONAL_DATA, RECORD.PER_VENDOR_DATA);
```

The following example copies data from a message into the Component buffers, then calls the page (using TransferPage) to redraw the page. You could do this to fill a page with message data that is in error, so that an end-user can make corrections to the message data. &WRK_ROWSET0 is the level zero rowset and &WRK_ROWSET1 is where the data is copied to.

```
/* Get the Message */
&MSG = GetSubContractInstance(&PUBID, &PUBNODE, &CHNLNAME, &MSGNAME, &SUBNAME);

/* Get the Message Rowset */
&MSG_ROWSET = &MSG.GetRowset();

/* Get Level 0 */
&WRK_ROWSET0 = GetLevel0();

/* Create Work rowset */
&WRK_ROWSET1 = GetLevel0()(1).GetRowset(SCROLL.EN_REVISION_TMP);

/* Populate Work Rowset */
&MSG_ROWSET.CopyTo(&WRK_ROWSET1, RECORD.EN_REVISION, RECORD.EN_REVISION_TMP);

SetNextPage("EN_REVISION_MSG");

TransferPage();
```

See Also

Chapter 32, “Row Class,” CopyTo, page 1409 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, “PeopleCode Built-in Functions,” CreateRowset.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer’s Guide, “Understanding Objects and Classes in PeopleCode,” Assigning Objects

DeleteRow

Syntax

DeleteRow(*n*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The DeleteRow method deletes the row in the rowset identified by the parameter.

If the program is being run from a component against Component buffer data, a RowDelete PeopleCode event also fires, followed by the events that normally follow a RowDelete, as if the user had manually pressed ALT+8 and ENTER.

This method initially marks the row as needing to be deleted. At save time the row is actually deleted from the database and cleared from the buffer. When the row is marked as deleted, it is ignored by other methods, such as GetCurrEffRow, Sort, and so on.

DeleteRow cannot be executed from the same rowset where the deletion takes place, or from a child rowset against a parent. Place your PeopleCode in a parent rowset and execute it against a child rowset.

When DeleteRow is used in a loop, you have to process rows from high to low to achieve the correct results, that is, you must delete from the bottom up rather than from the top down. This is necessary because the rows are renumbered after they are deleted (if you delete row one, row two becomes row one).

Note. If you use DeleteRow on a rowset created using the CreateRowset function, the row isn't automatically deleted in the database when the page is saved. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

Parameters

Parameter	Description
n	An integer identifying a row within the rowset object. This must be >= 1 and <= the number of active rows in the rowset (see ActiveRowCount).

Returns

An optional Boolean value: True if row is deleted, False otherwise.

Example

In the following example DeleteRow is used in a For loop. The example checks a value in each row, then conditionally deletes the row. Note the syntax of the For loop, including the use of the -1 in the Step clause to loop from the highest to lowest values. This ensures that the renumbering of the rows do not affect the loop.

```
For &I = &RS2.ActiveRowCount To 1 Step -1
   If None(&CHECK_SEQ) Then
      &RS2.DeleteRow(&I);
   End-If;
End-For;
```

See Also

Chapter 33, "Rowset Class," Flush, page 1432, Chapter 33, "Rowset Class," FlushRow, page 1433, Chapter 33, "Rowset Class," InsertRow, page 1437 and Chapter 31, "Record Class," Insert, page 1390.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Copyright © 1988-2007, Oracle. All rights reserved.

Fill

Syntax

```
Fill([wherestring [, bindvalue] . . .])
```

Description

The Fill method flushes the rowset then reads records from the database into successive rows. The records are read from the database tables corresponding to the primary database record of the scroll into that record. The records are selected by the optional *wherestring* SQL clause, in which the optional *bindvalues* are substituted, using the usual bind placeholders *(:n)*.

In general, use this method only with rowsets that were created using the CreateRowset function.

Note. Because Flush always leaves one row in the scroll, there will be one row in the scroll even if you don't read any records.

The actual number of records read into the rowset is an optional return of this method.

Note. This method does not work with Application Engine state records. Also, you cannot use this method in dynamic views.

When this method executes, unlike the Select method, it does *not* cause any associated PeopleCode to run as part of reading data into the rowset.

Note. Fill reads only the primary database record. It does not read any related records, nor any subordinate rowset records.

For every record read with the Fill method, if the set language is not the base language and the record has related language records, the Fill method tries to read the related language record and does related language processing.

The Fill method uses a correlation ID of FILL for the table it reads. You must use the correlation ID if you want to refer to the rowset table name as part of the *wherestring*. You receive a runtime error if you use the table name as a column prefix instead of the correlation ID.

Sorting Considerations

Rows come unsorted from the database when using Fill. This is not a problem for SQL server, however, it can be a problem for DB2 UDB for OS/390 and z/OS and Oracle.

See Chapter 33, "Rowset Class," Sort, page 1443.

Parameters

Parameter	Description
wherestring	Specify a SQL WHERE clause to use for selecting records to fill the rowset. This can be a string or a SQL definition.
bindvalue	Specify optional bind variables to be used with the WHERE clause.

Returns

The number of records read into the rowset.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example reads all of the QA_MYRECORD records into a rowset, and returns the number of rows read:

```
&RS = CreateRowset(RECORD.QA_MYRECORD);
&NUM_READ = &RS.Fill();
```

The following example reads all of the QA_MYRECORD records that have a MYRECORD field equal to the value of &UVAL into a rowset, and returns the number of rows read:

```
&NUM_READ = &RS.Fill("where MYRECORD = :1", &UVAL);
```

To re-use a WHERE clause for the *wherestring* you can use the SQL repository, and a SQL object.

```
&NUM_READ = &RS.Fill(SQL.MYWHERE, &UVAL);
```

The following gets all the SET_CNTRL_REC rows related to the row on the page, then updates SETID with the value from the page. Fill is used with a rowset that was created from a message that was just created, that is, a rowset that was unpopulated.

```
If FieldChanged(SETID) Then
   &MSG = CreateMessage(OPERATION.SET_CNTRL_REC);
   &MSG_ROWSET = &MSG.GetRowset();
   &MSG_ROWSET.Fill("where SETCNTRLVALUE =:1 and REC_GROUP_ID =:2", SETCNTRLVALUE,⇒
 REC_GROUP_ID);

For &I = 1 To &MSG_ROWSET.ActiveRowCount
   &MSG_ROWSET.GetRow(&I).SET_CNTRL_REC.SETID.Value = SETID;
   &MSG_ROWSET.GetRow(&I).PSCAMA.AUDIT_ACTN.Value = "C";
End-For;

&MSG.Publish();

End-If;
```

The following example uses a correlation ID for the table in the Fill SELECT, to enable the use of correlated subqueries in the WHERE clause, such as the usual effective-date subquery:

```
&RSOWNER_NAME = CreateRowset(RECORD.PERSONAL_D00);
&RSOWNER_NAME.Fill("where SETID=:1 AND EMPLID=:2 AND %EffDtCheck(PERSONAL_⇒
D00,FILL,:3)", &SETID, &EMPLID, &EFFDT);
```

The Fill method implicitly uses Fill as an alias for the Rowset record. This is helpful for complex Fill *where* clauses with subqueries.

```
&oprclasscountries = CreateRowset(Record.SCRTY_TBL_GBL);

&oprclasscountries.Fill("WHERE FILL.OPRCLASS = :1 AND NOT EXISTS (SELECT 'X' FROM⇒
 PS_SCRTY_SEC_GBL GBL2 WHERE GBL2.OPRCLASS = FILL.OPRCLASS AND GBL2.COUNTRY =⇒
 FILL.COUNTRY AND GBL2.PNLNAME = :2)", &OPRCLASS, %Component);
```

In the following example, the necessary key field values are loaded into a rowset, then the following function is called, and the values are used as part of the Fill method.

```
Function FillRS2();
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   Local SQL &MySql;
   Local string &MySqlString;
   Local Record &ElemDefnRec;
   Local string &ElemDefnRecName, &fldname;
   Local Rowset &CompRec2RS;

   /* Build the record object that is used for building the SQL and executing the⇒
 select */
   &ElemDefnRec = CreateRecord(@("Record." | &pkgRecName));

   /* Initialize the SQL string */
   &MySqlString = "%SelectByKey(:1 A)";

   /* Create a SQL to select a rows based on the key fields. */
   For &i = 1 To &ElemDefnRec.FieldCount
      If &ElemDefnRec.GetField(&i).IsKey Then
         &fldname = &ElemDefnRec.GetField(&i).Name;
         &ElemDefnRec.GetField(&i).Value = &CompRec2RS.GetRow(1).GetRecord(@⇒
("Record." | &pkgRecName)).GetField(@("Field." | &fldname)).Value;
      End-If;
   End-For;

   /* Create the SQL and execute the select */
   &MySql = CreateSQL(&MySqlString);
   &MySql.Execute(&ElemDefnRec);

   /* Copy each selected row into the rowset */
   While &MySql.Fetch(&ElemDefnRec)
      &ElemDefnRec.CopyFieldsTo(&CompRec2RS(&CompRec2RS.ActiveRowCount).GetRecord⇒
(1));
   End-While;

End-Function;
```

See Also

Chapter 33, "Rowset Class," CopyTo, page 1424, Chapter 33, "Rowset Class," Select, page 1439 and Chapter 33, "Rowset Class," FillAppend, page 1430.

Chapter 38, "SQL Class," page 1563

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

Enterprise PeopleTools 8.49 PeopleBook: Global Technology, "Using Related Language Tables"

FillAppend

Syntax

FillAppend([*wherestring* [, *bindvalue*] . . .])

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The FillAppend method reads records from the database into successive rows of the rowset starting *after* the last row that exists in the rowset. Unlike Fill, it doesn't first flush the rowset.

Note. FillAppend appends rows after the last *active* row in the rowset. If you have deleted rows in the rowset, they will still be the last rows.

When a rowset is selected into, any autoselected child rowsets are also read. The child rowsets are read using a where clause that filters the rows according to the where clause used for the parent rowset, using a subselect.

When a rowset is created using CreateRowset, it contains one empty row. If the rowset hasn't been filled with data, FillAppend fills that row also (so you don't have an empty row at the start of your rowset.)

The records are read from the database tables corresponding to the primary database record of the scroll into that record. The records are selected by the optional *wherestring* SQL clause, in which the optional *bindvalues* are substituted, using the usual bind placeholders (*:n*).

In general, use this method only with rowsets that were created using the CreateRowset function.

The actual number of records read into the rowset is an optional return of this method.

Note. This method does not work with Application Engine state records. Also, you cannot use this method in dynamic views.

When this method executes, unlike the Select method, it does *not* cause any associated PeopleCode to run as part of reading data into the rowset.

Note. FillAppend reads only the primary database record. It does not read any related records, nor any subordinate rowset records.

For every record read with the FillAppend method, if the set language is not the base language and the record has related language records, the FillAppend method tries to read the related language record and does related language processing.

The FillAppend method uses a correlation ID of FILLAPPEND for the table it reads. You must use the correlation ID if you want to refer to the rowset table name as part of the *wherestring*. You receive a runtime error if you use the table name as a column prefix instead of the correlation ID.

Parameters

Parameter	Description
wherestring	Specify a SQL WHERE clause to use for selecting records to fill the rowset. This can be a string or a SQL definition.
bindvalue	Specify optional bind variables to be used with the WHERE clause.

Returns

The number of records read into the rowset.

Example

The following example reads all of the QA_MYRECORD records that have a MYRECORD field equal to the value of &UVAL into a rowset, and returns the number of rows read:

Copyright © 1988-2007, Oracle. All rights reserved.

```
&NUM_READ = &RS.FillAppend("where MYRECORD = :1", &UVAL);
```

To re-use a WHERE clause for the *wherestring* you can use the SQL repository, and a SQL object.

```
&NUM_READ = &RS.FillAppend(SQL.MYWHERE, &UVAL);
```

See Also

Chapter 33, "Rowset Class," CopyTo, page 1424, Chapter 33, "Rowset Class," Select, page 1439 and Chapter 33, "Rowset Class," Fill, page 1428.

Chapter 38, "SQL Class," page 1563

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

Enterprise PeopleTools 8.49 PeopleBook: Global Technology, "Using Related Language Tables"

Flush

Syntax

Flush()

Description

Use the Flush method to remove all rows from the rowset and free its associated buffer. Rows that are flushed are not deleted from the database. This function is often used to clear a work scroll before using the Select method.

Note. Flush always leaves one row in the scroll.

You cannot flush the current context of the executing program. This means Flush cannot be used for the rowset containing the executing program, or in any child rowset and executed against the parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Considerations Initializing New Rows

Flush removes all rows and inserts a row.

If you want to initialize the row, that is, have the defaults and any RowInit PeopleCode run, you must do something to invoke the default value processing. This can be as simple as setting the value of another field on the same page that has a PeopleCode program associated with it.

If the rowset is created from message data, an Application Engine program, and so on, the rows are flushed but the row that is inserted is not initialized.

Parameters

None.

Returns

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example checks for the value of the field CHECKLIST_CD. If something exists in this field, the second level rowset is flushed and new values are selected into it.

```
If All(CHECKLIST_CD) Then
   &RS1H.Flush();
   &RS1H.Select(RECORD.CHECKLIST_ITEM, "where Checklist_CD = :1 and EffDt = ⇒
(Select Max(EffDt) from PS_CHECKLIST_ITEM Where CheckList_CD = :2)", CHECKLIST_CD,⇒
 CHECKLIST_CD);
End-If;
```

See Also

Chapter 33, "Rowset Class," FlushRow, page 1433, Chapter 33, "Rowset Class," DeleteRow, page 1426 and Chapter 33, "Rowset Class," InsertRow, page 1437.

FlushRow

Syntax

FlushRow(*n*)

Description

Use the FlushRow method to remove a specific row from a rowset and from the Component buffer. Rows that are flushed are not deleted from the database.

FlushRow is a specialized and infrequently used method. In most situations, you want to use DeleteRow to remove a row from the Component buffer and remove it from the database as well when the component is saved.

You cannot flush the current context of the executing program. This means FlushRow cannot be used for the rowset containing the executing program, or in any child rowset and executed against the parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Parameters

Parameter	Description
n	An integer identifying a row within the rowset object. This must be >= 1 and <= the number of rows in the rowset (see property RowCount).

Returns

None.

Example

```
&ROWSET.FlushRow(&ROWNUMBER);
```

See Also

Chapter 33, "Rowset Class," DeleteRow, page 1426, Chapter 33, "Rowset Class," Flush, page 1432 and Chapter 33, "Rowset Class," InsertRow, page 1437.

Copyright © 1988-2007, Oracle. All rights reserved.

GetCurrEffRow

Syntax

GetCurrEffRow()

Description

GetCurrEffRow returns a row object for the row with the current effective date. This includes dates equal to the current date, but not dates in the future. If the primary record associated with the rowset is not effective-dated this method returns a null object.

Newly inserted rows are intentionally skipped when looking for the current effective-dated row. This is to find the current effective row from the data that already exists in the database. In other words, a row cannot be considered as a current effective row until it is saved. To find the current effective-dated row in data that has been newly inserted and not yet saved, use the GetNewEffRow method instead.

Parameters

None.

Returns

A row object for the row with the current effective date.

Example

```
&tmp =  &ROWSET.GetCurrEffRow().RowNumber;
/* note RowNumber is a property of the row class */
```

See Also

Chapter 33, “Rowset Class,” DeleteEnabled, page 1446, Chapter 33, “Rowset Class,” GetRow, page 1435, Chapter 32, “Row Class,” GetNextEffRow, page 1409 and Chapter 33, “Rowset Class,” GetNewEffRow, page 1434.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetRow

GetNewEffRow

Syntax

GetNewEffRow()

Description

GetNewEffRow returns a row object for the row with the effective date closest to the current date, including newly inserted rows. This means dates equal to the current date, but not future dates. If the primary record associated with the rowset is not effective-dated this method returns a null object.

To find the current effective-dated row from the data that already exists in the database, that is, only from saved rows and *not* a newly created row, use the GetCurEffRow method instead.

Parameters

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A row object for the row with the current effective date.

Example

```
Local Row &MyRow = &ROWSET.GetNewEffRow();
```

See Also

Chapter 33, "Rowset Class," GetRow, page 1435, Chapter 32, "Row Class," GetNextEffRow, page 1409 and Chapter 33, "Rowset Class," GetCurrEffRow, page 1434.

Chapter 33, "Rowset Class," DeleteEnabled, page 1446

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

GetRow

Syntax

GetRow(*n*)

Description

GetRow returns a row object from a rowset. This is a default method for rowsets. This means that any rowset object, followed by a parameter list, acts as if GetRow is specified.

Parameters

Parameter	Description
n	An integer identifying a row within the rowset object. This must be >= 1 and <= the number of rows in the rowset (see property RowCount).

Returns

Returns a row object for the specified row of the rowset.

Example

In the following example, all of the lines of code do the same thing: they return the 5th row from the rowset (&ROWSET is a rowset object.)

```
&ROW = GetRowset().GetRow(5);
&ROW = GetRowset()(5);
&ROW = &ROWSET.GetRow(5);
&ROW = &ROWSET(5);
```

The following example loops through all the rows in a rowset:

```
Local rowset &LEVEL1;
Local row &ROW;

&LEVEL1 = GetRowset(SCROLL.EMPL_CHECKLIST);
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &ROW = &LEVEL.GetRow(&I);
   /* do some processing */
End-For;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRow

HideAllRows

Syntax

HideAllRows()

Description

HideAllRows hides all rows of the rowset. Using this method is equivalent to a loop setting the visible property of each row of the rowset to False.

You cannot hide rows in the current context of the executing program. This means HideAllRows cannot hide the rowset containing the executing program, or in any child rowsets and be executed against the parent rowset. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Parameters

None.

Returns

None.

Example

The following example hides the second level scroll if a value exists in the NO_COMMENTS field in the first level of the scroll. The code is running from the first level of the scroll.

```
Local Rowset &RS1, &RS2;

&RS1 = GetRowset();
&RS2 = GetRowset(SCROLL.EMPL_CHKLST_ITM);

For &I = 1 to &RS1.ActiveRowCount
   If ALL(&RS1.GetRow(&I).EMPLOYEE_CHECKLIST.NO_COMMENTS) Then
      &RS2.HideAllRows();
   End-If;
/*other processing */
End-For;
```

See Also

Chapter 33, "Rowset Class," ShowAllRows, page 1443 and Chapter 32, "Row Class," Visible, page 1419.

 Copyright © 1988-2007, Oracle. All rights reserved.

InsertRow

Syntax

```
InsertRow(n)
```

Description

InsertRow programmatically performs the ALT+7 and ENTER (RowInsert) function. InsertRow inserts a new row in the current rowset *after* the row specified by the parameter if the primary record for the rowset is not effective-dated or a standalone rowset. If the primary record for the rowset is effective-dated or a standalone rowset, the new row is inserted *before* the current row, and all values from the current row are copied into the new row, except for EFFDT, which is set to the current date.

In addition, InsertRow propagates the keys from the higher level (if any) into the inserted row.

If the program is being run from a component against Component buffer data, a RowInit PeopleCode event also fires, followed by the events that normally follow a RowInsert, as if the user had manually pressed ALT+7 and ENTER.

The InsertRow method can be executed against the same rowset where the insertion will take place.

For rowsets corresponding to Component buffer data, the InsertRow method only executes in turbo mode: that is, default processing is performed, but only on the row being inserted.

InsertRow cannot be executed from the same rowset where the insertion will take place, or from a child rowset against a parent. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Note. If you use InsertRow on a rowset created using the CreateRowset function, the row isn't automatically inserted in the database when the page is saved. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

Effective-Dated Row Considerations

When a row is inserted, if that row contains child scrolls, this method also inserts an empty row for any child scrolls. The effective-date field for this empty row is also empty. The current date is *not* used.

Parameters

Parameter	Description
n	An integer identifying a row within the rowset object. This must be >= 0 and <= the number of active rows in the rowset (see property ActiveRowCount). A value of 0 inserts in front of the first row.

Returns

An optional Boolean value: True if the row is inserted, False if the row is not inserted.

Example

In the following example, as the primary database record isn't effective-dated, the new row is inserted *after* the second row in the rowset.

```
&ROWSET.InsertRow(2);
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 33, "Rowset Class," DeleteRow, page 1426 and Chapter 31, "Record Class," Delete, page 1385.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

Refresh

Syntax

```
Refresh()
```

Description

Refresh reloads the rowset object from the database using the current page keys. This causes the page to be redrawn. The following code example refreshes the entire page:

```
GetLevel0().Refresh()
```

If you only want a specific scroll to be redrawn, you can use the refresh method with that rowset. You don't have to use a level zero rowset with this method. This is particularly useful after using RemoteCall.

Note. If a scroll is marked as No Auto Select in PeopleSoft Application Designer, Refresh will not work on it. The Refresh method clears the existing buffers and does a refresh from the database. You do *not* want to use this method if your rowset is based on message data. Instead, you can use the CopyTo Rowset method combined with TransferPage.

Parameters

None.

Returns

None.

Example

The following example refreshes the entire page.

```
&Mylevel0 = GetLevel0().Refresh();
```

The following example refreshes only the second level scroll of the page:

```
&RowSetLevel2.Refresh();
```

See Also

Chapter 33, "Rowset Class," CopyTo, page 1424

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," RemoteCall

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransferPage

 Copyright © 1988-2007, Oracle. All rights reserved.

Select

Syntax

```
Select([parmlist], RECORD.selrecord [, wherestr, bindvars])
```

Where *paramlist* is a list of child rowsets, given in the following form:

```
SCROLL.scrollname1 [, SCROLL.scrollname2] . . .
```

The first *scrollname* must be a child rowset of the rowset object executing the method, the second *scrollname* must be a child of the first child, and so on.

Description

Select, like the related method SelectNew, reads data from the database tables or views into either a row or rowset object.

Note. This method is valid only when used with rowsets that reference data from the component buffers. You cannot use this method with a message rowset, a rowset from an Application Engine state record, and so on. You can't use this method with rowsets created using the CreateRowset function (use the Fill method instead.)

Select automatically places child rowsets in the rowset object executing the method under the correct parent row. If it cannot match a child rowset to a parent row, an error occurs.

When a rowset is selected into, any autoselected child rowsets is also read. The child rowsets are read using a where clause that filters the rows according to the where clause used for the parent rowset, using a subselect.

If you don't specify any child rowsets in *paramlist*, Select selects from a SQL table or view specified by *selrecord* into the rowset object executing the method.

If you specify a child rowset in *paramlist*, Select selects from a SQL table or view specified by *selrecord* into the child rowset specified in *paramlist*, under the appropriate row of the rowset executing the method.

If the rowset executing the method is a level zero rowset, and you specify Select without specifying any child rowsets with *paramlist*, the method reads only a single row, because only one row is allowed at level zero.

The rowset executing the method *does not* have to be a level zero rowset, so the Select method can produce the functionality of both the ScrollSelect and RowScrollSelect functions.

Note. For developers familiar with previous releases of PeopleCode: If the rowset executing the method is a level zero rowset, and you specify a child rowset with *paramlist*, this method functions exactly like ScrollSelect. If the rowset executing the method is not a level zero rowset, and no child rowsets are specified with *paramlist*, the method acts like RowScrollSelect.

The record definition of the table or view being selected from is called the *select record*, and identified with **RECORD.** *selrecord*. The select record can be the same as the primary database record associated with the rowset, or it can be a different record definition that has compatible fields. If you define a select record that differs from the primary database record for the rowset, you can restrict the number of fields that are loaded into the buffers on the client workstation by including only the fields that you actually need.

Select accepts a SQL string that can contain a WHERE clause restricting the number of rows selected into the object. The SQL string can also contain an ORDER BY clause to sort the rows.

Select and its related methods generate a SQL SELECT statement at runtime, based on the fields in the select record and the WHERE clause passed to them in the method parameters.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
paramlist	An optional parameter list, for specifying children rowsets of the rowset executing the method, as the rowset to be selected into. The first *scrollname* in *paramlist* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.
RECORD.*selrecord*	Specifies the select record. The *selrecord* record must be defined in PeopleSoft Application Designer and be a build SQL table (using Build, Project) as a table or a view, unless *selrecord* is the same record as the primary database record associated with the rowset. *selrecord* can contain fewer fields than the primary record associated with the rowset, although it must contain any key fields to maintain dependencies with other records.
wherestr	Contains a WHERE clause to restrict the rows selected from *selrecord* and/or an ORDER BY clause to sort the rows. The WHERE clause can contain the PeopleSoft meta-SQL functions such as %DateIn() or %CurrentDateIn. It can also contain inline bind variables.
bindvars	A list of bind variables to be substituted in the WHERE clause. The same restrictions that exist for SQLExec exist for these variables. See SQLExec for further information.

Returns

The number of rows read (optional.) This counts only the lines read into the specified rowset. It does not include any additional rows read into any child rowsets of the rowset.

Example

The following example selects into the level zero rowset, only one row. Depending on the autoselect settings on any child scrolls, they may be reselected too.

```
&LEVEL0 = GetLevel0();
&LEVEL0.Select(RECORD.PERSONAL_DATA, "WHERE.  .  .");
```

The following example selects into the level one scroll specified. Depending on the autoselect settings on any child (level two) scrolls, they may also be selected.

```
&LEVEL0.Select(SCROLL.EMPL_CHECKLIST, RECORD.EMPL_CHECKLIST, "WHERE.  .  .");
```

The following example selects into the child scroll of the level one rowset. Each row fetched is placed under the appropriate row in &LEVEL1. Note that instead of hard-coding the WHERE clause, the SQL repository is used to access a SQL definition named SELECT_WHERE.

```
&LEVEL1 = &LEVEL0()(1).GetRowset(SCROLL.EMPL_CHECKLIST);
&LEVEL1.Select(SCROLL.EMPL_CHKLST_ITM, RECORD.EMPL_CHKLST_ITM, SQL.SELECT_WHERE);
```

The following example first flushes the hidden work scroll, then selects into it based on a field on the page.

```
&RS1H.Flush();
&RS1H.Select(RECORD.CHECKLIST_ITEM, "where Checklist_CD = :1 and EffDt = (Select⇒
 Max(EffDt) from PS_CHECKLIST_ITEM Where CheckList_CD = :2)", CHECKLIST_CD,⇒
 CHECKLIST_CD);
```

See Also

Chapter 33, "Rowset Class," SelectNew, page 1441 and Chapter 31, "Record Class," SelectByKey, page 1394.

 Copyright © 1988-2007, Oracle. All rights reserved.

SelectNew

Syntax

```
SelectNew([parmlist], RECORD.selrecord [, wherestr, bindvars])
```

Where *paramlist* is a list of child rowsets, given in the following form:

```
SCROLL.scrollname1 [, SCROLL.scrollname2] . . .
```

The first *scrollname* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.

Description

The SelectNew method is similar to the Select method, except that all rows read into the rowset are marked *new* so they are automatically inserted into the database at Save time. This capability can be used, for example, to insert new rows into the database by selecting data using a view of columns from other database tables.

Note. This method is valid only when used with rowsets that reference data from the Component buffers. You cannot use this method with a message rowset, a rowset from an Application Engine state record, and so on. You can't use this method with rowsets created using the CreateRowset function (use the Fill method instead.)

Parameters

Parameter	Description
paramlist	An optional parameter list, for specifying children rowsets of the rowset executing the method, as the rowset to be selected into. The first *scrollname* in *paramlist* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.
RECORD.*selrecord*	Specifies the select record. The *selrecord* record must be defined in PeopleSoft Application Designer and be a build SQL table (using Build, Project) as a table or a view, unless *selrecord* is the same record as the primary database record associated with the rowset. *selrecord* can contain fewer fields than the primary record associated with the rowset, although it must contain any key fields to maintain dependencies with other records.
wherestr	Contains a WHERE clause to restrict the rows selected from *selrecord* and/or an ORDER BY clause to sort the rows. The WHERE clause can contain the PeopleSoft meta-SQL functions such as %DateIn() or %CurrentDateIn. It can also contain inline bind variables.
bindvars	A list of bind variables to be substituted in the WHERE clause. The same restrictions that exist for SQLExec exist for these variables. See SQLExec for further information.

Returns

The number of rows read (optional.) This counts only the lines read into the specified rowset. It does not include any additional rows read into any child rowsets of the rowset.

Example

The following example selects rows from DATA2 and reads them into a level one scroll. If the user saves the page, these rows are inserted into DATA1.

```
&LEVEL0.SelectNew(SCROLL.DATA1, RECORD.DATA2, "Where SETID = :1 and CUST_ID =:2",⇒
 CUSTOMER.SETID, CUSTOMER.CUST_ID);
```

Copyright © 1988-2007, Oracle. All rights reserved.

If you use the same WHERE clause in more than one Select, you may want to use the SQL repository to store the SQL fragment. That way, if you ever want to change it, you will have to change it in only one place.

```
&LEVEL0.SelectNew(SCROLL.DATA1, RECORD.DATA2, SQL.Select_New);
```

See Also

Chapter 33, "Rowset Class," Select, page 1439 and Chapter 31, "Record Class," SelectByKey, page 1394.

SetDefault

Syntax

SetDefault(*recname.fieldname*)

Description

The SetDefault method sets the value of the specified *recname.fieldname* to a null value in every row of the rowset. If this method is being run from a component against Component buffer data, the next time default processing is performed, this value is reinitialized to its default value: either a default specified in its record field definition or one set programmatically by PeopleCode located in a FieldDefault event. If neither of these defaults exist, the Component Processor leaves the field blank.

No default processing is performed against data that is not in the Component buffers.

Blank numbers correspond to zero on the database. Blank characters correspond to a space on the database. Blank dates and long characters correspond to NULL on the database. SetDefault gives each field data type its proper value.

Parameters

Parameter	Description
recname.fieldname	Specifies a field. The field must be in the specified record, on the rows of the rowset.

Returns

None.

Example

This example resets the PROVIDER to its null value. This field is reset to its default value when default processing is next performed:

```
If COVERAGE_ELECT = "W" Then
   &ROWSET.SetDefault(INS_NAME_TBL.PROVIDER);
End-if;
```

See Also

Chapter 15, "Field Class," SetDefault, page 499

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "PeopleCode and the Component Processor," Default Processing

 Copyright © 1988-2007, Oracle. All rights reserved.

ShowAllRows

Syntax

```
ShowAllRows()
```

Description

ShowAllRows "unhides" all rows of the rowset object executing the method. Using this method is equivalent to a loop setting the visible property of each row of the rowset to True.

ShowAllRows cannot be executed from the same rowset you want to display, or from a child rowset against a parent. Place your PeopleCode in a parent rowset and execute it against a child rowset.

Parameters

None

Returns

None.

Example

```
&ROWSET.ShowAllRows();
```

See Also

Chapter 33, "Rowset Class," HideAllRows, page 1436 and Chapter 32, "Row Class," Visible, page 1419.

Sort

Syntax

```
Sort([paramlist,] sort_fields)
```

Where *paramlist* is a list of child rowsets, given in the following form:

```
SCROLL.scrollname1 [, SCROLL.scrollname2] . . .
```

The first *scrollname* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.

And where *sort_fields* is a list of field specifiers in the form:

```
[recordname.]field_1, order_1 [, [recordname.]field_2, order_2]_
```

Description

Sort programmatically sorts the rows in a rowset. The rows can be sorted on one or more fields.

If you specify a child rowset in *paramlist*, Sort selects a set of child rowsets to be sorted. If you don't specify a child rowset in *paramlist*, the rowset executing the method is sorted.

The type of sort done by this function, that is, whether it is a linguistic or binary sort, is determined by the Sort Order Option on the PeopleTools Options page.

Note. The Sort method sorts only *visible* rows in a rowset. If the rowset you're sorting has hidden rows, the hidden rows are not sorted. Rows marked for deletion are also not sorted.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
paramlist	An optional parameter list, for specifying children rowsets of the rowset executing the method, as the rowset to be sorted. The first *scrollname* in *paramlist* must be a child rowset of the rowset executing the method, the second *scrollname* must be a child of the first child, and so on.
sort_fields	A list of field and order specifiers which act as sort keys. The rows in the rowset will be sorted by the first field in either ascending or descending order, then by the second field in either ascending or descending order, and so on.
[*recordname*.]*field_n*	Specifies a sort key field in the rowset. The *recordname* prefix is required if you've specified a field that is not on the current record.
order_n	A single-character string. "A" specifies ascending order; "D" specifies descending order.

Returns

None.

Example

The first example repopulates a rowset in a page programmatically by first flushing its contents, selecting new contents using Select, then sorting the rows in ascending order by EXPORT_OBJECT_NAME:

```
Function populate_rowset;

   &RS1 = GetLevel0()(1).GetRowset(SCROLL.EXPORT_OBJECT);
   &RS1.Flush();
   &RS1.Select(RECORD.EXPORT_OBJECT, "where export_type =:EXPORT_TYPE_VW.EXPORT_⇒
TYPE");
   &RS1.Sort(EXPORT_OBJECT_NAME, "A");

End-Function;
```

See Also

Chapter 33, "Rowset Class," Select, page 1439 and Chapter 32, "Row Class," GetRowset, page 1412.

Chapter 32, "Row Class," GetRowset, page 1412

Rowset Class Properties

In this section, we discuss the Rowset class properties. The properties are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

ActiveRowCount

Description

This property returns the number of active (non-deleted) rows in the rowset.

This property returns a value of 1 for an empty scroll (Flush always leaves an empty row.) You can use the IsChanged or IsNew properties to verify if the row is new.

This property is read-only.

Example

```
&tmp = &ROWSET.ActiveRowCount;
```

ChangeOnInit

Description

Normally, if a field value is changed, whether through PeopleCode or by an end user, the IsChanged property for the row is set to True. The exception to this is when a change is done in the FieldDefault or FieldFormula event, that is, if a value is set in one of these events, the row is not marked as changed.

At save time, all newly inserted *and* changed rows are written to the database. All newly inserted but *not* changed rows are *not* written to the database.

Use this property to specify whether a change made to a new row from RowInit or RowInsert PeopleCode is marked as changed. This property takes a Boolean value. The default value is True.

Setting this property to False causes changes to fields for the rowset done in RowInit and RowInsert PeopleCode to *not* mark a new row as IsChanged.

This property is read-write.

Runtime Considerations

You must set this property *before*anything is changed for a row, otherwise the row is marked as changed.

This property propagates to child rowsets. That is, if it's set for a parent rowset before doing an insert row, it's automatically set on any and all child rowsets.

In addition, parent rows are marked as changed if this property is set only for the child rowset. That is, if you've changed a level three child of a level two row, the level two and level one rows are marked as changed to maintain data integrity.

DataAreaCollapsed

Description

This property specifies whether the view of a data area is collapsed or expanded.

Note. You must set the Collapsible Data Area field on the properties for the level-based control for this property to have any effect.

Copyright © 1988-2007, Oracle. All rights reserved.

This property changes to reflect the current state of the data area, according to whether the user has collapsed or expanded it. Changing the value collapses or expands the data area, but it does *not* prevent the user from collapsing (or expanding) it themselves.

Note. Because the user can change the value of this property, whatever value is set in PeopleCode isn't guaranteed to be still set the next time it is checked, because the user may have collapsed or expanded the data area in the meantime.

This property overwrites the value of the Default Initial View to Expanded field set in PeopleSoft Application Designer. For example, if Default Initial View to Expanded is selected in PeopleSoft Application Designer, then the value for the DataAreaCollapsed property is set to True, the control initially displays collapsed.

This property takes a Boolean value: True, initially display the data area collapsed, False, initially display the data area expanded.

This property is read-write.

Note. To collapse just a group box, use the DataAreaCollapsed field method.

See Also

Chapter 15, "Field Class," DataAreaCollapsed, page 500

Example

The following example checks the number of rows in the level one rowset. If the number of rows returned is larger than 100, the data area is initially displayed collapsed.

```
If &Level1.RowCount > 100 Then
   &Level1.DataAreaCollapsed = True;
Else
   &Level1.DataAreaCollapsed = False;
End-If;
```

DBRecordName

Description

This property returns the name of the primary database record as a string for the rowset.

This property is read-only.

Example

```
&DBNAME = &ROWSET.DBRecordName;
```

DeleteEnabled

Description

This property determines whether a rowset allows rows to be deleted (the equivalent of the user pressing ALT+8 and ENTER). This property takes a Boolean value.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. This property controls only whether an end-user can delete a row. Rows can still be deleted using PeopleCode.

The initial value of this property depends on how the scroll was created at design time. If the No Row Delete setting is selected on the Use tab of the scroll properties dialog box, DeleteEnabled will be False: otherwise it will be True.

Note. If No Row Delete is selected in PeopleSoft Application Designer, setting DeleteEnabled to True will *not* override this value. To control whether a rowset allows deletions at runtime, you should *not* select No Row Delete at design time.

For consistency, PeopleSoft recommends that either all rowsets at a level should disable deletions, or they should all allow deletions.

For rowsets created with non-Component Processor data (such as message rowsets, Application Engine rowsets, and so on) this property has no effect.

Note. Don't use this property with rowsets that are created using CreateRowset. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

This property is read-write.

See Also

Chapter 33, "Rowset Class," DeleteEnabled, page 1446 and Chapter 33, "Rowset Class," InsertEnabled, page 1448.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Using Page Controls," Using Scroll Areas and Scroll Bars

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

EffDt

Description

This property references the effective date of the primary record associated with the rowset. If the primary record associated with the rowset is not effective-dated, this property has a null value. To find the primary record associated with a rowset object, you can use the DBRecordName property.

Note. This property isn't valid with rowsets created using the CreateRowset function.

This property is read-only.

Example

```
&tmp = &ROWSET.EffDt;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

EffSeq

Description

This property references the effective-sequence number of the primary record associated with the rowset. If the primary record associated with the rowset does not have an effective-sequence number, this property has the value 0. To find the primary record associated with a rowset object, you can use the DBRecordName property.

Note. This property isn't valid with rowsets created using the CreateRowset function.

This property is read-only.

Example

```
&tmp = &ROWSET.EffSeq;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowset

InsertEnabled

Description

This property determines whether a rowset allows rows to be inserted (the equivalent of the user pressing ALT+7 and ENTER). This property takes a Boolean value.

Note. This property controls only whether an end-user can insert a row. Rows can still be inserted using PeopleCode.

The initial value of this property depends on a value set at design time. If the No Row Insert setting is selected on the Use tab of the scroll properties dialog box, InsertEnabled is False: otherwise it is True.

Note. If No Row Insert is selected in PeopleSoft Application Designer, setting InsertEnabled to True does *not* override this value. To control whether a rowset allows inserts at runtime, you should *not* select No Row Insert at design time.

For consistency, PeopleSoft recommends that either all rowsets at a level should disable inserts, or they should all allow inserts.

For rowsets created with non-Component Processor data (such as message rowsets, Application Engine rowsets, and so on) this property has no effect.

Note. Don't use this property with rowsets created using CreateRowset. Rowsets created using the CreateRowset function are standalone rowsets, not tied to the database, so there are no database updates when they are manipulated. Delete and insert activity on these types of rowsets aren't automatically applied at save time.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 33, "Rowset Class," DeleteEnabled, page 1446

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Application Designer, "Using Page Controls," Using Scroll Areas and Scroll Bars

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using Methods and Built-In Functions," Using Standalone Rowsets

IsEditError

Description

This property is True if an error has been found on any field in any record in any row or child rowset of the current rowset after executing the ExecuteEdits method on either a message object or a record object. This property can be used with the Field Class properties EditError (to find the field that's in error), MessageSetNumber and MessageNumber to find the error message set number and error message number.

This property is read-only.

Example

The following is an example showing how IsEditError, along with ExecuteEdits could be used:

```
&REC.ExecuteEdits();
If &ROWSET.IsEditError Then
   For &I = 1 to &ROWSET.ActiveRowCount
      &ROW = &ROWSET(&I);
         For &J to &ROW.RecordCount
            &REC = &ROW.GetRecord(&J);
               For &K = 1 to &REC.FieldCount
                  If &REC.GetField(&K).EditError Then
                     LOG_ERROR();
                        /* application specific call */
                  End-If;
            End-For;
         End-For;
   End-For;
End-If;
```

See Also

Chapter 31, "Record Class," ExecuteEdits, page 1386, Chapter 31, "Record Class," IsEditError, page 1403, Chapter 15, "Field Class," EditError, page 504, Chapter 15, "Field Class," MessageNumber, page 512 and Chapter 15, "Field Class," MessageSetNumber, page 513.

Level

Description

This property returns the level, that is, the nesting depth, of the rowset object. The top-level rowset has a level number of 0.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

```
&tmp = &ROWSET.Level;
```

Name

Description

This property refers to the name of the rowset. This property returns different values, based on the type of rowset.

Type of Rowset	Returns
Rowset created using GetLevel0	returns an empty string
Component buffer rowset	returns primary record name
Message rowset	returns primary record name
Component Interface rowset	returns primary record name
Application Engine rowset	always returns AESTATE

This property is read-only.

Example

```
&tmp = &ROWSET.Name;
```

ParentRow

Description

This property returns a row object containing a reference to the parent row, that is, the row containing the rowset. If this is a top-level rowset (level zero), the ParentRow property has a null value.

This property is read-only.

Example

```
&tmp = &ROWSET.ParentRow.RowNumber;
/* note that RowNumber is a property of the row class */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ParentRowset

Description

This property returns a rowset object containing a reference to the parent rowset, that is, the rowset containing the rowset. If this is a top-level rowset (level zero), the ParentRowset property has a null value.

This property is read-only.

Example

```
&tmp = &ROWSET.ParentRowset.Level;
/* note Level is another property of the rowset class */
```

RowCount

Description

This property returns the total number of rows in the rowset. It includes deleted rows. (The ActiveRowCount property doesn't include deleted rows.)

This property is read-only.

Example

```
&tmp = &ROWSET.RowCount;
```

SetComponentChanged

Description

This property determines whether any changes to the rowset using PeopleCode marks the component as changed and the data written to the database if the rowset is not a based on a derived/work record.

This property takes a Boolean value: true, changes to the rowset mark the component as changed, false, the component is treated as if unchanged. The default is true.

If you set this property to false, this means that no row insert, row delete, field change on the rowset using PeopleCode would cause the system to treat the component data as changed.

The SetComponentChanged rowset class property overrides the SetComponentChanged field class property. If a field has SetComponentChanged set to true, but its associated rowset has SetComponentChanged set to false, any change to the field does not mark the component as changed.

This property is read-write.

See Also

Chapter 15, "Field Class," SetComponentChanged, page 515

TopRowNumber

Description

This property returns the row that is being displayed at the top of the scroll (if any) for the rowset.

Copyright © 1988-2007, Oracle. All rights reserved.

Generally, this property is used to return the top row number of a scroll. However, sometimes you want to reposition the scroll. For example, if you use SetCursorPos to move the focus to a given field, there is no guarantee that the row containing the field is at the top of the scroll.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 34

RowsetCache Class

This chapter provides an overview of the RowsetCache class and discusses:

- Using the RowsetCache class
- Data type of a RowsetCache object
- Scope of a RowsetCache object
- RowsetCache object reference

Understanding a Rowset Cache

Many PeopleSoft applications use a metadata based design, where application PeopleCode reads metadata from a database record or some other source, then presents the user with an interface that is modified based on the metadata.

PeopleTools stores application data in a database cache to increase system performance. The RowsetCache class enables you to access this memory structure, created at runtime, and shared by all users.

Data that would qualify as metadata and would be appropriate for the RowsetCache class would be data that was commonly used yet fairly static, for example, country or currency tables.

The RowsetCache class does not provide a mechanism for synchronizing the data between database tables and the cache. If the underlying data changes, you must repopulate the rowset cache.

Note. Non-base language users may see different performance due to language table considerations.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Improving Your PeopleCode," Setting MaxCacheMemory

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Setting Application Server Domain Parameters," Cache Settings

Using the RowsetCache Class

The first time a user accesses a database, the rowset cache for that database is automatically created and populated. After that first user, all users and applications can take advantage of the rowset cache.

Copyright © 1988-2007, Oracle. All rights reserved.

If the underlying data for a rowset cache changes, at the very least your application must delete the existing rowset cache. Your application can also delete, then recreate and repopulate the rowset cache if it changes the underlying data. For example:

```
Local Rowset &RS;
Local RowsetCache &CRS;
Local Boolean &RSLT = False;

Repeat
   &I = &I + 1;
   If &RS(&I).IsChanged Then
      &RSLT = True;
   End-If;
Until &RSLT or
   &RS.RowCount = &I;

If &RSLT Then
   &CRS = GetRowsetCache(&RS);
   &CRS.Delete();
   &CRS = CreateRowsetCache(&RS);
   &CRS.Save();
End-if;
```

PeopleSoft recommends that any standard operations used by your application, such as deleting and repopulating the rowset cache, be encapsulated in an application class.

See Chapter 6, “Application Classes,” page 153.

The RowsetCache contains a rowset, and serializes the contained rowset to binary form for speed. As with most PeopleTools objects, RowsetCache objects are cached to memory and file, but they are also cached to the data base.

RowsetCache objects are intended to be local to a data base : they should not be transferred to another data base.

Error Handling

Every time you use the GetRowsetCache function, you should verify that the function executed successfully by testing for a null object. For example:

```
Local RowsetCache &RSC;

&RSC = GetRowsetCache(Rowset.MyRowset);

If (&RSC <> NULL) Then
   /* do processing */
Else
   /* call to populate rowset cache */
End-if;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Data Type of a RowsetCache Object

Cached rowset objects are declared as type RowsetCache. For example,

```
Local RowsetCache &Cache;
```

Scope of a RowsetCache Object

A RowsetCache object can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, Component Interface PeopleCode, and so on.

Note. PeopleSoft recommends that you do not specify RowsetCache objects as Global, due to their potential size.

RowsetCache Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowsetCache

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowsetCache

RowsetCache Class Methods

In this section, we discuss the RowsetCache class methods. These methods are discussed in alphabetical order.

Delete

Syntax

```
Delete()
```

Description

Use the Delete method to delete the RowsetCache from memory. The RowsetCache must have already been created, either using the CreateRowsetCache and saving the RowsetCache, or by a user using the application.

If this method is called as part of a larger transaction, such as in the SavePreChange or SavePostChange event, the commit is tied to the commit of the outer transaction. If an error occurs in this method, the outer transaction will be rolled back.

If this method is invoked in a non-transactional PeopleCode event (such as RowInit) the commit is controlled by the regular Component Processor flow.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A Boolean value: True if the delete was successful, False if the specified RowsetCache wasn't found.

This method terminates with an error message if there was another problem.

Example

```
Local RowsetCache &CRS;
Local Boolean &RSLT;

&CRS = GetRowsetCache(Rowset.MyRowset);

/* do processing*/

If (&CRS <> NULL) Then
   &RSLT = &CRS.Delete();
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateRowsetCache and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetRowsetCache.

Get

Syntax

```
Get()
```

Description

Use the Get method to convert a RowsetCache object into a rowset object. After using this method, you can manipulate the data in the rowset like you would any other rowset, using the regular rowset methods and properties.

Parameters

None.

Returns

A rowset object populated with the cached rowset data.

Example

```
Local Rowset &RS;
Local RowsetCache &CRS;

&CRS = GetRowsetCache(Rowset.MyRowset);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
If (&CRS <> NULL) Then
   &RS = &CRS.Get();
Else
   /* do error processing */
End-if;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowsetCache

Chapter 33, "Rowset Class," page 1421

Save

Syntax

Save([[**Rowset.**]*name*] [, *Description*])

Description

Use the Save method to save a new or existing rowset to the cache. If a rowset of the specified name already exists, it will be replaced. Otherwise, the rowset will be added.

If this method is called as part of a larger transaction, such as in the SavePreChange or SavePostChange event, The commit is tied to the commit of the outer transaction. If an error occurs in this method, the outer transaction will be rolled back.

If this method is invoked in a non-transactional PeopleCode event (such as RowInit) the commit is controlled by the regular Component Processor flow.

Parameters

Parameter	Description
Rowset. *name*	Specify the name of the rowset to be saved to the cache. If you just specify *name*, you must enclose the name in quotation marks.
Description	Specify a text string describing the rowset.

Returns

A Boolean value: True if the save was successful, False otherwise.

Example

The following code cashes &RS as a rowset definition.

```
Local RowsetCache &CRS;
Local Rowset &RS;

&CRS = GetRowsetCache(Rowset.MyRowset);

If (&CRS <> Null) Then
   &RS = &CRS.Get();
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
. . . /* do processing */
Else
   /* do error processing */
End-if;

&CRS.Save();
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetRowsetCache

RowsetCache Class Properties

In this section, we discuss the RowsetCache class properties These properties are discussed in alphabetical order.

Content

Description

Use this property to specify the contents of a RowsetCache object.

This property is read-write.

Example

The following code example verifies if the rowset cache is populated. If it isn't, the Else statement populates the rowset cache.

```
&Cache2 = GetRowsetCache("CACHE1");
If &Cache2.Get() <> Null Then
   WinMessage("Cache found");
Else
   &RS = CreateRowset(Record.PSLANGUAGES);
   &NUM_READ = &RS.Fill();
   &Cache2.Content = &RS;
   &Cache2.Description = "test " | %Language_Data;
   &RSLT = &Cache2.Save();
End-If;
```

Description

Description

This property can be used to set the description of the rowset cache as a string.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 35

Search Classes

This chapter provides an overview of Search classes and discusses the following topics:

- Search class hierarchy
- Error handling
- Search object declaration
- Scope of the search objects
- Session class method
- Search class reference section

See Also

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Configuring Search and Building Search Indexes"

Understanding Search Class

The Search API enables you to create a logical search index as well as access a search index and query its contents. Every search index in the PeopleSoft system must have a logical search index created to describe it.

The SearchIndex objects can be created for the following types of content:

- Indexing content references in a PeopleSoft portal
- Indexing data in PeopleSoft records
- Spidering a file system
- Spidering URL's
- User defined indexes (such as a PeopleSoft application)

Portal users can index and search on content references through the Search API using PeopleCode. The Portal Administration pages provide a GUI access to indexing a portal. This search index is what is used when doing searches on the portal. Users can create search indexes and index data in PeopleSoft records. The Search Index Designer provides a GUI tool to create these indexes.

Users can create search indexes and spider files systems or URL's. The Search Index Designer provides a GUI tool to create these indexes.

Users can create their own custom search indexes.

PeopleSoft has integrated the Verity search engine into PeopleTools.

Copyright © 1988-2007, Oracle. All rights reserved.

All searches done using the Search API are executed against a Verity search index. A search index is similar to a record in many ways:

- It has fields that hold information about a document.
- It stores one row of information for each document indexed.
- It can store information in different languages.

You must build a search index before you can use the Search API.

For the portal, you can build a search index either using the Search API (BuildSearchIndex) PortalRegistry method) or using the Portal Administration.

To index and search something that isn't registered in the portal, you must build your own search index.

After you've built a search index, you can execute a search against the index, and execute a search query. A search query (SearchQuery object) represents the items in an index that match a query. These matched items are retrieved in the SearchResult collection.

In addition, you can also check for any errors that were generated by the search.

If you specify multiple indexes to search over, and none of the collections' indexes are built, an error is returned. Otherwise, if there is at least one built index collection, and there is a matching search result, it is displayed.

See Also

Chapter 35, "Search Classes," BuildSearchIndex, page 1467

Using the SearchResults Collection

You specify the size of the SearchResult collection when you execute the search. If more SearchResults were returned than can be contained in the single SearchResult collection, you can access the next set of SearchResults by executing the search again, specifying only a different starting point.

For example, suppose your search returns 28 documents that match the query, but the size you specified when you executed the search was 10. You have to execute the search three times to access all the documents that matched the query.

The following pseudo-code could be behind a "Next Matches" button.

- &Start indicates at what document you want to start your search query.
- &Size is the number of search results returned in every search.
- &SearchQuery is the query object you created at the start of your program.

```
&Start = &Start + &Size;
&SearchQuery.Execute(&Start, &Size);
```

Understanding SearchResults

Each SearchResult in the SearchResult collection contains:

- Key
- Score
- SearchFields

 Copyright © 1988-2007, Oracle. All rights reserved.

The key uniquely identifies each document in the SearchResult collections. Keys provide a link back to the item that was indexed. The value returned by the Key property depends on the search index. If you're using a search index with the portal, the key contains the content provider and the URL. If you're not using the portal search index, the key is determined by the developer who created the search index.

Each query returns matched documents in relevance-ranked order, with those documents considered most relevant appearing at the top of the list. During search processing, a score is calculated for each retrieved document. This is the value stored in the Score property. A Score is assigned a value from 0.00 to 1.00, where 1.00 represents a perfect match to the search criteria provided.

Each SearchResult contains fields. If this is a non-portal search, these fields were specified by the developer when they created the search index. If this is a portal search, the fields are the following:

Field	Description
VALID_FROM_DATE	Valid from date
VALID_TO_DATE	Valid to date
CREATION_DATE	Creation date
CONTENT_PROVIDER	Content provider name
URL	URL
DESCRIPTION	Description

An external URL has the full URI attached so it might look like this:

```
URL: http://sports.mysports.com/nba/teams/chi/
```

Whereas a URL with a content provider specified is comprised of two portions.

```
Content Provider: HRMS
URL:ICType=Panel&Menu=ADMINISTER_WORKFORCE_(U.S.)&Market=GBL&PanelGroupName=⇒
ABSENCE_HISTORY1
```

You can interrogate each field in the SearchField collection to find its name and value.

See Also

Chapter 35, "Search Classes," Key, page 1478

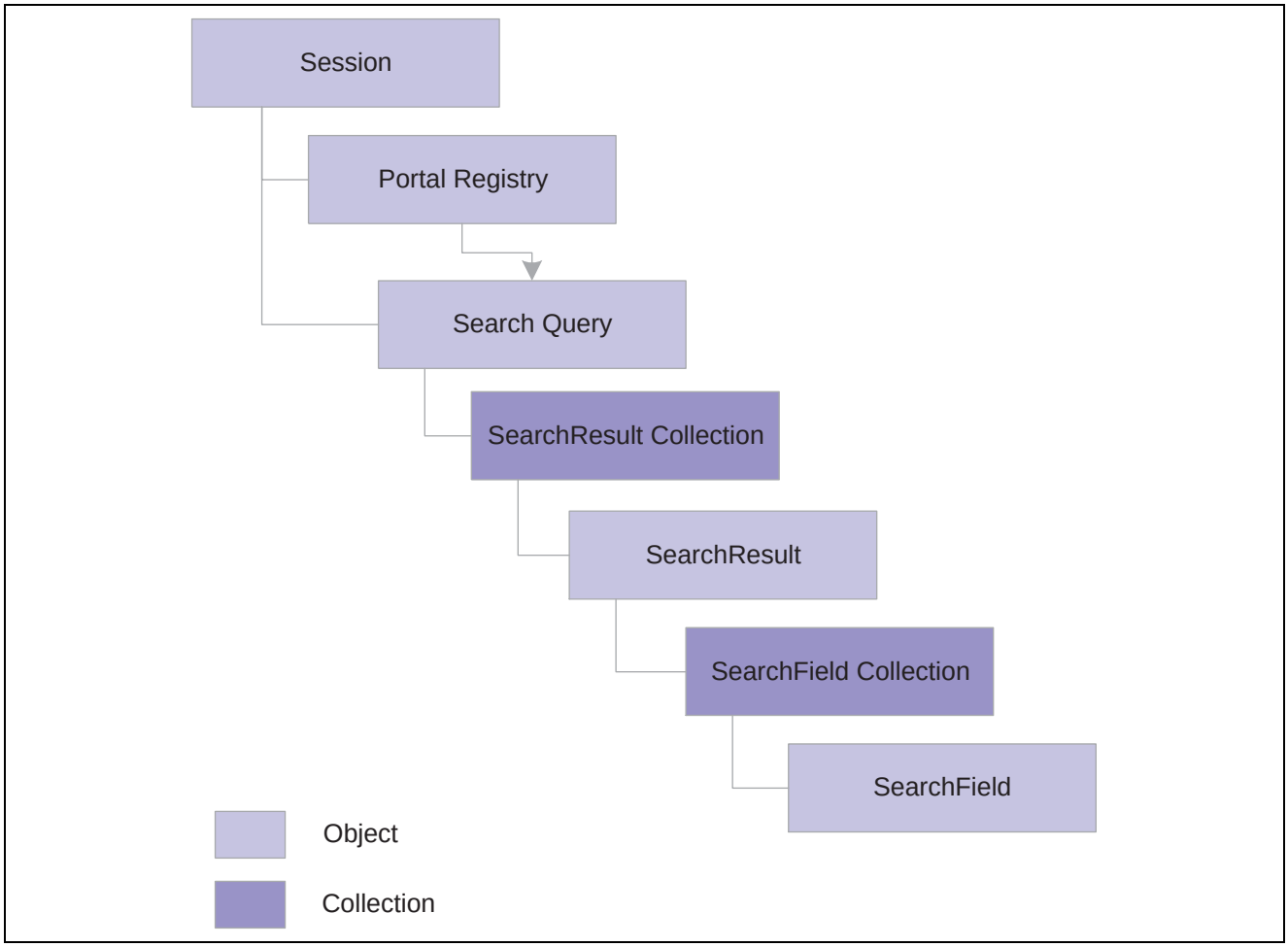
Chapter 35, "Search Classes," Score, page 1478

Chapter 35, "Search Classes," SearchFields, page 1478

Copyright © 1988-2007, Oracle. All rights reserved.

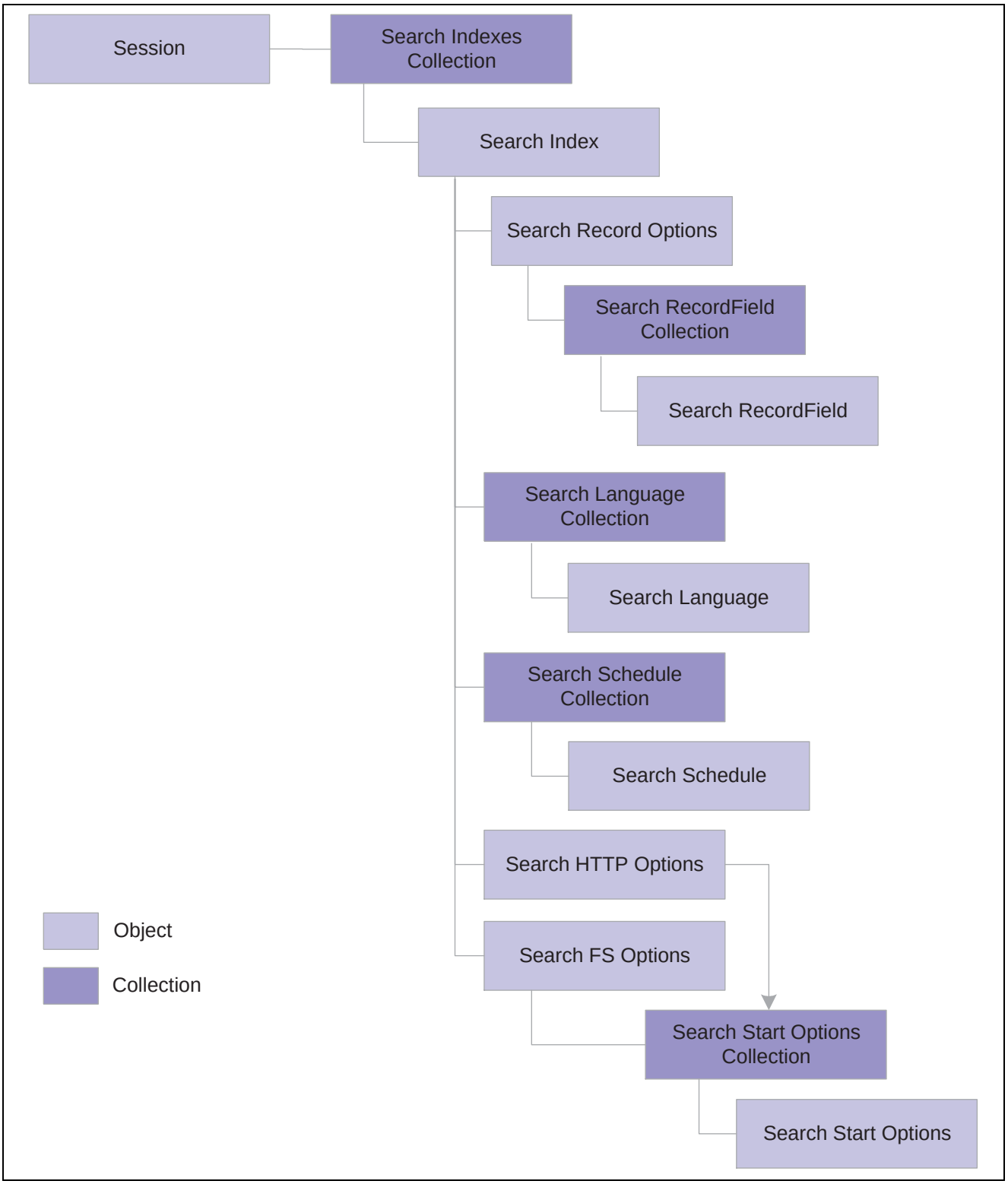
Search Classes Hierarchy

There are many different classes used with the Search API. The following flowchart shows the different classes, and how they are interrelated.



Search class hierarchy (part 1 of 2)

Copyright © 1988-2007, Oracle. All rights reserved.



Search class hierarchy (part 2 of 2)

Copyright © 1988-2007, Oracle. All rights reserved.

Error Handling

All errors for the Search API, like the other APIs, are logged in the PSMessages collection, instantiated from a session object.

The search classes log errors "interactively", that is, as they happen. For example, suppose you specified an invalid SearchField name. The error would be logged in the PSMessages collection as soon as you executed the ItemByName method.

If you want to search for errors in the query string before you execute the search, you can use the SearchQuery Parse method.

When to check for errors depends on your application. However, if you check for errors after every assignment, you may see a performance degradation.

The easiest way to check for errors is to check the number of messages in the PSMessages collection, using the Count property. If the Count is 0, there are no errors.

```
Local ApiObject &MySession;
Local ApiObject &ERRORCOL;
Component ApiObject &SearchQuery, &SearchResultCol, &SearchResult;
Component Number &Start, &Size;

&MySession = %Session;

If &MySession Then
   /* connection is good */

   &SearchQuery = &MySession.GetSearchQuery();
   &SearchQuery.Indexes = "PSC";
   &SearchQuery.Language = %Language;
   &SearchQuery.QueryText = SEARCH_RECORD.USER_QUESTION.Value;

   &Start = 1;
   &Size = 20;

   &SearchResultsCol = &SearchQuery.Execute(&Start, &Size);

   If &SearchResult.HitCount > 0 Then
      For &I = 1 to &SearchResultCol.Count
         &SearchResult = &SearchResultCol.Item(&I);

         /* Do processing */

      /* Do error checking */

         &ERRORCOL = &MySession.PSMessages;
         If (&ERRORCOL.Count <> 0) Then
            /* errors occurred . . . do processing */
         Else
            /* no errors */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
            End-If;

         End-For;
      Else
         /* do processing for no returned matches to query */
      End-if;
   Else
      /* do processing for no connection */
   End-If;
```

See Also

Chapter 36, “Session Class,” Error Handling, page 1512

Chapter 35, “Search Classes,” Parse, page 1469

Data Type of Search Objects

All search objects, like a SearchResult collection, a SearchField, and so on, are declared as type ApiObject. For example,

```
Local ApiObject &SearchResultCol;
Local ApiObject &SearchField;
```

Note. All Search objects can be declared only as Local.

Scope of the Search Objects

A search object can be instantiated only from PeopleCode using a PortalRegistry or Session object.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on. The limits placed on where you can use this object depend on how the object is used. For example, if you’re doing search index administration, you are updating the database, and that can be done only in certain events.

Note. The Search API currently does not work in an Application Engine program run on the System 390.

Session Class Methods

A SearchQuery object must be instantiated from either a session object or a Portal Registry object. The Search API doesn’t have any built-in functions.

In this section, we discuss the Session class methods. The methods are discussed in alphabetical order.

See Also

Chapter 36, “Session Class,” page 1511 and Chapter 27, “PortalRegistry Classes,” page 997.

Copyright © 1988-2007, Oracle. All rights reserved.

GetSearchIndexes

Syntax

```
GetSearchIndexes()
```

Description

The GetSearchIndexes method returns a collection of SearchIndexes objects.

A SearchIndexes object and its children are used only for index administration, that is, creating and building search indexes. This method is *not* used with queries.

Parameters

None.

Returns

A SearchIndexes Collection object, populated with zero or more SearchIndex objects.

See Also

Chapter 35, "Search Classes," SearchIndexes Collection, page 1481

GetSearchQuery

Syntax

```
GetSearchQuery()
```

Description

The GetSearchQuery method returns an empty search query object used to start a search.

Parameters

None.

Returns

An empty search query object.

Example

```
&SearchQuery = %Session.GetSearchQuery();
```

See Also

Chapter 35, "Search Classes," GetSearchQuery, page 1467

PortalRegistry Class Methods

A SearchQuery object must be instantiated from either a session object or a Portal Registry object. The Search API doesn't have any built-in functions.

In this section, we discuss the PortalRegistry class methods. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 29, "ProcessRequest Class," page 1193

BuildSearchIndex

Syntax
BuildSearchIndex(*Language*)

Description

The BuildSearchIndex method builds a portal search index for the specified language.

Search Collection Path Considerations

The Search Collection Path field in the configuration file points to a directory where collections can be built and queried through the search API. However, the BuildSearchIndex method *always* builds the collection in the default path, that is, in the following:

```
<PS_HOME>\data\search\<portal_name>\<db_name>\<lang_cd>
```

Parameters

Parameter	Description
Language	Specify the language code that you want the search index built in.

Returns

An optional Boolean value: True if the search index is built successfully, False otherwise.

GetSearchQuery

Syntax
GetSearchQuery()

Description

The GetSearchQuery method returns an empty search query object.

Note. When you use this method to get a search query, the index property is already set to the portal registry search index, and the language property is defaulted to the end-user's language.

Parameters

None.

Returns

An empty SearchQuery object with the index and language property already set.

Example

```
&Portal = %Session.GetPortalRegistry();
&Portal.Open("PORTAL");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&SearchQuery = &Portal.GetSearchQuery();
```

See Also

Chapter 35, "Search Classes," GetSearchQuery, page 1466

SearchQuery Class

The SearchQuery object is used to specify the search index and execute a query against it.

A SearchQuery object is returned by the following:

- The GetSearchQuery method from a PortalRegistry object.
- The GetSearchQuery method from a Session object.

See Chapter 35, "Search Classes," GetSearchQuery, page 1467.

See Chapter 35, "Search Classes," GetSearchQuery, page 1466.

SearchQuery Class Methods

In this section, we discuss the SearchQuery class methods.

Execute

Syntax

Execute(*Start*, *Size*)

Description

The Execute method runs a search query beginning at the document number specified by *Start*, and returns the number of results (or less) specified by *Size* in a SearchResult Collection.

Parameters

Parameter	Description
Start	Specify the document with which you want to start your query search. This parameter takes an integer, 1 or higher.
Size	Specify the number of results that you want returned. This parameter takes an integer value.

Returns

A SearchResult collection.

Example

```
&SearchResultCol = &SearchQuery.Execute(1, 20);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 35, “Search Classes,” SearchResult Collection, page 1476

Parse

Syntax

Parse()

Description

Use the Parse method to verify the syntax of the query string specified through the QueryText property for syntax errors, such as whether there is a missing quotation mark or closing bracket.

The Parse method does not produce a collection of search results. If there are any errors, it returns a reference to a ParseResult collection that you can search through for errors. If there are not any errors, it returns a Null object.

Parameters

None.

Returns

A reference to a ParseResult collection if there are errors, Null otherwise.

Example

```
Local ApiObject &MySession;
Local ApiObject &ParseResults;
Component ApiObject &SearchQuery, &SearchResultCol, &SearchResult;
Component Number &Start, &Size;

&MySession = %Session;

If &MySession Then
   /* connection is good */

   &SearchQuery = &MySession.GetSearchQuery();
   &SearchQuery.Indexes = "PSC";
   &SearchQuery.Language = %Language;
   &SearchQuery.QueryText = SEARCH_RECORD.USER_QUESTION.Value;

   &ParseResults = &SearchQuery.Parse();

   If All(&ParseResults) AND &ParseResults.ErrorCount > 0 OR &Parse⇒
Results.WarningCount > 0 Then

      /* Process results */

   End-if;
End-if;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 35, “Search Classes,” ParseResult Collection, page 1473

SearchQuery Class Properties

In this section, we discuss the SearchQuery class properties. The properties are discussed in alphabetical order.

HitCount

Description

The HitCount property returns the total number of documents that actually match the query.

This property is read-only.

Indexes

Description

Use the Indexes property to specify the list of indexes you want the search to query. This list is in the form of a list of comma separated index names. Use the Execute method to execute the query on all the indexes specified in this property.

If you want to search against a single index, specify the index name without any comma.

This property is read-write.

Example

The following example uses two search indexes called CATALOG1 and CATALOG2 for the word "bicycle", with the requested fields called "DESCRIPTION" and "PRICE" in the results.

```
&sqQuery = %Session.GetSearchQuery();
&sqQuery.Indexes = "CATALOG1, CATALOG2";
&sqQuery.Language = %Language;
&sqQuery.Querytext = "bicycle";
&sqQuery.SortSpecifications = "PRICE desc DESCRIPTION asc";
&sqQuery.RequestedFields = "PRICE, DESCRIPTION";

/* Execute the search, getting the first 20 results. */
&srcResults = &sqQuery.Execute(1, 20);
If &srcResults <> Null And &srcResults.Count > 0 Then

&srCurrent = &srcResults.First();
Repeat
       Local ApiObject &srFieldColl;
&srFieldsColl = &srCurrent.SearchFields;
       Local ApiObject &srField = &srFieldColl.First();
 Repeat
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
      /* Do something with this field. */
         &srField = &srFieldColl.Next();
       Until None(&srField);
   Until None(&srCurrent);
Else
   ReportSearchAPIErrors();
   Local string &msg;
   &msg = MsgGetText(145, 40, "No results for your search");
End-If;

Function ReportSearchAPIErrors()
   Local ApiObject &msgColl = %Session.PSMessages;
   Local number &i;
   For &i = 1 To &msgColl.count
      Local ApiObject &msg = &msgColl.item(&i);
      Error (&msg.Text);
   End-For;
   &msgColl.DeleteAll();
End-Function;
```

IndexName

Description

Note. This property remains for backward compatibility only. Use the Indexes property instead.

See Chapter 35, "Search Classes," Indexes, page 1470.

The IndexName property specifies the SearchIndex this query is to retrieve the result from.

You *must* set this property before a query can be performed on the SearchQuery object returned from the GetSearchQuery method.

This property is read-write.

Example

```
&SearchQuery.IndexName = "PSA";
```

KnowledgeBase

Description

Use the KnowledgeBase property to specify a topicset (that is, a file generated by using the mktopics Verity command for a search query. This property takes the name of a file, as a string.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Language

Description

The Language property specifies the language of the search index.

The default value for this property is the current user's language.

This property is read-write.

ProcessedCount

Description

The ProcessedCount property returns the total number of documents that were subjected to the search query.

This property is read-only.

QueryText

Description

The QueryText property returns the query text to be submitted to the SearchQuery object.

This property is read-write.

RequestedFields

Description

The RequestedFields property specifies the set of fields to be returned with the results.

The value of this property is a list of fields, separated by commas.

If this property is not specified, all the fields (except a few internal fields) are returned with the result. You may get better performance by specifying only the fields you want returned.

This property is read-write.

See Also

Chapter 35, "Search Classes," SortSpecifications, page 1473 and Chapter 35, "Search Classes," SearchField Collection, page 1479.

ScorePrecision

Description

The ScorePrecision property specifies how precise the score should be displayed with the results.

The value of this property is a string, and can be one of the following (case is insensitive)

- 8bit
- 16bit
- 32bit

 Copyright © 1988-2007, Oracle. All rights reserved.

The system displays two digits after decimal point if "8bit" is selected, and 4 digits if "16bit" is selected. The default value of this property is "8bit".

If this property is set to any other value, the Execute method generates an error and terminates the search.

This property is read-write.

SortSpecifications

Description

The SortSpecifications property specifies how the results of a search should be sorted.

This property should be in the following format:

```
fieldname1 {asc | desc} [fieldname2 {asc | desc} ...]
```

where *fieldname1* and *fieldname2* are field names to be used to sort the results and **asc** and **desc** are keywords for sorting in ascending or descending order.

Specifying multiple fields allows you to perform a secondary sort within the primary sorted results. For example, specifying `price asc descr desc` sorts the returned results by price in ascending order. Within the price, the results are sorted by descr field in descending order.

The default value of this property is to sort the results by score field in descending order.

Note. If you specify this property, you must also specify the RequestedFields property.

See Also

Chapter 35, “Search Classes,” RequestedFields, page 1472

ParseResult Collection

A ParseResult Collection is returned from the Parse SearchQuery method.

See Chapter 35, “Search Classes,” Parse, page 1469.

ParseResult Collection Methods

In this section, we discuss the ParseResult collection methods. The methods are discussed in alphabetical order.

First

Syntax

```
First()
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The First method returns the first ParseResult object in the ParseResult collection. If the ParseResult collection is empty, it returns Null.

Example

```
&MyResult = &MyParseResultCollection.First();
```

Next

Syntax

Next()

Description

The Next method returns the next ParseResult object in the ParseResult collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

This method returns a Null when there are no more ParseResult objects in the collection.

Parameters

None.

Returns

Either a reference to a ParseResult object, or, if there are no more results in the ParseResult collection, a Null.

Example

```
Repeat
   /* &parseInfo.Message() contains the message, process it */
   &parseInfo.Next()
Until None(&parseInfo);
```

ParseResult Collection Properties

In this section, we discuss the ParseResult collection properties The properties are discussed in alphabetical order.

ErrorCount

Description

The ErrorCount property returns the number of ParseResult objects in this collection with a severity of ERROR.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
For &I = 1 to &ParseResults.ErrorCount
   /* process message */
End-For;
```

WarningCount

Description

The WarningCount property returns the number of ParseResult objects in this collection with a severity of WARNING.

This property is read-only.

ParseResult Class

A ParseResult object is returned from either the First or Next ParseResult Collection methods.

See Chapter 35, "Search Classes," ParseResult Collection, page 1473.

ParseResult Class Properties

In this section, we discuss the ParseResult class properties The properties are discussed in alphabetical order.

Message

Description

The Message property returns the text of the error or warning message as a string.

This property is read-only.

Severity

Description

The Severity property returns the severity of the message as a string. Values are:

Value	Description
ERROR	Message is an error message.
WARNING	Message is a warning message.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

SearchResult Collection

A SearchResult collection is returned from the Execute SearchQuery method.

See Chapter 35, "Search Classes," Execute, page 1468.

SearchResult Collection Methods

In this section, we discuss the SearchResult collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first SearchResult object in the SearchResult collection. If the SearchResult collection is empty, it returns Null.

Example

```
&MyResult = &MyCollection.First();
```

Item

Syntax

Item(*number*)

Description

The Item method returns the SearchResult object with the position in the SearchResult collection specified by *number*.

Parameters

Parameter	Description
Number	Specify the position number in the collection of the SearchResult object that you want returned. Values start at 1.

Returns

A SearchResult object if successful, Null otherwise.

Example

```
For &I = 1 to &MyCollection.Count
  &MyResult = &MyCollection.Item(&I);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
  /* Do processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next SearchResult object in the SearchResult collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyResult = &MyCollection.Next();
```

SearchResult Collection Property

In this section, we discuss the SearchResult collection property Count.

Count

Description

This property returns the number of SearchResult objects in the SearchResult collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchResult Class

The SearchResult object represents a specific SearchResult in a SearchResult Collection object.

SearchResult objects are instantiated from a SearchResult Collection with the First, ItemByName or Next methods.

See Chapter 35, "Search Classes," SearchResult Collection, page 1476.

SearchResult Class Properties

In this section, we discuss the SearchResult class properties. The properties are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

Key

Description

The Key property returns the key for this SearchResult object.

This property is read-only.

Score

Description

The Score property returns the score for this SearchResult object as a string. The decimal separator in this string depends on the Peoplesoft language that was used for search. For example, if you searched in Spanish, the decimal separator would be a ’,’.

Use this property to display the score.

If you convert this string to a number using the Value function, you might not receive the correct result. This is because the Value function always expects the decimal separator character to be a period. You need to explicitly replace the decimal separator with a period before using the Value function. To avoid such conversion problems, if you want to perform score arithmetic, such as evaluating a score to see if it’s below a threshold, use the ScoreAsNumber property.

This property is read-only.

See Also

Chapter 35, “Search Classes,” ScoreAsNumber, page 1478

ScoreAsNumber

Description

This property returns the relevancy score as a floating point number between 0 and 1. This property is the same as the Score property, except that this property returns the score as a floating point number instead of a string. Use this value to perform score arithmetic such as comparing it to a threshold value. If you want to display the score, use the Score property.

This property is read-only.

See Also

Chapter 35, “Search Classes,” Score, page 1478

SearchFields

Description

The SearchFields property returns a SearchField Collection.

This property is read-only.

See Also

Chapter 35, “Search Classes,” SearchField Collection, page 1479

 Copyright © 1988-2007, Oracle. All rights reserved.

SearchField Collection

A SearchField collection is returned by the SearchFields SearchResult property.

See Chapter 35, "Search Classes," SearchFields, page 1478.

SearchField Collection Methods

In this section, we discuss the SearchField collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first SearchField object in the SearchField collection. If the SearchField collection is empty, it returns Null.

Example

```
&MyField = &MyCollection.First();
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the SearchField object specified by *Name*.

Parameters

Parameter	Description
Name	Specify the name of the SearchField you want returned.

Returns

A SearchField object if successful, Null otherwise.

Example

```
&MyField = &MyCollection.ItemByName(VALID_TO_DATE);
```

Copyright © 1988-2007, Oracle. All rights reserved.

Next

Syntax

Next()

Description

The Next method returns the next SearchField object in the SearchField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyField = &MyCollection.Next();
```

SearchField Collection Property

In this section, we discuss the SearchField collection property Count.

Count

Description

This property returns the number of SearchField objects in the SearchField collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchField Class

A SearchField is returned by First, ItemByName, and Next SearchField Collection methods.

See Chapter 35, "Search Classes," SearchField Collection, page 1479.

SearchField Class Properties

In this section, we discuss the SearchField class properties. The properties are discussed in alphabetical order.

Name

Description

The Name property returns the name of this SearchField object as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Value

Description

The Value property returns the value for this SearchField object as a string.

This property is read-only.

SearchIndexes Collection

A SearchIndexes collection is returned by the GetSearchIndexes Session class property.

The SearchIndexes in this collection represent all of the *logical* search indexes that have been created in the PeopleSoft system.

Every search index built and used in a PeopleSoft system must have a logical search index associated with it.

See Chapter 35, "Search Classes," GetSearchIndexes, page 1466.

SearchIndexes Collection Methods

In this section, we discuss the SearchIndexes collection methods. The methods are discussed in alphabetical order.

First

Syntax

First()

Description

The First method returns the first SearchIndex object in the SearchIndexes collection. If the SearchIndex collection is empty, it returns Null.

Example

```
&MySearchI = &MyCollection.First();
```

ItemByName

Syntax

ItemByName(*Name*)

Description

The ItemByName method returns the SearchIndex object specified by *Name*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of the SearchIndex that you want returned.

Returns

A SearchIndex object if successful, Null otherwise.

Example

```
&MySearchI = &MyCollection.ItemByName("MySearchIndex");
```

Next

Syntax

Next()

Description

The Next method returns the next SearchIndex object in the SearchIndexes collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySearchI = &MyCollection.Next();
```

SearchIndex Collection Property

In this section, we discuss the SearchIndex collection property Count.

Count

Description

This property returns the number of SearchIndex objects in the SearchIndexes collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchIndex Class

A SearchIndex is returned by First, ItemByName, and Next SearchIndexes Collection methods.

 Copyright © 1988-2007, Oracle. All rights reserved.

A SearchIndex represents a logical search index, and all the information describing it, in the PeopleSoft system. Every search index built and used in a PeopleSoft system must have a logical search index associated with it.

See Chapter 35, "Search Classes," SearchIndexes Collection, page 1481.

SearchIndex Class Method

In this section, we discuss the SearchIndex class method Save.

Save

Syntax

```
Save()
```

Description

Use the Save method to save any changes to the search index.

Note. If you change a value of one of the properties of the subobjects (such as some of the search options, and so on) the change won't be written to the database until the SearchIndex object has been saved.

Parameters

None.

Returns

A Boolean value: True if the SearchIndex object is saved successfully, False otherwise.

SearchIndex Class Properties

In this section, we discuss the SearchIndex class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

ExtraOptions

Description

This property is used for the spidering search indexes. Parameters that are not covered in the SearchIndex object and its children can be passed to the Verity spidering utility using this property.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

FSOpts

Description

This property returns a reference to a Search FS Options object, for FSYS type search indexes.

This property is read-only.

See Also

Chapter 35, “Search Classes,” Search FS Options Class, page 1502

HTTPOpts

Description

This property returns a reference to a Search HTTP Options object, for HTTP type search indexes.

This property is read-only.

See Also

Chapter 35, “Search Classes,” Search HTTP Options Class, page 1499

Languages

Description

This property returns a reference to the Search Language collection.

This property is read-only.

See Also

Chapter 35, “Search Classes,” Search Language Collection, page 1492

Location

Description

This property specifies the file system location where the search index (collection) is located. You must use a full path name.

This property is read-write.

Name

Description

This property returns the name of the SearchIndex, as a string.

This property is read-only.

RecOpts

Description

This property returns a reference to a SearchRecord Options collection, for RECD type search indexes.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

See Also
Chapter 35, “Search Classes,” SearchRecord Options Class, page 1485

Schedules

Description

This property returns a reference to the Search Schedule collection.

This property is read-only.

See Also
Chapter 35, “Search Classes,” Search Schedule Collection, page 1495

Type

Description

The Type property specifies the type of index. This property takes a string value. The values are:

Value	Description
RECD	An index generated from a PeopleSoft record
HTTP	An index generated from a spider search of the World Wide Web (WWW)
FSYS	An index generated from a spider search of a file system
PRTL	An index generated from a portal
USER	A user defined index (that is, not one of the previous indexes)

This property is read-write.

SearchRecord Options Class

A reference to a search record options object is returned by the RecOpts SearchIndex property.

This object is valid only for Search Index types of RECD. These types of indexes are used to index data in a PeopleSoft database. The properties used with this object represent the values that must be set for this type of search index.

See Chapter 35, “Search Classes,” RecOpts, page 1484.

Copyright © 1988-2007, Oracle. All rights reserved.

SearchRecord Options Class Properties

In this section, we discuss the SearchRecord option class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

Fields

Description

This property returns a reference to a SearchRecordField collection.

The SearchRecordField collection describes the records and fields that are indexed.

The maximum number of fields returned as part of this collection is 50.

This property is read-only.

See Also

Chapter 35, "Search Classes," SearchRecordField Collection, page 1488

Filter

Description

This property specifies a filter (that is, a SQL WHERE clause) used to filter out unwanted records from being indexed, as a string.

This property is read-write.

IncrementalView

Description

This property specifies a view used for accessing the records that are to be used for incremental indexing, as a string.

This property is read-write.

RecName

Description

This property returns the record name for the primary record that the search index is based on. Multiple records can be used, but they must all share a common key structure. The record name returned with this property specifies this common base key structure.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

VeggieKey

Description

This property specifies a replacement pattern for the VdkVgwKey (VeggieKey) used by Verity to uniquely identify a document. The setting of this property determines what is filled in for the unique identifier for each document that gets indexed. Text in this string is replaced with either literal text in the unique identifier or a value determined by one of the following tags:

Tag	Description
<pairs/>	Inserts a string of NAME=VALUE; pairs for each key on the Record being indexed
<row/>	Inserts the Record keys in a SQL-like syntax
<field fieldname='*MYFIELD*'/>	Inserts the value of *MYFIELD* for the Row being indexed (if it exists in the record; an error occurs when you save the data if it is missing)
<sql stmt='*SQL STATEMENT*'/>	Inserts the value returned by the SQL statement. Only the first row returned is accepted. SQL statements that return more than one column cause an error at runtime.

To use these tags, you include them as part of the VeggieKey string. The indexing process replaces them with values. For example, given the following Row of data:

```
Record: PROD_SRCH
Field: PRODUCTID=A11111 (a key on the record)
Field: DESCR="Baby goldfish"
Field: PRICE=5.00
```

You could produce any of the following unique identifiers with various settings of VeggieKey:

VeggieKey Setting	String Produced
"<pairs/>"	"PRODUCTID=A11111;"
"SELECT * FROM <row/>"	"SELECT * FROM PS_PROD_SRCH WHERE PRODUCTID='A11111'"
"<pairs/> <field filedname='DESCR'/>"	"PRODUCTID=A11111; Baby goldfish"
"Total price: <sql stmt='SELECT SUM(PRICE) FROM PS_PROD_SRCH'/>"	"Total price: 550.78"

Copyright © 1988-2007, Oracle. All rights reserved.

Note that the last example is not truly unique, although it is legal. It is up to the application to make sure that the setting of VeggieKey produces a unique string for each indexed document, as the indexing process does not perform this check. Applications that use VdkVgwKey to identify the results of search queries cannot find the correct document if the documents do not have a unique key.

This property is read-only.

ZoneOptions

Description

This property specifies the zone option for the index as a string. Values are:

Value	Description
ONE	All the fields for each row of data are inserted into the same zone in the word index. This is a simple scheme which does not allow you to search only parts of the word index but may be appropriate for applications without easily-differentiated text regions.
FLD	One zone is created for each field. The data in that field for each row is inserted into the same zone. This enables applications to search only a specific part of the word index at a time, with a potential performance gain. This does not happen automatically, however; the query interface built by the application must support searching in specific zones. The name of the zone created is the name of the field. The build process may take longer with this option set.

This property is read-only.

SearchRecordField Collection

A SearchRecordField collection is returned from the Fields SearchRecord Options property.

The SearchRecordField collection describes the records and fields that are indexed.

The maximum number of fields returned as part of this collection is 50.

See Chapter 35, “Search Classes,” Fields, page 1486.

SearchRecordField Collection Methods

In this section, we discuss the SearchRecordField collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*Record, Field*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the DeleteItem method to delete the SearchRecordField specified by the record and field name.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

Parameters

Parameter	Description
Record	Specify a record name as a string.
Field	Specify a field name as a string.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

```
First()
```

Description

The First method returns the first SearchRecordField object in the SearchRecordField collection. If the SearchRecordField collection is empty, it returns Null.

Example

```
&MySearchField = &MyCollection.First();
```

InsertItem

Syntax

```
InsertItem(Record, Field)
```

Description

Use the InsertItem method to insert a SearchRecordField object into the SearchRecordField collection.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	Description
Record	Specify a record name as a string.
Field	Specify a field name as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a new SearchRecordField object if successful, Null otherwise.

ItemByName

Syntax

ItemByName(*Record, Field*)

Description

The ItemByName method returns the SearchRecordField object specified by the record and field name.

Parameters

Parameter	Description
Record	Specify a record name as a string.
Field	Specify a field name as a string.

Returns

A SearchRecordField object if successful, Null otherwise.

Example

```
&MySearchField = &MyCollection.ItemByName("SEARCH", "DATE");
```

Next

Syntax

Next()

Description

The Next method returns the next SearchRecordField object in the SearchRecordField collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySearchOpts = &MyCollection.Next();
```

SearchRecordField Collection Property

In this section, we discuss the SearchRecordField collection property Count.

 Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

This property returns the number of SearchRecordField objects in the SearchRecordField collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

SearchRecordField Class

A SearchRecordField is returned by the First, ItemByName, InsertItem, and Next SearchRecordField collection methods.

A SearchRecordField object describes the records and fields that are indexed.

See Chapter 35, "Search Classes," SearchRecordField Collection, page 1488.

SearchRecordField Class Properties

In this section, we discuss the SearchRecordField class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

FieldName

Description

This property returns the field name for this SearchRecordField as a string.

This property is read-only.

IsAttachment

Description

This property specifies whether the search record field is an attachment. This property takes a Boolean value: True, the field is an attachment, False otherwise.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

IsVerityField

Description

This property specifies whether the search record field is a Verity field. This property takes a Boolean value: True, the field is a Verity field, False otherwise.

This property is read-write.

IsWordIndex

Description

This property specifies whether the search record field is a word index. This property takes a Boolean value: True, the field is a word index, False otherwise.

This property is read-write.

RecordName

Description

This property returns the name of the record as a string.

This property is read-only.

Search Language Collection

A reference to a Search Language collection is returned by the Languages property.

See Chapter 35, "Search Classes," Languages, page 1484.

Search Language Collection Methods

In this section, we discuss the Search Language collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*Name*)

Description

Use the DeleteItem method to delete the Search Language specified by *Name*.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of the Search Language that you want to delete.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

```
First()
```

Description

The First method returns the first Search Language object in the Search Language collection. If the Search Language collection is empty, returns Null.

Example

```
&MyLang = &MyCollection.First();
```

InsertItem

Syntax

```
InsertItem(LanguageCode, MapLanguageCode)
```

Description

Use the InsertItem method to insert a Search Language object into the Search Language collection.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	Description
LanguageCode	Specify a language code as a string.
MapLanguageCode	Specify a language code to map *LanguageCode* to (typically this is the same as *LanguageCode*).

Returns

A reference to a new Search Language object if successful, Null if not successful.

Copyright © 1988-2007, Oracle. All rights reserved.

ItemByName

Syntax

ItemByName(*LanguageCode*)

Description

The ItemByName method returns the Search Language object specified by *LanguageCode*.

Parameters

Parameter	Description
LanguageCode	Specify the language code of the object that you want to retrieve.

Returns

A reference to a Search Language object if successful, Null otherwise.

Example

```
&MyLang = &MyCollection.ItemByName("ENG");
```

Next

Syntax

Next()

Description

The Next method returns the next Search Language object in the Search Language collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MyLang = &MyCollection.Next();
```

Search Language Collection Property

In this section, we discuss the Search Language collection property Count.

Count

Description

This property returns the number of Search Language objects in the Search Language collection, as a number.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Search Language Class

A reference to a Search Language object is returned by the First, ItemByName, InsertItem, and Next Search Language Collection methods.

See Chapter 35, "Search Classes," Search Language Collection, page 1492.

Search Language Class Properties

In this section, we discuss the Search Language class properties. The properties are discussed in alphabetical order.

LanguageCd

Description

This property specifies the language code for the Search Language object as a string.

This property is read-only.

MapLanguageCd

Description

Specify a language code to map the given language to (typically this is the same as LanguageCode).

This property is read-only.

Search Schedule Collection

A reference to a Search Schedule collection is returned by the Schedules SearchIndex property.

See Chapter 35, "Search Classes," Schedules, page 1485.

Search Schedule Collection Methods

In this section, we discuss the Search Schedule collection methods. The methods are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteItem

Syntax

DeleteItem(*Name*)

Description

Use the DeleteItem method to delete the Search Schedule specified by *Name*.

This method is not executed automatically. It is executed only when the SearchIndex is saved.

Parameters

Parameter	Description
Name	Specify the name of the Search Schedule that you want to delete.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

First()

Description

The First method returns the first Search Schedule object in the Search Schedule collection. If the Search Schedule collection is empty, this method returns Null.

Example

```
&MySch = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*RunCntrlID*)

Description

Use the InsertItem method to insert a Search Schedule object into the Search Schedule collection.

This method is not executed automatically. It is executed only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	Description
RunCntrlId	Specify the run control ID of the search schedule object as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a new Search Schedule object if successful, Null if not successful.

ItemByName

Syntax

ItemByName(*RunCntrlId*)

Description

The ItemByName method returns the Search Schedule object specified by *RunCntrlId*.

Parameters

Parameter	Description
RunCntrlId	Specify the run control ID of the search schedule object as a string.

Returns

A Search Schedule object if successful, Null otherwise.

Example

```
&MySchedule = &MyCollection.ItemByName("MySched");
```

Next

Syntax

Next()

Description

The Next method returns the next Search Schedule object in the Search Schedule collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySched = &MyCollection.Next();
```

Search Schedule Collection Property

In this section, we discuss the Search Schedule collection property Count.

Count

Description

This property returns the number of Search Schedule objects in the Search Schedule collection, as a number.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Search Schedule Class

A reference to a Search Schedule object is returned by the First, ItemByName, InsertItem, and Next Search Schedule Collection methods.

See Chapter 35, "Search Classes," Search Schedule Collection, page 1495.

Search Schedule Class Properties

In this section, we discuss the Search Schedule class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

BuildType

Description

This property specifies how you want the index rebuilt with this schedule item. Values are:

Value	Description
INCR	Incremental updates an existing index. Use this schedule item for indexes with small amounts of data, and for cleaning up large indexes periodically when they start to fragment.
RBLD	Rebuild deletes the index and starts from scratch. Use this to add to an existing, large index.

This property is read-write.

RunCntlId

Description

This property returns the Run Control ID associated with the schedule item.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

RunRecurrence

Description

This property specifies the name of the Recurrence Definition as a string. For example: "Daily Search Rebuild".

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Defining Jobs and JobSets," Creating Job Definitions

ServerName

Description

This is the name of the Process Scheduler Server that processes the requests for this schedule item, for example, "PSNT".

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Process Scheduler, "Setting Server Definitions"

Search HTTP Options Class

A reference to a Search HTTP Options object is returned by the HTTPOpts SearchIndex property.

This object is valid only for Search Index types of HTTP. These types of indexes spider URLs. The properties available using this object represent values that must be set for spidering URL's.

See Chapter 35, "Search Classes," HTTPOpts, page 1484.

Search HTTP Options Class Properties

In this section, we discuss the Search HTTP Option class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

AllowHTTPS

Description

This property is used to cause the HTTP Spider Gateway to descend HTTPS (SSL) links and HTTP links when it builds the index. This property takes a Boolean value: True, use the HTTPS (SSL) links, False otherwise.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

DomainLimit

Description

This property limits spidering (indexing) to the specified domain. This property takes a string value.

This property is read-write.

GlobList

Description

This property specifies a list of filename "globs" which the indexing process uses to filter its list. The GlobListType determines how the filtering is done. The value of this property is a space-separated list of filenames with or without wildcards. The following is an example of the value for this property:

```
"*.doc *.txt *.html robots.txt"
```

This property is read-write.

See Also

Chapter 35, “Search Classes,” GlobListType, page 1500

GlobListType

Description

This property determines how the values specified with GlobList are used. Values are:

Value	Description
ALL	Ignore the values specified with GlobList. Index all filenames encountered regardless of their pattern.
EXC	Exclude the values specified with GlobList. Index everything except names that match a pattern in GlobList.
INC	Include the values specified with GlobList. Index only filenames that match a pattern in GlobList and exclude all other filenames.

This property is read-write.

See Also

Chapter 35, “Search Classes,” GlobList, page 1500

LinkDepth

Description

This property determines how many links the HTTP Spider descends from its starting point as a number.

A value of 1, for example, indexes the original page as well as the documents linked to by that page, and then stop.

A value of 2 indexes the original page, and documents linked to it, as well as documents linked to by the second page.

 Copyright © 1988-2007, Oracle. All rights reserved.

PeopleSoft recommends not setting this value over 10 without combining it with the DomainLimit property as the amount of data retrieved into the index increases geometrically with the LinkDepth.

This property is read-write.

See Also
Chapter 35, "Search Classes," DomainLimit, page 1500

MIMEList

Description

This property is used in a similar fashion to GlobList. It is a space-separated list of MIME Types. The MIMEListType determines how they affect indexing. The following is an example of a value for this property:

```
"text/html text/plain application/*"
```

This property is read-write.

See Also
Chapter 35, "Search Classes," MIMEListType, page 1501 and Chapter 35, "Search Classes," GlobList, page 1500.

MIMEListType

Description

This property is used like GlobListType, but applies to MIMEList instead of GlobList. Instead of testing the pattern of the filename, this property tests the MIME-type detected for the document when determining whether to index it.

This property is read-write.

See Also
Chapter 35, "Search Classes," MIMEList, page 1501 and Chapter 35, "Search Classes," GlobListType, page 1500.

ProxyHost

Description

This property specifies the hostname or IP address of the HTTP proxy, if the HTTP Spider Gateway needs an HTTP proxy to connect to remote websites during the indexing process.

This property is read-write.

ProxyPort

Description

This property specifies the port number of the HTTP proxy set in ProxyHost as a number.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

StartOpts

Description

This property returns a reference to a Search Start Options collection.

This property is read-only.

See Also

Chapter 35, “Search Classes,” Search Start Options Collection, page 1504

Search FS Options Class

A reference to a Search FS Options object is returned by the FSOpts SearchIndex property.

This object is valid only for Search Index types of FSYS. These types of indexes spider a file system. The properties available using this object represent the values that must be set for spidering a file system.

See Chapter 35, “Search Classes,” FSOpts, page 1484.

Search FS Options Class Properties

In this section, we discuss the Search FS Option class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won’t take effect until the SearchIndex object has been saved.

GlobList

Description

This property specifies a list of filename "globs" which the indexing process uses to filter its list. The GlobListType determines how they affect indexing. The value of this property is a space-separated list of filenames with or without wildcards. The following is an example of the value for this property:

```
"*.doc *.txt *.html robots.txt"
```

This property is read-write.

See Also

Chapter 35, “Search Classes,” GlobListType, page 1502

GlobListType

Description

This property determines how the values specified with GlobList are used. Values are:

 Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
ALL	Ignore the values specified with GlobList. Index all filenames encountered regardless of their pattern.
EXC	Exclude the values specified with GlobList. Index everything except names that match a pattern in GlobList.
INC	Include the values specified with GlobList. Index only filenames that match a pattern in GlobList and exclude all other filenames.

This property is read-write.

See Also
Chapter 35, “Search Classes,” GlobList, page 1502

MIMEList

Description

This property is used like GlobList. It is a space-separated list of MIME Types. The MIMEListType determines how they affect indexing. The following is an example of the value for this property:

```
"text/html text/plain application/*"
```

This property is read-write.

See Also
Chapter 35, “Search Classes,” MIMEListType, page 1503 and Chapter 35, “Search Classes,” GlobList, page 1502.

MIMEListType

Description

This property is used like GlobListType, but applies to MIMEList instead of GlobList. Instead of testing the pattern of the filename, this property tests the MIME-type detected for the document when determining whether to index it.

This property is read-write.

See Also
Chapter 35, “Search Classes,” MIMEList, page 1503 and Chapter 35, “Search Classes,” GlobListType, page 1502.

StartOpts

Description

This property returns a reference to a Search Start Options collection.

This property is read-only.

See Also
Chapter 35, “Search Classes,” Search Start Options Collection, page 1504

Copyright © 1988-2007, Oracle. All rights reserved.

Search Start Options Collection

The values in the Search Start Options Collection get passed to the −start parameter of the Verity spider utility. These values represent the starting points for spidering.

A Search Start Option collection is returned from:

- StartOpts Search HTTP Options class property
- StartOpts Search FS Options class property

See Search HTTP Options Class property

See Chapter 35, “Search Classes,” StartOpts, page 1502.

See Search FS Options Class property

See Chapter 35, “Search Classes,” StartOpts, page 1502.

Search Start Options Collection Methods

In this section, we discuss the Search Start Options collection methods. The methods are discussed in alphabetical order.

DeleteItem

Syntax

DeleteItem(*Value*)

Description

Use the DeleteItem method to delete the Search Start Options specified by *Value*.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

Parameters

Parameter	Description
Value	Specify the value of the Search Start Option that you want to delete.

Returns

A Boolean value: True, item was deleted successfully, False otherwise.

First

Syntax

First()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The First method returns the first Search Start Options object in the Search Start Options collection. If the Search Start Options collection is empty, this method returns Null.

Example

```
&MySearchOpts = &MyCollection.First();
```

InsertItem

Syntax

InsertItem(*Value*)

Description

Use the InsertItem method to insert a Search Start Options object into the Search Start Options collection.

This method is executed immediately, however, the values are updated in the database only when the SearchIndex is saved.

If the item you are trying to insert already exists, this method returns Null.

Parameters

Parameter	Description
Value	Specify the value of the Search Start Option that you want to insert.

Returns

A reference to a new Search Start Option object if successful, Null if not successful.

ItemByName

Syntax

ItemByName(*Value*)

Description

The ItemByName method returns the Search Start Options object specified by *Value*.

Parameters

Parameter	Description
Value	Specify the value of the Search Start Options that you want returned.

Returns

A Search Start Options object if successful, Null otherwise.

Example

```
&MySearchOpts = &MyCollection.ItemByName("MyStartOptions");
```

Copyright © 1988-2007, Oracle. All rights reserved.

Next

Syntax

Next()

Description

The Next method returns the next Search Start Options object in the Search Start Options collection. You can use this method only after you have used the First method: otherwise the system doesn't know where to start.

Example

```
&MySearchOpts = &MyCollection.Next();
```

Search Start Options Collection Property

In this section, we discuss the Search Start Options collection property Count.

Count

Description

This property returns the number of Search Start Options objects in the Search Start Options collection, as a number.

This property is read-only.

Example

```
&COUNT = &MY_COLLECTION.Count;
```

Search Start Options Class

A reference to a Search Start object is returned by the First, ItemByName, InsertItem, and Next Search Start Options Collection methods.

See Chapter 35, "Search Classes," Search Start Options Collection, page 1504.

Search Start Options Class Properties

In this section, we discuss the Search Start Options class properties. The properties are discussed in alphabetical order.

Note. If you change a value of one of the properties, the change won't take effect until the SearchIndex object has been saved.

 Copyright © 1988-2007, Oracle. All rights reserved.

IsDomainRestricted

Description

The value of this property is passed to the **−domain** option of the Verity spider utility. This property specifies whether the domain is restricted. This property takes a Boolean value: True, the domain is restricted, False otherwise.

This property is read-write.

IsHostRestricted

Description

This property specifies whether the host is restricted. This property takes a Boolean value: True, the host is restricted, False otherwise.

This property is read-write.

Value

Description

This property returns the value of the Search Start Options, as a string.

This property is read-only.

Search API Examples

The following PeopleCode programs are examples of how to use the Search API.

General Purpose Search API Example

The following is a general example uses two search indexes called CATALOG1 and CATALOG2 for the word "bicycle", with the requested fields called "DESCRIPTION" and "PRICE" in the results.

```
&sqQuery = %Session.GetSearchQuery();
&sqQuery.Indexes = "CATALOG1, CATALOG2";
&sqQuery.Language = %Language;
&sqQuery.Querytext = "bicycle";
&sqQuery.SortSpecification = "PRICE desc DESCRIPTION asc";
&sqQuery.RequestedFields = "PRICE, DESCRIPTION";

/* Execute the search, getting the first 20 results. */
&srcResults = &sqQuery.Execute(1, 20);
If &srcResults <> Null And &srcResults.Count > 0 Then

&srCurrent = &srcResults.First();
Repeat
       Local ApiObject &srFieldColl;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&srFieldsColl = &srCurrent.SearchFields;
       Local ApiObject &srField = &srFieldColl.First();
 Repeat
      /* Do something with this field. */
         &srField = &srFieldColl.Next();
       Until None(&srField);
   Until None(&srCurrent);
Else
   ReportSearchAPIErrors();
   Local string &msg;
   &msg = MsgGetText(145, 40, "No results for your search");
End-If;

Function ReportSearchAPIErrors()
   Local ApiObject &msgColl = %Session.PSMessages;
   Local number &i;
   For &i = 1 To &msgColl.count
      Local ApiObject &msg = &msgColl.item(&i);
      Error (&msg.Text);
   End-For;
   &msgColl.DeleteAll();
End-Function;
```

Portal Search API Example

The following is a Portal search example.

```
&SearchKey = %Request.GetParameter("SEARCH_TEXT");
&StartPosition = %Request.GetParameter("START_POSITION")
SearchResultChunkSize = 1000;

/*---------------------------------------------------------------------------
  Get a portal registry object.
---------------------------------------------------------------------------*/
&Portal = %Session.GetPortalRegistry();

/*---------------------------------------------------------------------------
  Open the desired portal.
---------------------------------------------------------------------------*/
&Portal.Open("PORTAL")

/*---------------------------------------------------------------------------
  Get a search query object.
---------------------------------------------------------------------------*/
&SearchQuery = &PORTAL.GetSearchQuery();

/*---------------------------------------------------------------------------
  Set the text of the query the user entered
---------------------------------------------------------------------------*/
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&SearchQuery.QueryText = &SearchKey

/*---------------------------------------------------------------------------
  Execute the search passing in the starting position and "chunk size" (get me the⇒
 first 20 result or the third  20)
---------------------------------------------------------------------------*/
&SearchResultsColl = &SearchQuery.Execute(&StartPosition, &SearchResultChunkSize);
/*---------------------------------------------------------------------------
 Get the first result.
---------------------------------------------------------------------------*/
&SearchResult = &SearchResultsColl.First();

While (&SearchResult <> Null)

   /*---------------------------------------------------------------------------
     Example of getting the key.  In the portal case, the key is the URL, but in⇒
 the general case it could be a product id or some kind of other database key.
   ---------------------------------------------------------------------------*/
   &SearchKey = &SearchResult.Key;

   /*---------------------------------------------------------------------------
     Example of getting the Field.  In the portal case, the field is the URL.  Yes, it⇒
 is redundant with the key but the key includes some {} around the URL and this⇒
 just makes it easier to get the URL.
   ---------------------------------------------------------------------------*/
   &URLSearchField = &SearchResult.SearchFields.ItemByName("URL")
   &URL = &URLSearchField.Value

   /*---------------------------------------------------------------------------
     Another example of a field using dot notation to simplify the code.
   ---------------------------------------------------------------------------*/
   &ContentProvider = &SearchResult.SearchFields.ItemByName("Content⇒
Provider").Value

   /*---------------------------------------------------------------------------
     Call a function to insert the result into the page.
   ---------------------------------------------------------------------------*/
   AddStuffToPage(&URL, &ContentProvider);

   /*---------------------------------------------------------------------------
     Get the next result of the search.
   ---------------------------------------------------------------------------*/
   &SearchResult = &SearchResultsColl.Next();

End-While;
&Portal.Close();
```

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 36

Session Class

This chapter provides an overview of Session class and discusses the following topics:

- Error handling.
- Regional settings.
- Trace settings.
- Session object declaration.
- Scope of a Session object.
- Session class built-in function.
- Session class reference section.

Understanding Session Class

PeopleTools provides a family of APIs for external access into the PeopleSoft system. The Session class is the root of the APIs. It controls access to the PeopleSoft system, controls the environment, and enables you to do error handling for all APIs from a central location.

The PeopleSoft APIs are intended to be used, and are supported, in the following environments.

- PeopleSoft Windows client with either an existing two-tier connection or an existing connection to an application server.
- PeopleSoft Application Engine batch program.
- Non-PeopleSoft program connecting to an application server.

The PeopleSoft APIs are exposed to the following language environments:

- PeopleCode
- OLE/COM
- C/C++
- Java

The APIs that are accessible using a session object are:

- Component Interface Classes
- PortalRegistry Classes
- Query Classes
- Search Class

Copyright © 1988-2007, Oracle. All rights reserved.

- Tree Classes

Note. Tree Classes are accessible only through PeopleCode.

A session object controls the following:

- Security and access to the PeopleSoft system.
- Errors and error handling .
- Regional settings (that is, internationalization) for language, dates, times, and numbers.
- Tracing.

See Also

Chapter 12, “Component Interface Classes,” page 411

Chapter 27, “PortalRegistry Classes,” page 997

Chapter 30, “Query Classes,” page 1233

Chapter 40, “Tree Classes,” page 1609

Security and Access to the PeopleSoft System

A session object controls security and access to the PeopleSoft system. You must "sign-on" to the PeopleSoft session with a valid user ID and password when you use the Connect method. In addition, there’s an external authentication parameter that you can use with the connect.

Note. Additional authentication will be available after this release.

Error Handling

The session object handles error processing for all the APIs, such as Component Interfaces, the Query API, and so on. From the session object, you can check if there have been any errors.

Note. The exception to this is Subscription PeopleCode. All errors for Subscription PeopleCode get logged to the application message error table.

Error messages include system error messages or messages from the message catalog (a common set of error messages used by all PeopleSoft applications.) For a Component Interface, the session object level errors also contain any text errors that may have occurred from running ExecuteEdits.

On the session object, you can use the following properties to initially check for errors:

- ErrorPending indicates whether there are API errors
- WarningPending indicates whether there are API warnings

All errors are contained in the PSMessages collection. (The PSMessages property on a Session object returns this collection.)

 Copyright © 1988-2007, Oracle. All rights reserved.

Each item in this collection is a PSMessage object. A PSMessage object contains information about the specific error that has occurred, such as the explain text for the error, the message set number, and so on. (The type of information depends on the type of error.)

If the error was caused by a Component Interface, a contextual string is attached to the end of the Text property of the PSMessage object. This contextual string contains the exact location of the error, that is, which field in which data collection, on which row, caused the error. This string is also accessible (by itself) using the Source property.

When errors are loaded into the PSMessages collection depends on the type of error. System errors (that is, errors in the connection to the PeopleSoft Session) are logged as they occur. When an API error is logged depends on the API, as some APIs can be run in either interactive mode (meaning errors are logged as they happen) or in non-interactive mode (so errors are logged only when a particular method is run.)

If you are using Visual Basic or another COM environment, the PeopleSoft API system raises a COM exception the first time ErrorPending changes from False to True. It will *not* raise another exception until the PSMessages collection is cleared, which sets ErrorPending back to False.

Note. If you've called an API from an Application Engine program, all errors are also logged in the Application Engine error log tables.

See Also

Chapter 36, "Session Class," Accessing a PSMessages Collection, page 1522

Displaying Error Messages

Use the following PeopleCode to display messages from the PSMessage collection:

```
Local ApiObject &Session, &PSMessages;
   &Session = %Session;
   &PSMessages = &Session.PSmessages;
   If (&Session <> Null ) Then
       If &PSMessages.Count > 0 Then
               For &i = 1 To &PSMessages.Count
                  &MsgSetNbr = &PSMessages.Item(&i).MessageSetNumber;
                  &MsgNbr = &PSMessages.Item(&i).MessageNumber;
                  MessageBox( 0 , " ", &MsgSetNbr, &MsgNbr, "Message not found");
               End-For;
         &PSMessages.DeleteAll();
            End-If;
End-If;
```

Regional Settings

Regional settings enable the user to set the display of numbers, dates, times, and currencies to comply with usage in a specific locale. The session object exposes some of these values to the API through the RegionalSettings property. This property returns a regional settings object that has properties you can use to change these settings.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

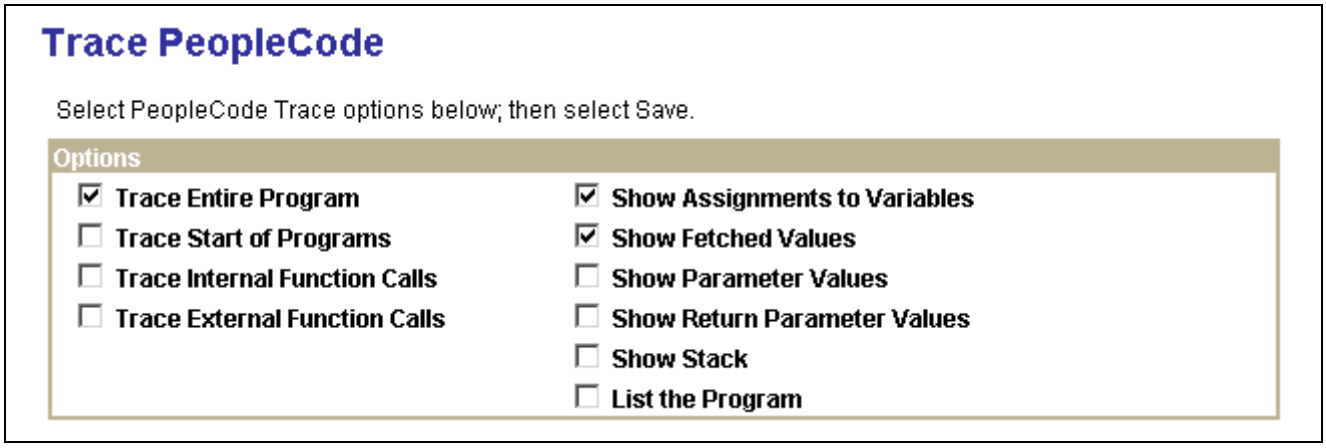
Chapter 36, "Session Class," Regional Settings Class Properties, page 1528

Trace Settings

The PeopleSoft debug pages enables you to control the environment in which your PeopleTools session is running. These pages include:

- Trace PeopleCode
- Trace SQL
- Trace Page

These pages enable you to select the SQL trace options and the PeopleCode trace options that you want. The TraceSettings object (assessable from a session object) lets you control many of the same options during your session.



Trace PeopleCode page

Not every option on the debug pages is represented with a trace settings property.

See Also

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Using Debug Utilities

Session Object Declaration

All session objects are declared as type ApiObject. For example,

```
Local ApiObject &MYSESSION;
Global ApiObject &PSMessage;
```

The following session objects can be declared as Global or Component:

- Session

 Copyright © 1988-2007, Oracle. All rights reserved.

- PSMessages Collection
- PSMessage

All other session objects *must* be declared as Local.

In this section, we discuss the following topics:

- State considerations.
- Considerations for declaring conditions.

State Considerations

In the PeopleSoft Pure Internet Architecture, all processing occurs on the server. If you do something in your PeopleCode program that causes a trip back to the user before the end of your PeopleCode program, the PeopleCode program *cannot*regain its state, that is, it can't reset local variables, and so on. This applies to all APIs that use the Session object.

If you must go back to the user before the end of your PeopleCode program, you must set all your objects to NULL before you go, or else you receive a runtime error.

For example, the following code causes an error:

```
&Session = %Session;
Rem &Session = Null;
WinMessage("Hello");
```

The following example does *not* cause an error:

```
&Session = %Session;
&Session = Null;
WinMessage("Hello");
```

Considerations for Declaring Collections

You *must* declare all collections for C/C++ and COM as objects, or else you receive errors at runtime.

For example if you're using RegionalSettings, include the following your declarations for a VB program:

```
Dim oRegionalSettings As Object
```

Then, use the following code to populate this variable:

```
Set oRegionalSettings = oCISession.RegionalSettings
oRegionalSettings.LanguageCd = "GER"
```

Scope of a Session Object

A Session can be instantiated from PeopleCode, from a Visual Basic program, from C++, COM, and so on.

See the example section in Connect for more details.

This object can be used anywhere you have PeopleCode, that is, in Application Engine PeopleCode, record field PeopleCode, and so on.

Copyright © 1988-2007, Oracle. All rights reserved.

Session Class Built-in Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “System Variables,” %Session

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” GetSession

Session Class Methods

In this section, we discuss the Session class methods. The methods are discussed in alphabetical order.

Connect

Syntax

```
(version, {"EXISTING" | //PSoftApplicationServer:JoltPort}, UserID, Password, ExtAuth)
```

Description

The Connect method connects a session object to a PeopleSoft application server.

Note. This function remains for backward compatibility only. Use the %Session system variable to return a reference of a session object instead.

If you already have a PeopleSoft session running (you are already connected to a PeopleSoft application server and are running a component, and so on) you must specify EXISTING, and not the *//PSoftApplicationServer:JoltPort*. Typically, use the parameter value of EXISTING from PeopleCode, not from other language environments.

If you are using an existing connection to the application server, you *cannot*specify a different user ID or password. If you don’t specify these values as NULL string, you must specify the exact same user ID (and password) as the one that originally started the session.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
version	Specify the API version that the client is expecting. Future releases will use this parameter. For now, you must use a 1.
EXISTING | *//PSoftApplicationServer: JoltPort*	Specify EXISTING if you're currently connected to an application server. Otherwise, specify the named application server machine and the IP port to connect to on an application server machine.
UserID	Specify the PeopleSoft user ID to use for the connection. This must be a valid user ID. If you are using an existing connection, you can specify a NULL string (that is, two quotation marks with no blank space between them ("")) for this parameter.
Password	Specify the password associated with the user ID to use for the connection. This must match the password assigned for this user ID *exactly*. If you are using an existing connection, you can specify a NULL string (that is, two quotation marks with no blank space between them ("")) for this parameter.
ExtAuth	This parameter is required, but it is unused in this release. You must enter a 0 for this parameter.

Returns

A Boolean value: True if the connection completed successfully, False otherwise.

Example

The following is an example of the code you would use if there is an existing connection to the application server:

```
Local ApiObject &MYSESSION;

&MYSESSION = GetSession();
&MYSESSION.Connect(1, "EXISTING", "", "", 0);
```

The following is an example connecting to a PeopleSoft system using a Visual Basic program. It first checks for error messages. If there are any errors, the text of the error is displayed to the user.

```
On Error GoTo eMessage

    Dim oSession As Object
    Dim oPSMessage As Object

    Dim oBC As Object

    Set oSession = CreateObject("PeopleSoft.Session")

    oSession.Connect 1, "//PSSOFTO061998:7000", "PTDMO", "PTDMO", 0

    &lsquo;Application Specific Processing

eMessage:
    If (oSession.PSMessages.Count > 0) Then
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
        For i = 1 To oSession.PSMessages.Count
           Set oPSMessage = oSession.PSMessages.Item(i)
           MsgBox (oPSMessage.Text)
        Next i
    End If
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "System Variables," %Session

Disconnect

Syntax

Disconnect()

Description

The Disconnect method disconnects the session associated with the session object executing the method from that PeopleSoft session.

This method returns a Boolean value: True if successfully disconnected, False otherwise.

API Instantiation Methods

The following is a list of the other methods used with the session object to instantiate an API object such as a Component Interface, a tree structure, and so on.

Component Interface:

- FindCompIntfcs(*partial_name*): returns a collection of Component Interfaces based on the partial Component Interface name
- GetCompIntfc(**COMPINTFC**.*componentname)*: returns a Component Interface object

Portal Registry:

- FindPortalRegistries(*partial_name*): returns a collection of PortalRegistry objects
- GetLocalNode(): returns a node object
- GetNodes(): returns a collection of node objects
- GetPortalRegistry(): returns a PortalRegistry object
- GetRemoteNodes(): returns a collection of node objects

Query:

- FindQueryDBRecords():returns a reference to a QueryDBRecord collection, filled with zero or more records
- FindQueries():returns a reference to a Query collection, filled with zero or more queries
- FindQueriesDateRange(*StartDateString*, *EndDateString*): returns a reference to a Query collection, filled with zero or more queries that match the specified date range
- GetQuery(): returns a query object

 Copyright © 1988-2007, Oracle. All rights reserved.

- GetQuerySecurityProfile(): a QuerySecurityProfile object
- SearchQueryDBRecord(*SearchType*, *Pattern*, *CaseSensitive*): returns a QueryDBRecord collection
- SearchPrivateQueries(*QueryType, SearchType*, *Pattern*, *CaseSensitive*): a query collection
- SearchPublicQueries(*QueryType, SearchType*, *Pattern*, *CaseSensitive*): a query collection

Search:

- GetSearchIndexes(): returns a SearchIndexes collection
- GetSearchQuery(): returns a search query object

Tree:

- GetTree(): returns a tree object
- GetTreeStructure(): returns a tree structure object

Session Class Properties

This section describes the Session class properties. The properties are discussed in alphabetical order.

ErrorPending

Description

This property indicates whether there are any error messages pending for the API that is currently running. After this property has been set, it returns True until the PSMessages collection is cleared (or deleted.)

This property is read-only.

Example

```
If &MYSESSION.ErrorPending Then
   &COUNT = &MYSESSION.PSMessages.Count;
   For &I = 1 To &COUNT
      &ERROR = &MYSESSION.PSMessages.Item(&I);
      &TEXT = &ERROR.Text;
      WinMessage("Error text " | &TEXT);
   End-For;

&MYSESSION.PSMessages.DeleteAll();
```

PSMessages

Description

This property returns a reference to a PSMessages collection object. If there are no messages, this property returns an object reference to PSMessages with a count of zero.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
If &SESSION.PSMessages.Count > NULL Then
   /* there are messages do processing */
End-if;
```

See Also

Chapter 36, "Session Class," PSMessages Collection Methods, page 1522

PSMessagesMode

Description

This property is used to determine how messages are output. This property takes either a numeric value or a constant. The default value is 1 (%PSMessages_CollectionOnly).

You must set this property before you check the type of message, that is, you can't check the type of message, then decide how it's displayed.

This property sets the value for the session. You can change modes during a session, such as, if you're starting a Component Interface. However, after you run the Component Interface, you should set the value back. For example:

```
&OldMode = &Session.PSMessageMode;
&Session.PSMessagesMode = 1;
...
&Session.PSMessagesMode = &OldMode;
```

You can specify either a numeric value or a constant. The values are:

Numeric Value	Constant Value	Description
0	%PSMessages_None	None.
1	%PSMessages_CollectionOnly	PSMessage Collection only (default).
2	%PSMessages_MessageBoxOnly	Message box only.
3	%PSMessages_Both	Both collection and message box.

Note. If you set this property to 0 (%PSMessages_None), *all* messages are ignored. Use this option with caution.

This property is read-write.

Example

```
Local ApiObject &SESSION;

&SESSION = %Session;
&MMODE = &SESSION.PSMessagesMode;
&SESSION.PSMessagesMode = %PSMessages_MessageBoxOnly;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

RegionalSettings

Description

This property returns a reference to a RegionalSettings collection object.

This property is read-only.

See Also

Chapter 36, “Session Class,” Regional Settings Class Properties, page 1528

Repository

Description

This property returns a reference to a Repository object.

See Also

Chapter 7, “API Repository,” page 193

SuspendFormatting

Description

This property turns off the PeopleSoft automatic formatting when coercing a string into a field. Formatting is typically necessary for external Component Interface users (such as for a Visual Basic program,) yet should be suspended for a PeopleCode program. You should not typically need to set this value yourself, as it’s set appropriately based on where the session object is created.

This property takes a Boolean value: True means formatting is suspended, False means it isn’t suspended. If you start your session from a PeopleCode program, the default for this property is True. If you don’t start the session from a PeopleCode program, the default is False.

This property is read-write.

TraceSettings

Description

This property returns a reference to a TraceSettings collection object.

This property is read-only.

See Also

Chapter 36, “Session Class,” Trace Setting Class Properties, page 1532

WarningPending

Description

This property indicates whether there are any warning messages pending for the API that is currently running.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Accessing a PSMessages Collection

Use the PSMessages collection to return all of the messages that occur during the session. The following example finds all the messages listed in the PSMessages collections when the Component Interface Save methods fails. The example assumes that &CI is the Component Interface object reference.

```
If Not &CI.Save() Then
   /* save didn&rsquo;t complete */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;

      /* do message processing */

   End-For;
&COLL.DeleteAll(); /* Delete all messages in queue */
End-if;
```

As you evaluate each message, delete it from the PSMessages collection. Or, delete all the messages in the queue at once using DeleteAll.

If there are multiple errors when saving a Component Interface, all errors are logged to the PSMessages collection, not just the first occurrence of an error.

Considerations for Declaring Collections

You *must* declare all collections for C/C++ and COM as objects, or else you receive errors at runtime.

PSMessages Collection Methods

This section describes the PSMessages collection methods. The methods are discussed in alphabetical order.

DeleteAll

Syntax

DeleteAll()

Description

The DeleteAll method deletes all the PSMessage objects in the PSMessages collection executing the method. You should use this property after you have processed all the errors in the collection. This method also resets the status of ErrorPending to False.

Returns

This method returns a Boolean value: True if all PSMessages in the collection were successfully deleted, False otherwise.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 36, "Session Class," ErrorPending, page 1519

DeleteItem

Syntax
DeleteItem(*number*)

Description

The DeleteItem method deletes the PSMessage object that exists at the *number* position in the PSMessages collection executing the method.

Parameters

Parameter	Description
number	Specify the position number in the collection of the PSMessage object that you want to delete. All other items within the collection are renumbered.

Returns

This method returns a Boolean value: True if the PSMessage object was successfully deleted, False otherwise.

Example

```
&ERROR.DeleteItem(&I);
```

First

Syntax
First()

Description

The First method returns the first PSMessage object in the PSMessages collection object executing the method.

Example

```
&ERROR = &PSMESSAGE.First();
```

Item

Syntax
Item(*number*)

Description

The Item method returns a PSMessage object that exists at the *number* position in the PSMessages collection executing the method

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
number	Specify the position number in the collection of the PSMessage object that you want returned.

Returns

A reference to a PSMessage object, NULL otherwise.

Example

```
&PSMSGCOL = &SESSION.PSMessages;
For &I = 1 to &PSMSGCOL.Count;
   &PSMESSAGE = &PSMSGCOL.Item(&I);
   /* do error processing */
End-For;
```

Next

Syntax

Next()

Description

The Next method returns the next PSMessage object in the PSMessages collection object executing the method. You can use this method only after you have used either the First or Item methods: otherwise the system doesn't know where to start.

PSMessages Collection Property

This section describes the PSMessages collection property.

Count

Description

This property returns the number of PSMessage objects in the PSMessages collection object.

This property is read-only.

Example

```
&COUNT = &PSMESSAGE.Count;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

PSMessage Class Properties

This section describes the PSMessages class properties. The properties are discussed in alphabetical order.

ExplainText

Description

This property returns the explanation text (as a string) for the error message associated with the PSMessage object. You can use this property in conjunction with the MessageNumber property (which contains the error message number) and the MessageSetNumber property (which contains the error message set number.)

Note. ExplainText should be accessed only when necessary because it requires a second database read or application server roundtrip.

This property is read-only.

MessageNumber

Description

This property returns the error message number (as a number) for the error message associated with the PSMessage object. You can use this property in conjunction with the MessageSetNumber property (which contains the error message set number) and the ExplainText property (which contains text explaining the error.)

This property is read-only.

MessageSetNumber

Description

This property returns the error message set number (as a number) for the error message associated with the PSMessage object. You can use this property in conjunction with the MessageNumber property (which contains the error message number) and the ExplainText property (which contains text explaining the error.)

This property is read-only.

MessageType

Description

This property indicates the type of the message, whether it's from the message catalog, if it's a warning, error, or information error.

This property returns a numeric value. The values are:

Value	Description
0	Message is not a message catalog entry or there is no message.
1	Message has a severity of Error.
2	Message has a severity of Warning.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
3	Message has a severity of Information.

This property is read-only.

Source

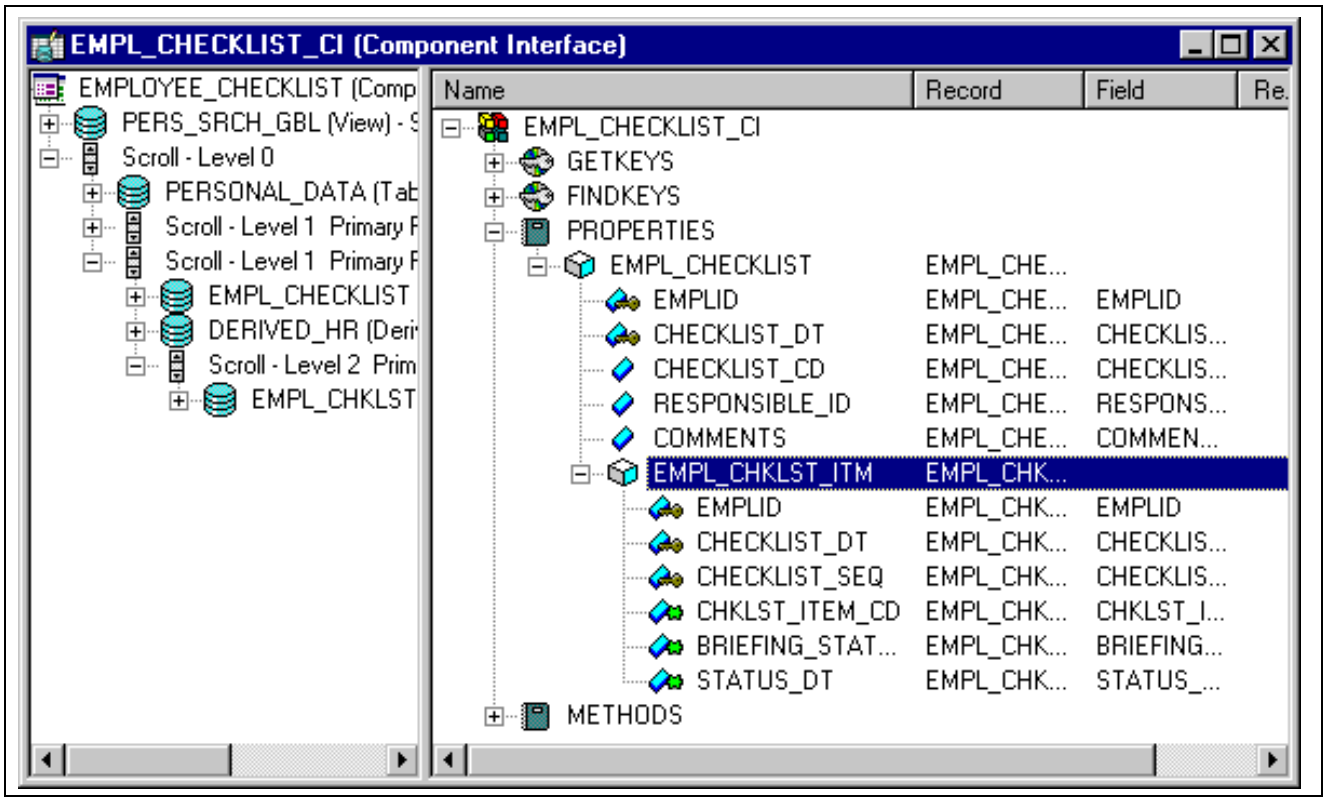
Description

This property returns a string indicating the actual field that's in error. The syntax of the string that's returned is as follows:

```
ComponentInterfaceName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName
```

This string is also returned as part of the Text property.

For example, suppose you had a Component Interface named EMPL_CHKLST_CI.



Sample Component Interface

The following indicates that the first level field, EMPLID, has the error:

```
EMPL_CHKLST_CI.EMPLID
```

The Component Interface EMPL_CHKLST_BC has a data collection (scroll) called EMPL_CHKLST_ITM. Suppose that it has 3 rows, and the STATUS_DT field was in error. The **Source** property would return the following string:

```
EMPL_CHKLST_CI.EMPL_CHKLST_ITM(3).STATUS_DT
```


Copyright © 1988-2007, Oracle. All rights reserved.

You can use the Source property in conjunction with the MessageNumber property (which contains the error message number), the MessageSetNumber property (which contains the error message set number), the ExplainText property (which contains text explaining the error), and the Text property (which contains the text of the message.)

This property is read-only.

Example

The following example code finds the error messages and displays them to the user. It finds the field that caused the error and displays that also.

```
Local ApiObject &PSMESSAGE;
Local Boolean &FIND;
Local string &SOURCE, &FIELD, &TEXT;
Local number &START, &LEN, &NSTART, &FLEN;
.
.
.
For &I = 1 to &SESSION.PSMessages.Count;
   &PSMESSAGE = &SESSION.PSMessages.Item(&I);
   /* only display errors, not warnings, to user */
   If &PSMESSAGE.Type = 1 Then
      &TEXT = &PSMESSAGE.Text;
      /* find name of field in error */
      &SOURCE = &PSMESSAGE.Source;
      &LEN = Len(&SOURCE);
      /* find last dot before field */
      &FIND = False;
      &START = Find(".", &SOURCE);
      While Not (&FIND)
         &NSTART = Find(".", &SOURCE, &START + 1);
            If &NSTART = 0 Then
               &FIND = True;
            Else
               &START = &NSTART;
            End-If;
         End-While;
         &FLEN = &LEN - &START;
         &FIELD = Substring(&SOURCE, &START + 1, &FLEN);
         /* display text and field to user */
         Winmessage("You received the following error: " | &TEXT | "For field " |⇒
 &FIELD);
   End-If;
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Text

Description

This property returns the text of the message (as a string) for the error message associated with the PSMessage object. It also includes a contextual string that contains the name of the field that generated the error. The contextual string has the following syntax:

```
{ComponentInterfaceName.[CollectionName(Row).[CollectionName(Row).[CollectionName⇒
(Row)]]].PropertyName}
```

The contextual string (by itself) is available using the Source property of the PSMessage.

You can use the Text property in conjunction with the MessageNumber property (which contains the error message number), the MessageSetNumber property (which contains the error message set number), and the ExplainText property (which contains text explaining the error.)

This property is read-only.

Example

```
Local ApiObject &MYSESSION, &COL, &ERROR;
Local string &TEXT;
.
.
.
If &MYSESSION.ErrorPending Then
   /* received error */
   &COLL = &MYSESSION.PSMessages;
   For &I = 1 to &COLL.Count
      &ERROR = &COLL.Item(&I);
      &TEXT = &ERROR.Text;
         /* do error processing */
      End-For;
End-if;
```

See Also

Chapter 36, “Session Class,” Source, page 1526

Regional Settings Class Properties

This section describes the regional settings class properties. The properties are discussed in alphabetical order.

ClientTimeZone

Description

This property specifies the client time zone. This property takes a string value.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

CurrencyFormat

Description

This property specifies how currency values are displayed. This property takes a number value. The default value is 1.

In the following values, @ represents the character specified with the CurrencySymbol property.

Value	Description
0	@1.1
1	1.1@
2	@.1.1
3	1.1.@

This property is read-write.

CurrencySymbol

Description

This property specifies the currency symbol. This property takes a string value. The default value is "$".

This property is read-write.

DateFormat

Description

This property specifies the date format defaults. PeopleTools supports the Short Date Format specification (MDY, DMY, or YMD), and the Date separator setting. When you add a Date field in a record definition, you indicate whether you want to display the century. PeopleTools Date and DateTime fields always show leading zeros for month, day, and year.

This property takes a number value. The values for this property are:

Value	Description
0	MDY (default)
1	DMY
2	YMD
3	Default value on system

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

DateSeparator

Description

This property specifies the character used to separate the month, day, year in the date. This property takes a string value. The default value is "/".

This property is read-write.

DecimalSymbol

Description

This property specifies the character used to separate the fractional of the number from the whole. This property takes a string value. The default value for this property is ".".

Note. If you change the value of this property to a comma (,), you should adjust the DigitGroupingSymbol to a different character or some applications may not operate correctly.

This property is read-write.

DigitGroupingSymbol

Description

This property specifies the character used as a thousand or other grouping separator. This property takes a string value. The default value for this property is ",".

This property is read-write.

LanguageCD

Description

This property specifies the language to be used for the transaction. This property takes a string value. The default value for this property is the user's base language.

This property is read-write.

Note. If you change this value dynamically using PeopleCode, this value will *not* change back to the original value when you disconnect. You must manually set it back to the original value.

Example

The following example sets the language code, then at the end of the program, sets it back to the original value.

```
Local ApiObject &SESSION;
Local ApiObject &CI;

/** Get Session ApiObject **/
&SESSION = %Session;

/** Get Message ApiObject **/
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&MSG = %IntBroker.GetMessage();
&RS = &MSG.GetRowset();

/** Get Component Interface **/
&CI = &SESSION.GetCompIntfc(compIntfc.NAMES_CI);

&REG_LNG = &SESSION.RegionalSettings.LanguageCd;

For &TRANS_NUM = 1 To &RS.RowCount

/** Store Language Code in Temp Var **/
&LNG = &RS(&TRANS_NUM).PSCAMA.LANGUAGE_CD.Value;

/** Set the Language for this Transaction **/
If &LNG <> &REG_LNG Then
&SESSION.RegionalSettings.LanguageCd = &LNG;
End-If;
/* application specific processing */

&SESSION.RegionalSettings.LanguageCd = &REG_LNG;
&SESSION.Disconnect();
End-For
```

UseLocalTime

Description

This property specifies whether the local time zone of the client should be used to format time fields. This property takes a Boolean value. The default value is True.

This property is read-write.

1159Separator

Description

This property specifies the morning designator. This property takes a string value up to five characters. The default value is AM.

Use the 2359Separator property to specify the afternoon designator.

This property is read-write.

Considerations Using the 1159Separator Property

You cannot use this property as you would other properties. You must use the ObjectGetProperty function for accessing the property. For example:

```
Local ApiObject &SESSION;
Local ApiObject &reg;

/** Get Session ApiObject **/
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&SESSION = %Session;
&reg = &SESSION.RegionalSettings;

&REG_LNG = &SESSION.RegionalSettings.LanguageCd;
&MornDesigination = ObjectGetProperty(&reg, "1159Separator");
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ObjectGetProperty

Chapter 36, "Session Class," 2359Separator, page 1532

2359Separator

Description

This property specifies the afternoon designator. This property takes a string value up to five characters. The default value is PM.

Use the 1159Separator property to specify the morning designator.

This property is read-write.

Considerations Using the 2359Separator Property

You cannot use this property as you would other properties. You must use the ObjectGetProperty function for accessing the property. For example:

```
Local ApiObject &SESSION;
Local ApiObject &reg;

/** Get Session ApiObject **/
&SESSION = %Session;
&reg = &SESSION.RegionalSettings;

&REG_LNG = &SESSION.RegionalSettings.LanguageCd;
&AfternoonDesigination = ObjectGetProperty(&reg, "2359Separator");
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ObjectGetProperty

Chapter 36, "Session Class," 1159Separator, page 1531

Trace Setting Class Properties

Each of the following properties relates to the debug pages in PeopleTools.

 Copyright © 1988-2007, Oracle. All rights reserved.

This section describes the trace setting class properties. The properties are discussed in alphabetical order.

See Also

Enterprise PeopleTools 8.49 PeopleBook: System and Server Administration, "Using PeopleTools Utilities," Using Debug Utilities

API

Description

This property is part of the PeopleSoft API trace and sets the API Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

COBOLStmtTimings

Description

This property is part of the SQL trace and sets the COBOL Statement Timings value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

ConnDisRollbackCommit

Description

This property is part of the SQL trace and sets the Connect, disconnect, rollback, and commit value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

DBSpecificCalls

Description

This property is part of the SQL trace and sets the Database API-specific Calls value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

ManagerInfo

Description

This property is part of the SQL trace and sets the Manager Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

NetworkServices

Description

This property is part of the SQL trace and sets the Network Services value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

NonSSBs

Description

This property is part of the SQL trace and sets the All Other API Calls besides SSBs value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

OutputUNICODE

Description

This property specifies if the trace file is written in ANSI or UNICODE (for example, Output Unicode Trace File).

This property takes a Boolean value: True means the file will be written in UNICODE, False means it will be ANSI.

This property is read-write.

PCExtFcnCalls

Description

This property is part of the PeopleCode trace and sets the Trace External Function Calls value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCFcnReturnValues

Description

This property is part of the PeopleCode trace and sets the Show Function Return Value value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

PCFetchedValues

Description

This property is part of the PeopleCode trace and sets the Show Fetched Values value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCIntFcnCalls

Description

This property is part of the PeopleCode trace and sets the Trace Internal Function Calls value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCListProgram

Description

This property is part of the PeopleCode trace and sets the List the Program value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCParameterValues

Description

This property is part of the PeopleCode trace and sets the Show Parameter Values value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCProgramStatements

Description

This property is part of the PeopleCode trace and sets the Trace Each Statement in Program value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCStack

Description

This property is part of the PeopleCode trace and sets the Show Stack value.

Copyright © 1988-2007, Oracle. All rights reserved.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCStartOfPrograms

Description

This property is part of the PeopleCode trace and sets the Trace Start of Programs value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCTraceProgram

Description

This property is part of the PeopleCode trace and sets the Trace Evaluator Instructions value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

PCVariableAssignments

Description

This property is part of the PeopleCode trace and sets the Show Assignments to Variables value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

RowFetch

Description

This property is part of the SQL trace and sets the Row Fetch value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SQLStatement

Description

This property is part of the SQL trace and sets the SQL Statement value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

SQLStatementVariables

Description

This property is part of the SQL trace and sets the SQL Statement Variables value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SSBs

Description

This property is part of the SQL trace and sets the Set Select Buffers value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SybBindInfo

Description

This property is part of the SQL trace and sets the Sybase Bind Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

SybFetchInfo

Description

This property is part of the SQL trace and sets the Sybase Fetch Information value.

This property takes a Boolean value: True means the trace will be run, False means it won't.

This property is read-write.

TraceFile

Description

This property specifies the location of the trace file (that is, PeopleTools Trace File value.)

This property takes a string value.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 37

SOAPDoc Class

This chapter provides an overview of SOAPDoc class and discusses the following topics:

- SOAP message exchange model.
- SOAP message document.
- The SOAPDoc class.
- SOAPDoc object creation.
- The ValidateSOAPDoc method.
- SOAPDoc objects and methods.
- Data type of a SOAPDoc object.
- Scope of a SOAPDoc object.
- SOAPDoc built-in functions.
- SOAPDoc class methods.
- SOAPDoc class properties.

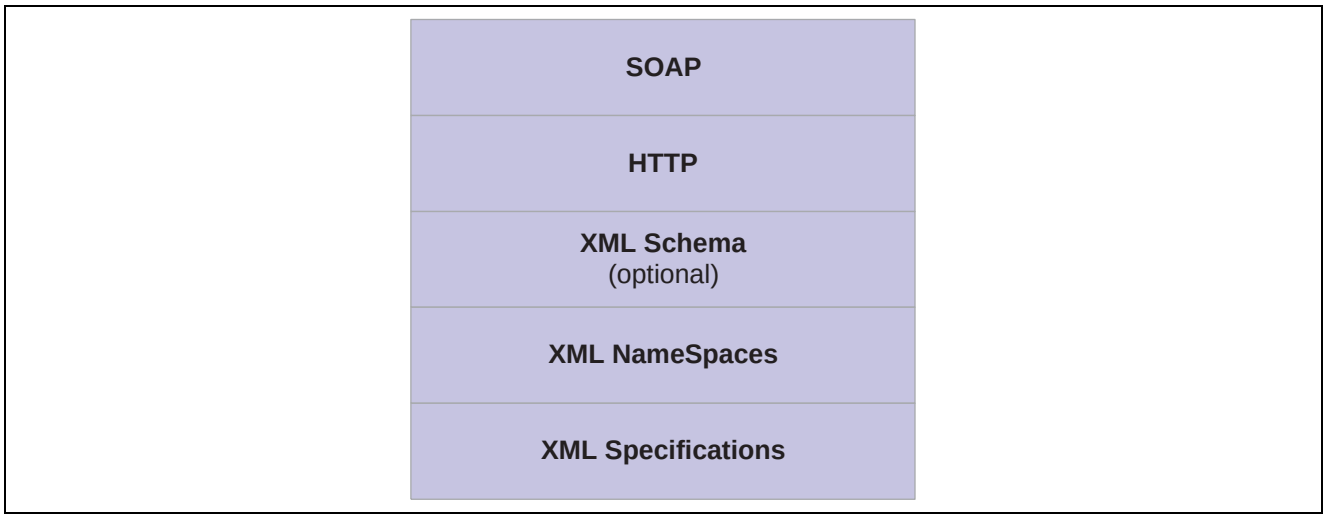
Understanding SOAPDoc Class

The Simple Object Access Protocol (SOAP) is a lightweight protocol for defining synchronous messages in a distributed Web environment. It's an XML based protocol that can be used in combination with other protocols such as HTTP using URLs as endpoints for a message. SOAP is an application of XML and XML namespaces and optionally uses XML Schemas.

SOAP is an industry standard, defined by the World Wide Web Consortium (W3C), that defines an XML grammar for identifying a method name and the method parameters and for returning the results.

More specific information about SOAP can be found in the W3C specification on SOAP. The SOAP serialization rules can also be found in the complete specifications and aren't outlined in this chapter. Also included in the W3C specification are sections on using SOAP with HTTP, which is also outside the domain of this chapter. You can also send and receive HTTP messages through the Integration Gateway.

Copyright © 1988-2007, Oracle. All rights reserved.



SOAP is an application built with XML and HTTP technologies

See Also

http://www.w3.org

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

SOAP Message Exchange Model

SOAP messages are always synchronous. They can implement Request/Response synchronous messages with the response having the output parameters returned in another SOAP document to the sender.

SOAP supports HTTP-POST and HTTP-GET formats. PeopleSoft supports only HTTP, using the Integration Gateway.

Processing the message requires validation of the mandatory sections of the SOAP XML document and understanding SOAP namespaces.

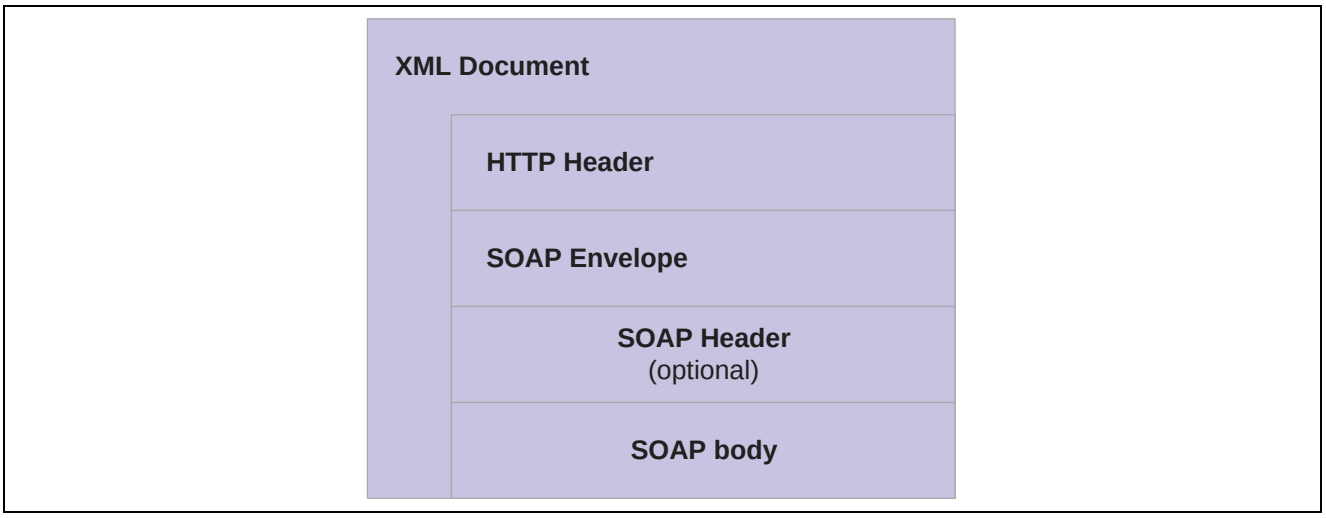
SOAP Message Document

A SOAP Message is an ordinary XML document that consists of the following parts:

- A mandatory SOAP envelope.
- An optional SOAP header.
- A mandatory SOAP body.

 Copyright © 1988-2007, Oracle. All rights reserved.

SOAP Message Part	Description
SOAP Envelope	Is the top element of the XML document representing the message. It defines the content of the message, as well as the framework of what is in a message, how to process it, who should deal with it and whether it is optional or mandatory.
SOAP Header	Contains header information, and attributes that can be set to encode and further identify the type of processing as well as additional features of the message. This portion is optional.
SOAP Body	Contains the call and response information intended for the recipient of the message. A SOAP Fault element can appear in the body of the SOAP message to carry error or status information.



SOAP parts in an XML document

The following are *required* for a SOAP message:

- Encoded using XML.
- Have a SOAP envelope.
- Have a SOAP body.
- Use the SOAP encoding and envelope namespaces.

The following are optional for a SOAP message:

- SOAP Header.
- XML Schema use.

The following are *invalid* for a SOAP message:

- DTD references.
- XML Processing Instructions.

Copyright © 1988-2007, Oracle. All rights reserved.

- XML CData Sections.

The SOAPDoc Class

The SOAPDoc class is a separate class in PeopleCode that works similarly to the XmlDoc and XmlNode classes. Many of the same methods and properties that work with an XmlDoc or XmlNode work with a SOAPDoc. However, there are also methods and properties that don't apply to a SOAPDoc.

The following XmlDoc and XmlNode methods do *not* apply to a SOAPDoc:

- AddCDataSection
- AddProcessInstruction
- CopyToPSFTMessage
- CreateDocumentType
- InsertCDataSection
- InsertProcessInstruction
- LoadIBContent

If a Merchant Profile has been created it can be used with the SOAPDoc class and the Message class. You can use the XmlDoc property to convert the message object to an SOAPDoc object, then use the SOAPDoc methods and properties, as well as the XmlDoc methods and properties that are applicable.

You can only use the message object when the XmlDoc object is structured.

It should be noted that a SOAPDoc object is really an extension of the XmlDoc class. You have the option of getting portions of the SOAPDoc–like the body node–and accessing it or manipulating it as if it were a standard element in an XML document, because in actuality it is.

SOAPDoc Object Creation

There are a few steps to creating a SOAPDoc. The following is an overview of these steps. More detail about the methods and properties mentioned in this section can be found in the reference section of this chapter.

SOAP Header Section

Because the SOAP header is optional, the header is not validated when you use the ValidateSOAPDoc method.

A SOAP header must be added *before* any other parts of the SOAPdoc.

Use the AddHeader method to add a header to a SOAPDoc.

Use the HeaderNode property to access an existing SOAPDoc header.

After you have a reference to the header, you can use XmlDoc methods to read the contents of the header (like any node.)

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 37, “SOAPDoc Class,” AddHeader, page 1551

Chapter 37, “SOAPDoc Class,” HeaderNode, page 1558

SOAP Envelope Section

A SOAP envelope is a required portion of the XML Message and is generated automatically using the SOAP schema as specified by SOAP W3C.

A SOAP envelope must be added *before* you add a SOAP body or a SOAP method. This is an optional method. You need to use this method only if you need to set any additional attributes for the Envelope section. The default SOAP schemas are added automatically.

To set additional custom attributes or schemas on the Envelope element, use the AddEnvelope method. This method adds the envelope element tags and the default schemas for either SOAP or UDDI depending on the value of the parameter passed in. The default encoding styles are also set here. Only one envelope section can be set within one PeopleSoft SOAP XmlDoc.

The SOAPDoc adds only a default URL to the envelope. For some UDDI sites, you may need to add a specific schema yourself. To do this, generate the envelope with no schema, then access the envelope section and add the specific schema URL using the AddAttribute method.

See Also

Chapter 37, “SOAPDoc Class,” AddEnvelope, page 1547

SOAP Body Section

The body is the XmlDoc where the method is defined. The method AddBody starts the body section of the XML SOAPDoc and must be in the XmlDoc before adding in method sections of the message. Only one body section can be set within one SOAPDoc. This is an optional method and must called only if you need to set any additional attributes for the body section. Otherwise the body section is automatically generated in the XmlDoc.

The SOAP body must be added *after* you add the SOAP envelope, and *before* you add any methods.

See Also

Chapter 37, “SOAPDoc Class,” AddBody, page 1546

SOAP Method Section

The AddMethod method adds the name of the method element being processed. Additional attributes for this method can be set using the AddAttribute and InsertAttribute XmlNode methods.

You don’t have to specifically add a SOAP body section. The body is automatically added when you call the AddMethod method.

The AddParm method sets the parameters for the parameter element of the method as a name, value pair. Multiple parameters can be set and are used in the order that this method is called. If an XML Schema is defined, types can be used using the AddAttribute XmlNode method. Any number of parameter name-value pairs can be added for a method and are added to the XmlDoc in the order that the AddParm methods are called.

You can add parameters to a method only *after* you add the method. The parameters are added in the order you add them in the PeopleCode program.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 37, “SOAPDoc Class,” AddMethod, page 1552

Chapter 37, “SOAPDoc Class,” AddParm, page 1554

Chapter 42, “XmlDoc Class,” AddAttribute, page 1765

SOAPDoc Section Access

Use the following properties to access the different parts of a SOAPDoc object. All of these properties return an XmlNode object.

- BodyNode
- EnvelopeNode
- MethodNode

After you have access to the portion of the SOAPDoc that you want, use the XmlNode properties and methods to manipulate the SOAPDoc.

Use the XmlDoc property to convert a SOAPDoc object to an XmlDoc object, and vice-versa.

The MethodName property returns the name of the method in the SOAPDoc object.

Note. If you use the MethodName property, you receive only the method name, as a string, not a reference to an XmlNode object that represents the method.

Use the GetParmName and GetParmValue methods to get the name and the values in a SOAPDoc object for each output parameter. You must use the index number of the parameter you want with these methods. You can determine the total number of parameters with the ParmCount property.

See Also

Chapter 37, “SOAPDoc Class,” BodyNode, page 1557

Chapter 37, “SOAPDoc Class,” EnvelopeNode, page 1557

Chapter 37, “SOAPDoc Class,” MethodName, page 1559

Chapter 37, “SOAPDoc Class,” MethodNode, page 1559

Chapter 37, “SOAPDoc Class,” GetParmValue, page 1555

PeopleCode to Create a SOAPDoc

The following is the order in which the Soap methods *must* be called in order to create a valid SOAPDoc.

```
&SOAPDoc = CreateSOAPDoc(); /* required */
&SOAPDoc.AddEnvelope(0); /* optional */
&SOAPDoc.AddBody(); /* optional */
&SOAPDoc.AddMethod("GetQuote", 1); /* required */
&SOAPDoc.AddParm("symbols", "PSFT");
&SOAPDoc.AddParm("format", "l1c1d1t1");
&OK = &SOAPDoc.ValidateSOAPDoc(); /* optional */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

SOAPDoc Fault Section

In the body of a SOAP XML response document, you can define a fault section. Generally this is used if in an inbound message has been processed and some sort of error occurred. The text of the message gives further information about the error.

Use the AddFault method to add a fault code and a fault string to the body of a SOAPDoc. Only one fault section can be set within one SOAP XML document.

If you receive a response message that contains a fault, use the FaultCode and FaultString methods to return these values.

See Also

Chapter 37, “SOAPDoc Class,” AddFault, page 1550

Chapter 37, “SOAPDoc Class,” FaultCode, page 1557

Chapter 37, “SOAPDoc Class,” FaultString, page 1558

The ValidateSOAPDoc Method

Use the ValidateSOAPDoc method to validate the SOAP document for correctness as follows:

- Encoded using XML.
- Has SOAP envelope.
- Has SOAP Body.
- Has a SOAP Method.
- Used SOAP encoding and envelope namespaces.

See Also

Chapter 37, “SOAPDoc Class,” ValidateSOAPDoc, page 1556

SOAPDoc Objects and Messages

To publish the data of a SOAPDoc, you must first transform the SOAPDoc into an XmlDoc object by using the XmlDoc property. After you do this, you can use the XmlDoc object with the SyncRequestXmlDoc, PublishXmlDoc, or GetXmlDoc functions, to publish the data either synchronously or asynchronously.

All XmlDoc messages must be associated with a message definition. An XmlDoc message must be associated with an *unstructured* message, that is, a message definition that’s created without any record definitions.

Use the SOAPDoc class if all of the following are true:

- Your third-party source or target node requires SOAP-compliant messages.
- Your third-party source or target node requires synchronous transactions.
- Your message conforms to the SOAP specification.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 42, "XmlDoc Class," page 1745

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, "Sending and Receiving Messages"

Data Type of a SOAPDoc Object

SOAPDoc objects are declared as type SOAPDoc. For example,

```
Local SOAPDoc &MyDoc;

Global SOAPDoc &SOAPDoc;
```

Scope of a SOAPDoc Object

A SOAPDoc object can only be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Note. Do not extend a SOAPDoc with an application class. This is currently not supported.

SOAPDoc Built-In Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSOAPDoc

SOAPDoc Class Methods

In addition to the methods listed here, most of the XmlDoc and XmlNode class methods can be used with a SOAPDoc object. The methods are discussed in alphabetical order.

See Also

Chapter 42, "XmlDoc Class," page 1745

AddBody

Syntax

AddBody()

Description

Use the AddBody method to add the body section of the SOAP XML document.

 Copyright © 1988-2007, Oracle. All rights reserved.

This method is optional when you're creating a SOAP document. Use it only if you must set any additional attributes for the body. Otherwise, the body section is automatically generated in the XML document.

If you're going to add a body section, you must do it *after* you've added the envelope section. If you are going to modify the body section, you must do this *before* you add the method sections of the SOAP XML document by calling the BodyNode property and using XmlNode methods and properties.

Note. Only one body section can be set within one Peoplesoft SOAP XML document.

Parameters

None.

Returns

None.

Example

From the following code:

```
&MyDoc.AddBody();
```

the following XML is created:

```
<SOAP-ENV:Body >

</SOAP-ENV:Body>
```

See Also

Chapter 37, "SOAPDoc Class," AddMethod, page 1552 and Chapter 37, "SOAPDoc Class," AddEnvelope, page 1547.

Chapter 42, "XmlDoc Class," AddAttribute, page 1765

AddEnvelope

Syntax

AddEnvelope(*SOAP_Schema*)

Description

Use the AddEnvelope method to add the envelope element tags and the default schema. The default encoding styles are also set with this method.

Note. Only one envelope section can be set within one Peoplesoft SOAP XML document.

If you're going to be adding an envelope section, you must do this *before* you add the body section or any methods by calling the EnvelopeNode property and using XmlNode methods and properties.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
SOAP_Schema	Specify whether to use the default SOAP schema or the UDDI schema. You can specify either a numeric value or a constant for this property. The values are:

Numeric Value	Constant Value	Description
0	%SOAP_Schema	Use the SOAP schema. Specifying this value adds the following attributes to the SOAP envelope: `xmlns:SOAP-ENV= "http:⇒ //schemas.xmlsoap.org/soap⇒ /envelope/" xmlns:SOAP-ENV= "http:⇒ //schemas.xmlsoap.org/soap⇒ /encoding/"`
1	%SOAP_UDDI	Use the UDDI schema. Specifying this value adds the following attribute to the SOAP envelope: `xmlns:SOAP-ENV= "urn:uddi-org:⇒ ⇒ ⇒ ⇒ api"`
2	%SOAP_Custom	Specify this value to add custom Namespace schemas to the SOAP envelope instead of the default SOAP schemas. **Note.** You must create the custom schema before you use it. See http://www.w3.org/XML/Schema You must add the SOAP_ENV URI (as an attribute, using AddAttribute) before adding any other SOAP attributes. **Note.** For highly customized SOAP documents, PeopleSoft recommends using the XmlDoc class and creating the tags yourself.

Returns

None.

Example

The following example uses the default SOAP schema. The following PeopleCode program:

```
&MyDoc.AddEnvelope(%SOAP_Schema);
```

Creates the following XML:

Copyright © 1988-2007, Oracle. All rights reserved.

```
<?xml version="1.0"?>

<SOAP-ENV:Envelope xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/encoding/">

</SOAP-ENV:Envelope>
```

The following example creates an envelope and validates it.

The following PeopleCode program:

```
&soapReq = CreateSOAPDoc(); /* required */
/* manually add SOAP XML envelope, header, body, method and parameters */
/* &soapReq.AddHeader(); MUST come before any other section, but is optional */
&soapReq.AddEnvelope(%SOAP_Custom);
&EnvNode = &soapReq.EnvelopeNode;
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:SOAP-ENV", "urn:MyCustom⇒
Schema.org");
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:xsi", http://www.w3.org/2001⇒
/XMLSchema-instance);
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:xsd", http://www.w3.org/2001⇒
/XMLSchema);
&AddEnvelopeAttribute = &EnvNode.AddAttribute("xmlns:soap", http:⇒
//schemas.xmlsoap.org/soap/envelope/);
&soapReq.AddMethod("requestAccessCode", 1);

/* Validate the message */
MessageBox(%MsgStyle_OK, "", 0, 0, "Validate the message");
&OK = &soapReq.ValidateSOAPDoc();
MessageBox(%MsgStyle_OK, "", 0, 0, "Message Validated and &OK = " | &OK);

/* Convert SOAP XML to XML */
MessageBox(%MsgStyle_OK, "", 0, 0, "Sending the message next");

&string = &soapReq.GenXmlString();
MessageBox(%MsgStyle_OK, "", 0, 0, "SOAP XML = " | &string);
```

Creates the following XML:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:xsd= "http://www.w3.org/2001/XMLSchema"
xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
xmlns:soap= "http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV= "urn:MyCustomSchema.org">
<SOAP-ENV:Body>
   <requestAccessCode/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 37, "SOAPDoc Class," AddBody, page 1546

AddFault

Syntax
AddFault(*Fault_Code, Fault_String*)

Description

Use the AddFault method to add a fault code, with its corresponding fault string, to a SOAPDoc. Use this method only in a Response SOAP XML Document. The Envelope and Body sections of the XML Document are automatically added if they don't exist.

Parameters

Parameter	Description
Fault_Code	Specify the fault code to add, as a string.
Fault_String	Specify the fault string to add, as a sting.

The following fault codes are supported:

Fault Code	Fault String	Description
100	Version Mismatch	The XML version or SOAP version is not supported by the current implementation.
300	Client error	An error occurred on the client while processing the SOAP document.
400	Server error	An error occurred on the server while processing the SOAP document.

Returns

None.

Example

The following example program:

```
&SOAPDoc = CreateSOAPDoc();
SOAPDoc.AddFault("400", "Server Error");


/* ===> The following statement executes this instance: */
&FaultCode = &SOAPDoc.FaultCode;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&FaultString = &SOAPDoc.FaultString;
&OK = &SOAPDoc.ValidateSOAPDoc();
```

Creates the following XML:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-⇒
⇒
⇒
⇒
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
- <SOAP-ENV:Body>
- <SOAP-ENV:Fault>
  <faultcode>400</faultcode>
  <faultstring>SOAP Server Error</faultstring>
  </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

See Also

Chapter 37, "SOAPDoc Class," FaultCode, page 1557, Chapter 37, "SOAPDoc Class," FaultString, page 1558 and Chapter 37, "SOAPDoc Class," FaultCodeS, page 1557.

AddHeader

Syntax

AddHeader()

Description

Use the AddHeader method to add the header section of the SOAP XML document.

This method is optional when you're creating a SOAP document. Use it only if you want to include a header in your SOAP XML document.

If you're going to add a header, you must do it *before* you've added any other sections, such as the envelope or the body.

After you've created the header, you can access it using the HeaderNode property. Once you have a reference to the node, you must use the XmlDoc methods to add specifics to the header section such as encription, as well as any required child nodes.

Note. Only one header section can be set within one Peoplesoft SOAP XML document.

Parameters

None.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following test adds a header section into a SOAP XML document where all sections are created explicitly:

```
Local SOAPDocs &Sdoc;
Local XMLNode &Node;

/* ===> Add inputs: */
&Sdoc = CreateSOAPDoc(); /* required */
&Sdoc.AddEnvelope(0); /* optional */
&Sdoc.AddHeader(); /* optional */
&Sdoc.AddBody(); /* optional */
&Sdoc.AddMethod("SOAPXml", 1); /* required */
&Sdoc.AddParm("symbols", "PSFT");

&OK = &Sdoc.ValidateSOAPDoc();

/*Append to TEMP file*/
&Xstring = &Sdoc.GenXmlString();
&myfile = GetFile("cmsDoc.xml", "A");
&myfile.WriteLine(&Xstring);
&myfile.Close();

&METHOD = &Sdoc.MethodName;
&ParmCnt = &Sdoc.ParmCount;
&HdrNode= &Sdoc.HeaderNode;

&HdrName = &HdrNode.NodeName; //returns SOAP-ENV:Header

/* Echo out the Returned Outputs */
out_BI_results("SOAP Header test 1");
out_BI_results(" ");
out_BI_results("Validated: " | &OK);
out_BI_results("Method: " | &METHOD);
out_BI_results("Parm Count: " | &ParmCount);
out_BI_results("Header : " | &HdrName);

out_BI_results("End-of-Test");
```

See Also

Chapter 37, “SOAPDoc Class,” HeaderNode, page 1558

AddMethod

Syntax

AddMethod(*MethodName*, *IsRequest*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AddMethod method to set the name of the method element being processed. Additional attributes for the added method can be set using the AddAttribute and InsertAttribute method. Executing AddMethod automatically adds any missing Envelope and Body sections.

Note. Only one method can be set within one Peoplesoft SOAP XML document.

The SOAPDoc method can be used only *after* the envelope and body sections have been added.

Set the parameters for the method using the AddParm method.

Parameters

Parameter	Description
MethodName	Specify the name of the method element being added.
IsRequest	Specify if the message is a request message. This parameter takes a numeric value: • 1 if the message is a request message • 0 if the message is a response message

Returns

None.

Example

The following sample program:

```
&MyDoc.AddMethod("GetPrice", 0);
```

Creates the following XML:

```
<GetPrice>

</GetPriceResponse>
```

The following sample program:

```
&MyDoc.AddMethod("GetPrice", 1);
```

Creates the following XML:

```
<GetPrice>

</GetPrice>
```

See Also

Chapter 37, “SOAPDoc Class,” AddParm, page 1554, Chapter 37, “SOAPDoc Class,” MethodName, page 1559 and Chapter 37, “SOAPDoc Class,” MethodNode, page 1559.

Chapter 42, “XmlDoc Class,” AddAttribute, page 1765

Copyright © 1988-2007, Oracle. All rights reserved.

AddParm

Syntax

AddParm(*ParmName*, *ParmValue*)

Description

Use the AddParm method to set the parameters for the parameter element of the current method. To set more than one parameter, use the method more than once. The parameters are set and used in the order that this method is called. Any number of parameter name-value pairs can be set for a method and are added to the XML document in the order that this method is called from the PeopleCode program.

Parameters

Parameter	Description
ParmName	Specify the name of the parameter as a string.
ParmValue	Specify the value of the parameter as a string.

Returns

None.

Example

The following PeopleCode:

```
&MyDoc.AddParm("Item", "Apples");
```

Creates the following XML:

```
<Item >Apples</Item>
```

See Also

Chapter 37, "SOAPDoc Class," AddMethod, page 1552

GetParmName

Syntax

GetParmName(*Index*)

Description

Use the GetParmName method access the parameter names for a method.

Parameters

Parameter	Description
Index	Specify the number of the parameter that you want to access. Values start at 1.

Returns

A string.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
For &I = 1 to &SOAPDoc.ParmCount

&ParmName = &SOAPDoc.GetParmName(&I);
&ParmValue = &SOAPDoc.GetParmValue(&I);

   /* Do processing */

End-For;
```

See Also

Chapter 37, “SOAPDoc Class,” GetParmValue, page 1555 and Chapter 37, “SOAPDoc Class,” ParmCount, page 1559.

GetParmValue

Syntax

GetParmValue(*Index*)

Description

Use the GetParmValue method to access the parameter values for a method.

Parameters

Parameter	Description
Index	Specify the number of the parameter you want to access. Values start at 1.

Returns

A string.

Example

```
For &I = 1 to &SOAPDoc.ParmCount

&ParmName = &SOAPDoc.GetParmName(&I);
&ParmValue = &SOAPDoc.GetParmValue(&I);

   /* Do processing */

End-For;
```

See Also

Chapter 37, “SOAPDoc Class,” GetParmName, page 1554 and Chapter 37, “SOAPDoc Class,” ParmCount, page 1559.

Copyright © 1988-2007, Oracle. All rights reserved.

ValidateSOAPDoc

Syntax

ValidateSOAPDoc()

Description

Use the ValidateSOAPDoc method to validate the SOAP document for correctness as follows:

SOAP message is:

- Encoded using XML.
- Has SOAP envelope.
- Has SOAP body.
- Used SOAP encoding and envelope namespaces.

Returns

This method returns either a number or a constant. The values are:

Numeric Value	Constant Value	Description
0	%SOAP_Valid	The SOAPDoc is valid.
1	%SOAP_NoEnvelope	Missing or invalid envelope section in XML document.
2	%SOAP_InvalidEnvelope	Envelope has improper Namespace used. Should be: SOAP-ENV or SOAP-ENC
3	%SOAP_NoBody	Missing body section in XML document.
4	%SOAP_NoMethod	Missing or invalid method section in XML document.
6	%SOAP_InvalidXml	Invalid XML syntax in SOAPDoc.

Example

```
&Return = &MyDoc.ValidateSOAPDoc();
If &Return <> 0 Then
   /* do error processing */
End-if;
```

See Also

Chapter 37, “SOAPDoc Class,” FaultCode, page 1557 and Chapter 37, “SOAPDoc Class,” FaultString, page 1558.

SOAPDoc Class Properties

In addition to the properties listed here, most of the XmlDoc and XmlNode class properties can be used with a SOAPDoc object. The properties are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 42, “XmlDoc Class,” page 1745

BodyNode

Description

This property returns a reference to the body section of a SOAPDoc, as an XmlNode.

This property is read-only.

See Also
Chapter 42, “XmlDoc Class,” page 1745

EnvelopeNode

Description

This property returns a reference to the envelope section of a SOAPDoc, as an XmlNode.

This property is read-only.

See Also
Chapter 42, “XmlDoc Class,” page 1745

FaultCode

Description

Use the FaultCode property to return the fault code if the SOAP message has one. Not every message has a fault code. It is returned only when the SOAP method has had an error in processing *and* the SOAP method supports fault codes.

This property is read-only.

Example

```
&Fault = &MyDoc.FaultCode;
```

See Also
Chapter 37, “SOAPDoc Class,” FaultString, page 1558, Chapter 37, “SOAPDoc Class,” FaultCodeS, page 1557 and Chapter 37, “SOAPDoc Class,” AddFault, page 1550.

FaultCodeS

Description

Use the FaultCodeS property to return the fault code string that was added using the AddFault method.

This property is read-write.

See Also
Chapter 37, “SOAPDoc Class,” AddFault, page 1550 , Chapter 37, “SOAPDoc Class,” FaultCode, page 1557 and Chapter 37, “SOAPDoc Class,” FaultString, page 1558.

Copyright © 1988-2007, Oracle. All rights reserved.

FaultString

Description

Use the FaultString property to return the fault string if the reply has one. Not every SOAP message has a fault string. It is only returned when the SOAP method has had an error in processing *and* the SOAP method supports fault strings.

Example

```
&Fault = &MyDoc.FaultString;
```

See Also

Chapter 37, "SOAPDoc Class," FaultCode, page 1557, Chapter 37, "SOAPDoc Class," FaultCodeS, page 1557 and Chapter 37, "SOAPDoc Class," AddFault, page 1550.

HeaderNode

Description

Use the HeaderNode property to return a reference to the header node. After you have a reference, you can use the XmlDoc methods to read and make changes to the node.

This property is read-write.

Example

The following test adds a header section, validates the SOAPDoc, then access the header.

```
Local SOAPDocs &Sdoc;
Local XMLNode &Node;

/* ===> Add inputs: */
&Sdoc = CreateSOAPDoc(); /* required */

&Sdoc.AddHeader(); /* optional */
&Sdoc.AddMethod("SOAPXml", 1); /* required */

&OK = &Sdoc.ValidateSOAPDoc();

/*Append to TEMP file*/
&Xstring = &Sdoc.GenXmlString();
&myfile = GetFile("cmsDoc.xml", "A");
&myfile.WriteLine(&Xstring);
&myfile.Close();

&METHOD = &Sdoc.MethodName;
&ParmCnt = &Sdoc.ParmCount;
&HdrNode= &Sdoc.HeaderNode; //returns SOAP-ENV:Header

&HdrName = &HdrNode.NodeName;

/* Echo out the Returned Outputs */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
out_BI_results("SOAP Header test 1");
out_BI_results(" ");
out_BI_results("Validated: " | &OK);
out_BI_results("Method: " | &METHOD);
out_BI_results("Parm Count: " | &ParmCount);
out_BI_results("Header : " | &HdrName);

out_BI_results("End-of-Test");
```

See Also

Chapter 37, “SOAPDoc Class,” AddHeader, page 1551

MethodName

Description

This property returns the name of the method of the SOAPDoc, as a string.

If you want a reference to the method that you can work with, as an XmlNode, use the MethodNode property.

This property is read-only.

MethodNode

Description

This property returns a reference to the method section of a SOAPDoc, as an XmlNode.

If you want only the name of the method returned, use the MethodName property.

This property is read-only.

See Also

Chapter 42, “XmlDoc Class,” page 1745

ParmCount

Description

This property returns the total number of parameters for the method section in a SOAP XML document as a number.

This property is read-only.

Example

```
For &I = 1 to &SOAPDoc.ParmCount
   &ParmName = &SOAPDoc.GetParmName(&I);
   &ParmValue = &SOAPDoc.GetParmValue(&I);
   /* do processing */
End-For;
```

Copyright © 1988-2007, Oracle. All rights reserved.

XmlDoc

Description

This property transforms a SOAPDoc object to an XmlDoc object. This is necessary only when sending or receiving a response message. You do *not* need to do this conversion to use the XmlDoc or XmlNode methods and properties.

This property is read-write.

Example

The following is an example of sending a SOAP message and receiving the response:

```
Local XmlDoc &request, &response;
Local string &strXml;
Local SOAPDoc &soapReq, &soapRes;

/* create the SOAP XML Document */
&soapReq = CreateSOAPDoc();
&soapReq.AddMethod("GetPrice");
&soapReq.AddParm("Item", "Apples");

/* convert SOAP to XmlDoc */
&request = &soapReq.XmlDoc;

/* Send the Request */
&response = SyncRequestXmlDoc(&request, Message.QE_SOAP_REQ, Node.UNDERDOG);

/* Get the SOAP response from the XmlDoc response */
&soapRes = CreateSOAPDoc();
&soapRes.XmlDoc = &response;
&OK = &soapRes.ValidateSOAPDoc();
&strXml = &soapRes.GenXmlString();
```

SOAPDoc Class Examples

The following are typical example of how you would use a SOAPDoc object.

Creating a SOAP Document

The following is the minimal PeopleCode you need to create a SOAP XML document:

```
Local SOAPDoc &SOAPDoc;

&SOAPDoc = CreateSOAPDoc();
&SOAPDoc.AddMethod("GetPrice");
&SOAPDoc.AddParm("Item", "Apples");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

The following is an example of creating a SOAP XML document with a simple method.

```
Local SOAPDoc &SOAPDoc;

&SOAPDoc = CreateSOAPDoc();

&SOAPDoc.AddEnvelope(0);
&SOAPDoc.AddBody();
&SOAPDoc.AddParm("symbols", "PSFT");
&SOAPDoc.AddParm("format", "l1c1d1t1");

&OK = &SOAPDoc.ValidateSOAPDoc();
```

Reading an Existing SOAP Document

The following code loads an SOAP XML document from a URL, then finds the method name and parameters to call that method.

```
Local SOAPDoc &SOAPDoc;
Local Number &ParmCount;
Local Boolean &Result;

&SOAPDoc = CreateSOAPDoc();
&SOAPDoc.ParseXmlFromURL("C:\ptdvl\840S2\files\inputSOAP.xml");

&ParmCount = &SOAPDoc.ParmCount;
For &I = 1 to &ParmCount

   &ParmName = &SOAPDoc.GetParmName(&I);
   &ParmValue = &SOAPDoc.GetParmValue(&I);

/* Do processing */
End-for;
```

Using SOAP XML Text

The following shows a SOAP XML Document text for the following pseudo-code:

```
Item = "Apples"
Price = GetPrice(Item);
```

Example of SOAP Request Header

Here's the HTTP Header for a SOAP message. This Host URL is the server where the method defined in the SOAP envelope is processed:

```
POST /GetPrice HTTP/1.1
Host: www.farmersmarket.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
SOAPAction:"GetPrice"
```

This is a simple example of a SOAP XML Document, requesting the price of Apples. It uses the default SOAP schemas. The name of the method is GetPrice with the parameter Item which has a value of "Apples".

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.SOAP.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.SOAP.org/soap/encoding/"
>
    <SOAP-ENV:Body>
        <m:GetPrice xmnls:m="SomeURI">
        <Item>Apples</Item>
        </m:GetPrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of SOAP HTML Reply Header

Here's the HTTP Header for a SOAP message Reply with a 200 return code that the message was processed OK.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
```

This simple example of a SOAP XML Document reply returns the price of Apples as 34. The SOAP reply always codes the name of the requested method with the Response tag attached to it, GetPriceResponse in this case.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.SOAP.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.SOAP.org/soap/encoding/"
>
   <SOAP-ENV:Body>
       <m:GetPriceResponse xmlns:m="Some-URI">
          <Price>34</Price>
       </m:GetPriceResponse>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 38

SQL Class

This chapter provides an overview of the SQL class and discusses the following topics:

- Using Record class SQL.
- Creating a SQL definition.
- Binding and executing of SQL statements.
- Fetching from a Select.
- Using Array of Any for bind values.
- Understanding different "Styles" of SELECT.
- Reusing a cursor.
- Understanding SQL objects and Application Engine programs.
- Declaring a SQL object.
- Scope of a SQL object.
- SQL class built-in functions.
- SQL class methods.
- SQL class properties.

See Also

Chapter 38, “SQL Class,” Understanding SQL Objects and PeopleSoft Application Engine Programs, page 1570

Chapter 31, “Record Class,” page 1377

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer’s Guide, “Accessing the Data Buffer”

Understanding SQL Class

You can create SQL definitions in PeopleSoft Application Designer. These can be entire SQL programs, or just fragments of SQL statements that you want to re-use. PeopleCode provides the SQL class for accessing these SQL definitions in your PeopleCode program at runtime.

The SQL class provides capability beyond that offered by SQLExec. Unlike SQLExec, which fetches just the first SELECTed row, operations for the SQL class allow iteration over all rows fetched. This can dramatically improve performance if you’re doing a million operations and you’ve set the BulkMode property to True.

A list of input (bind) values, and a list of output variables are supported, as they are in SQLExec. The input and output variables are limited to the same PeopleCode types that can be used with SQLExec, with the addition of a new class called Record.

Copyright © 1988-2007, Oracle. All rights reserved.

At runtime, you instantiate a record object from the Record class. A record object is a "one row" instantiation of a record definition.

- When used as an output from an SQL object, each next field in the record is populated with the next column returned.
- When used as an input, several fields in the record are used to replace a single bind marker in the SQL statement. The bind marker for a record describes the type of substitution to be done.

Both records and other PeopleCode types can be mixed in both the output and input.

At runtime, you instantiate a SQL object from the SQL class. The SQL object is loaded by either a constructor for the object, or an explicit Open method call. Optionally, a SQL constructor and the Open method support setting the SQL statement through a string parameter. This capability enables the creation and execution of ad-hoc SQL statements.

The SQL class has a Fetch method for iterating through the rows fetched by a select. A *cursor* is used to control this connection between the runtime SQL object and the database. The cursor is closed automatically when the object goes out of scope. The cursor can be closed before that by using the Close method. The status of the connection is available from the Boolean IsOpen property.

The general status of operations is available from function or method return values or properties. Detailed status is not available, as the operations are designed to terminate on unexpected errors or errors that cannot be reasonably recovered from by application level logic.

Considerations for Extra Spaces

On the DB2 UDB for OS/390 and z/OS platform, when the **zparm** option is set to decimal equals comma (so comma is used as the database decimal separator), extra blanks are added after commas to ensure that they are not mistaken for decimal separators.

Considerations for Case Sensitivity

When processing a SQL statement, the system automatically casts all fieldnames and possibly record names to uppercase when processing a SQL statement. When processing records from a third party, fields that are lowercase get cast into uppercase, which can create a runtime issue on case sensitive platforms.

To prevent this, use the %NoUppercase meta-SQL statement at the beginning of the SQL statement.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “Meta-SQL,” %NoUppercase

Using Record Class SQL

There are some SQL type operations that you can do with the Record class, such as INSERT, DELETE and UPDATE. The advantage of using the Record class methods is ease of use, re-use of code, and so on. However, if you’re doing many iterations of the same operation (like a million UPDATEs) you should use the SQL object with the BulkMode property set to True.

The SQL object maintains a state (that is, a cursor). Hence, if your database can take advantage of BulkMode, instead of a million operations, the commands are "bulked up" and committed only once. This can improve performance dramatically.

 Copyright © 1988-2007, Oracle. All rights reserved.

Creating a SQL Definition

You can create SQL definitions in PeopleSoft Application Designer, using the SQL editor. These SQL statements can be entire SQL programs, or just fragments that you want to re-use. After you have created a SQL definition, you can use it to populate a SQL object (using FetchSQL, Open, or GetSQL)

You can also create a SQL statement in PeopleCode (using CreateSQL), save it as a SQL definition (StoreSQL), then access it in PeopleSoft Application Designer.

See Also

Chapter 38, "SQL Class," Open, page 1576

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Using the SQL Editor"

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FetchSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," StoreSQL

Binding and Executing of SQL Statements

The processing of an SQL statement involves a series of steps.

1. The binding process is the replacement of (variable) input values in the statement, in places indicated by bind placeholders.

   The input values are substituted into the SQL statement in place of the bind placeholders. These placeholders have the form "*:number*", or "*%bindop*(*:number* [, *parm*]. . .)" where the *number* indicates which input value is to be substituted, and the *bindop* and *parm* strings indicate what meta-SQL binding function is to be performed.

   Note. There must be no spaces between the *bindop* and the left parenthesis.

   The following binding meta-SQL functions are used with record objects to substitute various forms of fieldnames and values into the SQL statement. The goal of these binding functions is to enable the writing of SQL and PeopleCode that can manipulate records without dependencies on the exact fields in the records.

   - %DTTM
   - %InsertValues
   - %KeyEqual
   - %KeyEqualNoEffDt
   - %List

Copyright © 1988-2007, Oracle. All rights reserved.

- %OldKeyEqual
- %Table
- %UpdatePairs

2. The execution of an SQL statement is the carrying out of the operation of the statement by the database engine.

   This view of SQL statement processing is actually simpler than what actually occurs. In actual practice, the binding occurs in two distinct phases. The database engine (or its supporting routines) is aware of only the latter phase. For some operations, some database engines are able to delay the second stage of binding and the execution of an SQL statement, so the statement can be rebound and re-executed (with different input values). The advantage of this is that these bindings and executions can be accumulated and transmitted across the network to the database server, with several database operations being done in one network operation. This is sometimes referred to as *bulk mode*. Because the network time consumes most database time operations, the performance advantages of bulk mode can be significant.

   However, in bulk mode, individual operations might not be executed immediately by the database engine. The result is that the application might not see errors that arise until later operations are performed.

   SQL SELECT statements are not bound multiple times, rather the retrieved rows are accumulated and sent across the network many rows at a time, also decreasing the effect of network delays.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "Meta-SQL"

Fetching From a Select

Some operations fetch a row of data into a list of output variables. The values of the fields from the select row are assigned in order to the given output variables. If one of these is a reference to a PeopleCode record object, the fields in the record are assigned the successive values from the row of the select, until all the fields are assigned. Assignment then continues with the next output variable, if any. The number of output fields and variables must equal the number of fields in the row of the select.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Accessing the Data Buffer"

Using Array of Any for Bind Values or Fetch Results

You can use a parameter of type "Array of Any" in place of a list of bind values or in place of a list of fetch result variables for the Execute, Fetch, and Open methods. This is generally used when you don't know how many values are needed until the code runs.

For example, suppose that you had a SQL definition INSERT_TEST, that dynamically (that is, at runtime) generated the following SQL statement:

```
"INSERT INTO PS_TESTREC (TESTF1, TESTF2, TESTF3, TESTF4, . . . N) VALUES (:1, :2,⇒
 %DateTimeIn(:3), %TextIn(:4)), . . . N";
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Suppose you have placed the values to be inserted into an Array of Any, say &AAny:

```
&AAny = CreateArrayAny("a", 1, %DateTime, "abcdefg", . . . N);
```

You can execute the insert with the following:

```
&AAny = CreateArrayAny("a", 1, %DateTime, "abcdefg", . . . N);
&SQL = GetSQL(SQL.INSERT_TEST);
While /* Still something to do */
   &SQL.Execute(&AAny);
End-While;
&SQL.Close();
```

Because the Array of Any promotes to absorb any remaining select columns, it must be the last parameter for the SQL object Fetch method or (for results) SQLExec. For binding, it must be the only bind parameter, as it is expected to supply all the bind values needed.

Understanding Different "Styles" of SELECT

Using the SQL object, you have different methods of using a SELECT statement in your code, depending on your application.

The most direct way is with an Open and Fetch. This fetches successive rows, and you can process each row after it's fetched.

The pseudo code for this type of select statement is:

```
&SQL.Open("Select .  .  .", bindpars);
While &SQL.Fetch(resultpars)
   /* process result row */
End-While;
```

Another method is to set up the loop based on something you want to change, and while you still want to change things, continue the loop.

The pseudo code for this type of select statement is:

```
&SQL.Open("Update/Insert/Delete/other.  .  .  :n .  .  .");
/* Bind references, but no bind parameters */
While you_want_to_do_changes
   /* set up for changes */
   &SQL.Execute(bindpars);
End-While;
```

Reusing a Cursor

Reusing a cursor means *compiling* a SQL statement just once, but *executing* it more than once. When a SQL statement is compiled, the database checks the syntax and chooses a query path. By doing this only once, but executing this statement several times, you can obtain an improvement in performance.

Copyright © 1988-2007, Oracle. All rights reserved.

In PeopleCode, you can reuse a cursor either by using the ReuseCursor property or by following certain restrictions in your code. In this section, we discuss how to:

- Use the ReuseCursor property.
- Reuse a cursor without the ReuseCursor property.

Using the ReuseCursor Property

If you specify the ReuseCursor property as True, the cursor isn't closed until either the SQL object is explicitly closed or re-opened. This provides greater control over the cursor associated with your SQL object. However, when you set ReuseCursor to True, you're essentially pledging to do the right thing in your program. There are some considerations for how you use this property:

In a PeopleSoft Application Engine program, if your program can be restarted, you *must*check for a restart after a checkpoint by testing to see if the SQL object is open after the checkpoint. If it isn't open, that means a restart has happened, and you must reopen the SQL object. In most cases, on checkpoints, your open SQL objects aren't closed, saving the overhead of re-establishing the SQL object after the checkpoint. If the SQL object is open on a select statement at a checkpoint it isn't restored to the open state because you cannot reliably establish the state of the execute-fetch sequence.

In the following example the SQL object is established on a select statement which is executed twice with different bind parameters but is compiled only once. Without the ReuseCursor property the SQL object would be closed after the first fetch cycle completes.

```
Local SQL &Sql;
Local String &Company1 = "CCB";
Local String &Company2 = "CCF";

&Sql = CreateSQL("SELECT A.COMPANY, %Timeout(A.EFFDT) FROM %Table(COMPANY_TBL) A⇒
 WHERE A.COMPANY=:1", &Company1);
/* commenting this out should make the subsequent Execute fail */
&Sql.ReuseCursor = True;

While &Sql.Fetch(&Company1, &Effdt)
   /* processing for the first Company */

End-While;

&Sql.Execute(&Company2);

While &Sql.Fetch(&Company1, &Effdt)
/* processing for the second company */

End-While;

&Sql.Close();
```

Reusing a Cursor Without the ReuseCursor Property

If you want to reuse a cursor in your program, there are several conditions:

 Copyright © 1988-2007, Oracle. All rights reserved.

- You have to use the SQL object. Only SQL objects retain SQL cursor information.
- For INSERT, DELETE, and UPDATE statements, you automatically reuse the cursor as long as you don't change the SQL statement as part of the binding process. If the SQL statement is textually the same, so only the binds have changed, you get reuse. For example, you won't get reuse if you first bind one kind of record object to %Insert(), then bind another, different kind of record object.
- For SELECTs, you *must*use the meta-SQL shortcuts (%SelectAll, %SelectByKey, or %SelectByKeyEffDt), *and* the CreateSQL or GetSQL functions without supplying any bind or buffer parameters. The bind parameters are supplied in the Execute function. The Fetch parameter must be the fetch buffer, and be the same as the first Execute parameter. You can supply WHERE clauses, ORDER-BY, and so on, on the end of the SQL string containing the meta-SQL.

Note. BulkMode doesn't reuse the cursor in the same way as the SQL object. BulkMode requests that, when the system can reuse the cursor, it also holds all the changes so they can be communicated to the database in batches.

By using one of the forms of the Select meta-SQL, you're guaranteeing the resulting fetched values are all put into one record object (buffer). This means the implementation doesn't have to ask the database for the length and type of each column: the record buffer is already defined. So this sample code uses the SQL object to maintain the state of the connection with the database, and the record object, to maintain a series of fields suitable for database operations.

```
Local SQL &SQL;
Local Record &REC;

&REC = CreateRecord(Record.KP_KPI_DFN);

/* start with select statement, no bind refs, no */
/* bind parameters */

&SQL = CreateSQL("%Selectall(:1) Where SETID = :2 and KPI_ID = :3 and EFFDT = ⇒
(SELECT MAX(EFFDT) FROM PS_KP_KPI_DFN WHERE SETID = :4 AND KPI_ID = :5 AND EFFDT <=⇒
⇒
⇒
⇒
 %DateIn(:6))");
```

Start Loop

```
/* bind and execute the statement */

&SQL.Execute(&REC, &SETID_KPI, &COMPID, &SETID_KPI, &COMPID, &EFFDT);

/* Note record parameter for Fetch statement must be
the same as the first Execute parameter
the results are in this record */

If &SQL.Fetch(&REC) Then

   /*process this record */

End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

End Loop

```
&SQL.Close(); /* there is no implicit close on a Fetch returning False */
```

The following example comes from a PeopleSoft Application Engine program. Because the loop goes in and out of PeopleCode, you must declare the SQL object as Global. This example is in three parts.

1. Program called by Application Engine before the loop:

```
Global SQL &SQL;

&SQL = CreateSQL("INSERT INTO %Table(MY_WORK) (TRANS , REGISTERWRITE) VALUES (:1, ⇒
:2)");

&SQL.BulkMode = True; /* not required for reuse, but will get better performance⇒
 on platforms that support bulk insert */
```

2. Program called in the Application Engine loop on SELECT 'X' FROM PSRECDEFN WHERE RECNAME LIKE '%A':

```
Global SQL &SQL;

&var1 = "X";
&var2 = "Y";

&SQL.Execute(&var1, &var2);
```

3. Program called by Application Engine after the loop:

```
Global SQL &SQL;

&SQL.Close();
```

Understanding SQL Objects and PeopleSoft Application Engine Programs

A global variable won't go out of scope until a PeopleSoft Application Engine program finishes. However, all SQL objects are forced closed (that is, the cursor closed) sooner than that. *Any open SQL object is forced closed just before any checkpoint in an Application Engine program.*This is to ensure that the application can be restarted successfully from the checkpoint. After the SQL object is closed, you can reopen the SQL object, or query its properties, (such as Status, IsOpen). The simplest way to avoid unnecessary closing of the SQL object is to set the ReuseCursor property to True. A restartable program should always check that SQL objects are open before using them in steps where it's expected they're open. In the absence of an intervening checkpoint, an open SQL object remains open until the Application Engine program finishes.

 Copyright © 1988-2007, Oracle. All rights reserved.

Declaring a SQL Object

SQL objects are declared as type SQL. For example:

```
Local SQL &MYSQL;

Global SQL &MySql = CreateSQL(SQL.MySql);
```

Scope of an SQL Object

An SQL object can be instantiated only from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

Your SQL statements sometimes change the database. When your SQL changes the database, your code should be only in one of the following events:

- SavePreChange
- WorkFlow
- SavePostChange
- Message Subscription
- FieldChange
- Application Engine PeopleCode action

SQL Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," DeleteSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," FetchSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," StoreSQL

Copyright © 1988-2007, Oracle. All rights reserved.

SQL Class Methods

In this section, we discuss the SQL class methods. The methods are discussed in alphabetical order.

Close

Syntax

```
Close()
```

Description

The Close method closes the SQL object. This terminates any incomplete fetching of select result rows, completes any buffered operations (that is, using BulkMode), and disassociates the SQL object from any SQL statement that was open on it.

After BulkMode operations, the RowsAffected property is not valid.

Parameters

None

Returns

True on successful completion, False if there was a duplicate record error. Any errors associated with buffered operations (that is, using BulkMode), other than duplicate record errors, cause termination.

Example

```
&SQL = CreateSQL("%Delete(:1)");
While /* Still something to do */
   /* Set key field values of &ABS_HIST */
   &SQL.Execute(&ABS_HIST);
End-While;
&SQL.Close();
```

See Also

Chapter 38, “SQL Class,” Open, page 1576

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, “PeopleCode Built-in Functions,” CreateSQL

Execute

Syntax

```
Execute(paramlist)
```

Where *paramlist* is an arbitrary-length list of values in the form:

```
inval1 [, inval2] ...
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Execute method executes the SQL statement of the SQL object. The SQL object must be open and unbound on a delete, insert, or update statement. That is, the CreateSQL, GetSQL, or Open preceding the Execute must have specified a delete, insert, or update statement with bind placeholders and must not have supplied any input values.

The values in *paramlist* are used to bind the SQL statement before it gets executed.

When using the optional BulkMode, the Execute operations may be buffered and are not guaranteed to have been presented to the database until a Close is done. Thus, in BulkMode, errors that arise may not be reported until later operations are done.

Parameters

Parameter	Description
paramlist	Specify input values for the SQL string.

Returns

True on successful completion, False for "record not found" and "duplicate record" errors. Any other errors cause termination.

Example

The following example creates a SQL object for inserting. The statement isn't automatically executed when it's created because there aren't any bind variables. The Execute occurs after other processing is finished. The name of the record is passed in as the bind variable in the Execute method.

```
&SQL = CreateSQL("%Insert(:1)");
While /* Still something to do */
   /* Set all the field values of &ABS_HIST.  */
   &SQL.Execute(&ABS_HIST);
End-While;
&SQL.Close();
```

The following example creates two SQL objects, one to be used for fetching, the other for updating the record. The first SQL object selects all the records in the &ABS_HIST record that match &EMPLID. The data is actually retrieved using the Fetch method. After values are set on the record, the update is performed by the Execute.

```
&SQL1 = CreateSQL("%Select(:1) where EMPLID = :2", &ABS_HIST, &EMPLID);
&SQL_UP = CreateSQL("%Update(:1)");
While &SQL1.Fetch(&ABS_HIST);
   /* Set some field values of &ABS_HIST.  */
   &SQL_UP.Execute(&ABS_HIST);
End-While;
&SQL_UP.Close();
```

The following is an example of inserting an array of records:

```
Local SQL &SQL;
Local array of Record &RECS;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
/* Set up the array of records.  */
.  .  .

/* Create the SQL object open on an insert */
/* statement, and unbound */
&SQL = CreateSQL("%Insert(:1)");
   /* While the array has something in it */
   While &RECS.Len
   /* Insert the first record of the array, */
   /* and remove it from the array.  */
   &SQL.Execute(&RECS.Shift());
   End-While;
```

See Also

Chapter 38, "SQL Class," Close, page 1572 and Chapter 38, "SQL Class," BulkMode, page 1577.

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL

Fetch

Syntax

Fetch(*paramlist*)

Where *paramlist* is an arbitrary-length list of values in the form:

outvar1 [, *outvar2*] ...

Description

The Fetch method retrieves the next row of data from the SELECT that is open on the SQL object. Any errors result in termination of the PeopleCode program with an error message.

If there are no more rows to fetch, Fetch returns as False, the *outvars* are set to their default PeopleCode values, and the SQL object is automatically closed.

Using Fetch with a closed SQL object is processed the same as when there are no more rows to fetch.

Note. If you want to fetch only a single row, the SQLExec function can perform better, as it fetches only a single row from the server.

The return of Fetch is not optional, that is, you *must* check for the value of the fetch.

Setting Data Fields to Null

This method will *not* set Component Processor data buffer fields to NULL after a row not found fetching error. However, it does set fields that aren't part of the Component Processor data buffers to NULL. It does set work record fields to NULL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
paramlist	Specify output variables from the SQL Select statement.

Returns

The result of Fetch is True if a row was fetched. If there are no more rows to fetch, the result is False.

Example

In the following example, the Fetch method is used first to process a single row, then to process the ABS_HIST record.

```
Local SQL &SQL;
Local Record &ABS_HIST;

&ABS_HIST = CreateRecord(RECORD.ABSENCE_HIST);
&SQL = GetSQL(SQL.SEL27, 15, "Smith");
While &SQL.Fetch(&NAME1, &BIRTH_DT)
   /* Process NAME1, BIRTHDT from the selected row.  */
End-While;

&SQL.Open(SQL.SEL_ABS_HIST, &NAME1, "Smith");
While &SQL.Fetch(&ABS_HIST)
   /* Process ABS_HIST record.  */
```

The following is an example of reading in an array of record objects:

```
Local SQL &SQL;
Local Record &REC;
Local Array of Record &RECS;

/* Get the SQL object open and ready for fetches.  */
&SQL = CreateSQL("%SelectAll(:1) where EMPLID = :2", RECORD.ABSENCE_HIST, &EMPLID);
/* Create the first record.  */
&REC = CreateRecord(RECORD.ABSENCE_HIST);
/* Create an empty array of records.  *
&RECS = CreateArrayRept(&REC, 0);
While &SQL.Fetch(&REC)
   /* We got a record, add it to the array */
   /* and create another.*/
   &RECS.Push(&REC);
   &REC = CreateRecord(RECORD.ABSENCE_HIST);
End-While;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 38, "SQL Class," Open, page 1576

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetSQL

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," SQLExec

Open

Syntax

```
Open(sql [, paramlist])
```

Where *paramlist* is an arbitrary-length list of values in the form:

```
inval1 [, inval2] ...
```

Description

The Open method associates the *sql* statement with the SQL object. The *sql* parameter can be either:

- A string value giving the SQL statement.
- A reference to a SQL definition in the form **SQL**.*sqlname*.

If the SQL object was already open, it is first closed. This terminates any incomplete fetching of select result rows, completes any buffered operations (that is, using BulkMode), and disassociates the SQL object from any SQL statement that was open on it.

Opening and Processing sql

If *sql* is a SELECT statement, it is immediately bound with the *inval* input values and executed. The SQL object should subsequently be the subject of a series of Fetch method calls to retrieve the selected rows. If you want to fetch only a single row, use the SQLExec function instead. If you want to fetch a single row into a PeopleCode record object, use the record Select method.

If *sql* is not a SELECT statement, and either: there are some inval parameters, or there are no bind placeholders in the SQL statement, the statement is immediately bound and executed. This means that there is nothing further to be done with the SQL statement and the IsOpen property of the returned SQL object will be False. In this case, using the SQLExec function would generally be more effective. If you want to delete, insert, or update a record object, use the record Delete, Insert, or Update methods with the record object.

If *sql* is not a SELECT statement, there are no *inval* parameters, *and* there are bind placeholders in the SQL statement, the statement is neither bound nor executed. The resulting SQL object should subsequently be the subject of a series of Execute method calls to affect the desired rows.

Setting Data Fields to Null

This method will *not* set Component Processor data buffer fields to NULL after a row not found fetching error. However, it does set fields that aren't part of the Component Processor data buffers to NULL.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
sql	Specify either a SQL string or a reference to a SQL definition in the form **SQL.** *sqlname*.
paramlist	Specify input values for the SQL string.

Returns

None.

Example

Generally, you use the Open method only after you've already gotten a reference to another SQL object. SELECT and SEL_ABS_HIST are the names of the SQL definitions created in PeopleSoft Application Designer.

```
Local SQL &SQL;

&SQL = GetSQL(SQL.SELECT);

/* do other processing */

/* get next SQL statement for additional processing */
/* The open automatically closes the previous */
/* SQL statement */

&SQL.Open(SQL.SEL_ABS_HIST, &NAME1, "Smith");
While &SQL.Fetch(&ABS_HIST)
   /* Process ABS_HIST record.  */
End-While;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSQL and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," GetSQL.

SQL Class Properties

In this section, we discuss the SQL class properties. The properties are discussed in alphabetical order.

BulkMode

Description

This property controls the use of bulk mode. Setting this property to True enables the use of bulk mode, and hence removes any guarantee of the synchronous presentation of error status.

Copyright © 1988-2007, Oracle. All rights reserved.

Bulk mode is used only with those database connections and operations that support it. Bulk mode can be used with any SQL operation, that is, with INSERTs, DELETEs, or UPDATEs.

If you're using a PeopleSoft Application Engine program, and have set this property to True, the rows inserted in BulkMode are committed at the next database commit in your program.

After BulkMode operations, the RowsAffected property is not valid.

The default value for BulkMode is False.

This property is read-write.

Example

The following code is an example of inserting an array of records using bulk mode:

```
Local SQL &SQL;
Local array of Record &RECS;

/* Set up the array of records.  */
   .  .  .
   /* Create the SQL object open on an insert */
   /* statement, and unbound.*/
   &SQL = CreateSQL("%Insert(:1)");
   /* Try for bulk mode.  */
   &SQL.BulkMode = True;
   /* While the array has something in it&mldr; */
   While &RECS.Len
      /* Insert the first record of the array, */
      /* and remove it from the array.  */
      If not &SQL.Execute(&RECS.Shift) then
         /* A duplicate record found, possibly */
         /* in bulk mode.  There is no way to  */
         /* tell which record had the problem. */
         /* One approach to recovery is to fail*/
         /* the transaction and retry it with a*/
         /* process that does only one record  */
         /* at a time, that is, doesn&rsquo;t use    */
         /* bulk mode.*/
         .  .  .;
      End-If;
End-While;
```

See Also

Chapter 38, "SQL Class," RowsAffected, page 1580

Chapter 38, "SQL Class," Binding and Executing of SQL Statements, page 1565

 Copyright © 1988-2007, Oracle. All rights reserved.

IsOpen

Description

This property returns as True if the SQL object is open on some SQL statement.

This property is read-only.

Example

You might use the following in a PeopleSoft Application Engine program, after a checkpoint. MYSELECT is the name of a SQL definition created in PeopleSoft Application Designer:

```
If Not &MYSQL.IsOpen Then
   &MYSQL.Open(SQL.MYSELECT);
End-if;
```

LTrim

Description

This property specifies whether values read by the Fetch method are trimmed of blanks on the left, except in the following cases:

- For long columns.
- The record is directly used to buffer the incoming data. In the following example, the values aren't LTrimmed - regardless of the setting of the LTrim property.

```
Local Record &Rec = CreateRecord(Record.QA_TEST);
Local SQL &Sql = CreateSQL("%SelectAll(:1)");
&Sql.LTrim = True; /* the default */
&Sql.Execute(&Rec);
While &Sql.Fetch(&Rec)
   /* do processing */
End-While;
```

This property takes a Boolean value. The default value is True. If this property is set to False, the selected values are not trimmed of blanks on the left.

This property is read-write.

Note. The removal of blanks from the right end of fetched values (RTrimming) still occurs for non-long columns.

Example

```
Local Record &Rec = CreateRecord(Record.QA_TEST);
Local SQL &Sql = CreateSQL(%SelectAll(:1)", &Rec);
&Sql.LTrim = False;
While &Sql.Fetch(&Rec)
   /* do processing */
End-While;
```

Copyright © 1988-2007, Oracle. All rights reserved.

ReuseCursor

Description

This property specifies whether the SQL object tries to reuse the open cursor. This property takes a Boolean value, True, to reuse the SQL cursor.

If specified as True, the SQL object won't be closed at checkpoints, and any non-SELECT SQL is restored. In addition the SQL object is not closed after a fetch cycle completes. It remains open ready to be executed, perhaps with different bind parameters.

If you use this property in your application program, you *must* close the SQL object explicitly or it is closed when the object goes out of scope (that is, when the program finishes.)

You must instantiate a SQL object first before you can reuse it.

This property is read-write.

See Also

Chapter 38, "SQL Class," Reusing a Cursor, page 1567

RowsAffected

Description

This property returns the number of rows affected by the last INSERT, UPDATE, or DELETE statement of the SQL object. After BulkMode operations, the RowsAffected property is not valid.

This property is read-only.

Example

The following code is an example that determines if a delete statement actually deleted anything:

```
Local SQL &SQL;

/* Create the SQL object and do the deletion.  */
   &SQL = CreateSQL("Delete from %Table(:1) where EMPLID = :1", RECORD.ABSENCE_⇒
HIST, &EMPLID);
   If &SQL.RowsAffected = 0 Then
   /* We did not delete any rows.  */
   .  .  .
End-If;
```

See Also

Chapter 38, "SQL Class," BulkMode, page 1577

 Copyright © 1988-2007, Oracle. All rights reserved.

Status

Description

This property returns the status of the last statement executed. You can use either the constant or the numeric value for this property. The values for this property are:

Numeric Value	Constant Value	Description
0	%SQLStatus_OK	No Errors.
1	%SQLStatus_NotFound	Record not found.
2	%SQLStatus_Duplicate	Duplicate Record Found

This property is read-only.

Example

The following example determines what went wrong after an update:

```
Local SQL &SQL;
Local Record &NEWREC, &OLDREC;

/* Create and initialize &OLDREC with the keys of the
record to be updated.  Create and initialize &NEWREC
with the new field values for the record.  */
...;

/* Create and execute the update.  */
&SQL = CreateSQL("%Update(:1, :2)", &NEWREC, &OLDREC);
Evaluate &SQL.Status
When = %SQLStatus_OK
   /* It worked.  */
When = %SQLStatus_NotFound
   /* The OLDREC keys were not found.  */
When = %SQLStatus_Duplicate
   /* The NEWREC keys were already there.  */
End-Evaluate;
```

TraceName

Description

This property enables you to assign a name to a SQL statement that has been created in PeopleCode using CreateSQL. This name is used in the PeopleSoft Application Engine timings trace. This property takes a string value.

Note. You *cannot* associate the TraceName property with the execution of a simple SELECT statement created with CreateSQL. This is because the SELECT is executed when the SQL is created, before it has the TraceName assigned. To do this, create a SQL object instead.

Copyright © 1988-2007, Oracle. All rights reserved.

If this property isn't set, it defaults to a substring of the SQL statement, indicating the operation and table, (for example, SELECT PS_VOUCHER_LINE.) It may be useful to set TraceName to indicate the origin of the SQL statement.

This property is read-write.

Example

```
&REC = CreateRecord(Record.VOUCHER_LINE);
&SQL = CreateSQL("%selectall(:1) WHERE BUSINESS_UNIT =:2 AND VOUCHER_ID =:3 AND⇒
 VOUCHER_LINE_NUM = :4");
&SQL.TraceName = "AEPROG.SECT1.STEP1.SQL2";
&SQL.Execute(&REC, MATCHING_AET.BUSINESS_UNIT, MATCHING_AET.VOUCHER_ID, &count);
If &SQL.Fetch(&REC) Then
      &count2 = &count2 + 1;
End-If;
```

The previous example would produce the following in the timings trace.

```
SQL Statement              Count   Time     Count   Time     Count   Time     Time
-------------------------- ------- -------- ------- -------- ------- -------- --------

PeopleCode

AEPROG1.SECT1.STEP1.SQL1                      169    5.371     169    3.083    4.000
AEPROG1.SECT1.STEP1.SQL2                      100    5.371     100    3.083    4.454
                                                                             --------
                                                                                8.454
```

Trace Timings Example

You can use this parameter with the Open statement also. The following is an example of how this works:

```
&Sql1 = CreateObject("SQL");
&Sql1.TraceName = "sql1";
&Sql1.Open("Select %FirstRows(1) 'x' FROM psstatus");
&Sql1.Fetch(&Temp);

&Sql2 = CreateObject("SQL");
&Sql2.TraceName = "sql2";
&Sql2.Open("Select 'x' FROM psstatus");
&Sql2.Fetch(&Temp);
```

Value

Description

This property returns the SQL statement associated with the SQL object as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

To report an error in a SQL definition, including the actual SQL executed, use the Value property to get the text of the SQL statement:

```
Local SQL &SQL;

/* Execute some SQL.  */
&SQL = CreateSQL(SQL.SOMESQL, &EMPLID);
If &SQL.Status = %SQLStatus.NotFound Then
   /* Get the SQL string used.  */
   &SQLSTR = &SQL.Value;
   /* Report the error.  */
   .  .  .;
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 39

SyncServer Class

This chapter provides an overview of the synchronization server and the SyncServer classes and discusses the following topics:

- Understanding the synchronization server events and SyncServer classes
- Using synchronization server events
- Using attachment events
- Scope of a SyncServer object
- SyncServer reference
- Advanced considerations for synchronization

Understanding Synchronization Server Events and the SyncServer Classes

PeopleSoft Mobile Agent extends the functionality of PeopleSoft Pure Internet Architecture to disconnected mobile devices, allowing users to continue working with their PeopleSoft applications on a laptop computer or personal digital assistant (PDA) while disconnected from the internet or local network.

The mobile events control the PeopleCode programs used with the mobile device. This PeopleCode uses the mobile classes to control the changes an end user makes to the mobile data.

See *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Mobile Agent*.

The synchronization server events control the synchronization of mobile data with the PeopleSoft database. The SyncServer class methods and properties are used for data selection and validation during synchronization.

Understanding the Synchronization Events

When the mobile device is synchronized with the PeopleSoft database, the following events occur on the synchronization server. All of these events only fire when any PeopleCode is associated with the event.

Copyright © 1988-2007, Oracle. All rights reserved.

Event Name	Description
OnConflict	This event only occurs when the conflict resolution for the synchronizable component interface is selected as Custom on the properties dialog box for the definition, on the Synchronization tab.
OnGetDefinition	This event is used for defining Component Interface objects accessed only by a PeopleCode reference.
OnSelect	This event is used for specifying what data should be downloaded (distributed) to the mobile device.
OnValidate	Occurs when any PeopleCode is associated with the event *and* no PeopleCode is associated with the OnValidateSet event. This event is only used for data validation of a medium to large number of objects.
OnValidateSet	This event is used for data validation of a very large number of objects.

In addition, if you create references in the synchronizable Component Interface to be used with attachments, the following events are also available:

Event Name	Description
OnGetProperty	This event is used to specify how attachments are accessed and sent to the mobile device.
OnGetPropertyFilter	Occurs before the OnGetProperty event. This event is used for specifying filter rules to an individual attachment.
OnSetProperty	Occurs when the value of an attachment has changed on the mobile device. This event can is to specify which attachments are synchronized from the mobile device to the PeopleSoft database.

Many of the SyncServer class methods and properties are only valid in specific events. The following table lists the event restrictions for the methods and properties. If a method or property is not listed, it does not have restrictions about which event it can be used in.

Method or Property	Valid Events
AddReference	Only valid in the PreBuild, PostBuild, and RowInit events.
AddReferenceType	Only valid in the OnGetDefinition event.

 Copyright © 1988-2007, Oracle. All rights reserved.

Method or Property	Valid Events
CheckForChanges	No restrictions, but generally used in the OnSelect event.
SelectAll	No restrictions, but generally used in the OnSelect event.
ConflictAlgorithm	This property is checked after any PeopleCode in the OnConflict event has run.
ConflictStatus	Only valid in the ConflictStatus event.
ExcludeProperty	No event restrictions, but only used with attachments.
ExportLocation	Only valid in the OnGetProperty event.
LastSyncDateTime	Only valid in the OnSelect event.
LastSyncRowCount	Only valid in the OnSelect event.
PropertyValue	Only valid in the OnGetProperty and OnSetProperty events.
SessionID	Only valid in the OnValidateSet event.
SyncIDs	Only valid in the OnSelect event.
TypeID	Only valid in the OnValidateSet event.
validateID	Only valid in the OnValidate event.
validateRowCount	Only valid in the OnValidate event.
validateVersion	Only valid in the OnValidate event.

See Also

Chapter 39, "SyncServer Class," Using the Synchronization Server Events, page 1589

Understanding the SyncServer Class

During synchronization, use the %SyncServer system variable to instantiate and populate a SyncServer object. You can then use this object in one of the synchronization events to specify the details of the current synchronization.

The %SyncServer variable is ignored outside of the synchronization server events, or events called by the synchronization server.

PeopleSoft recommends using the Synchronizing property in events outside of the synchronization server events to separate code that should only be executed during synchronization. For example, suppose you had additional business logic to apply to data coming from a mobile device. You could put the following in one of the Component events that gets called during synchronization, such as SavePreChange.

```
If %SyncServer.Synchronizing Then
   /* more data rules here */
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

Considerations Using Synchronization Events and the SyncServer Class

The synchronization events are not required for synchronizing a mobile device. You should only start using the SyncServer class methods and properties in the events after your mobile application is running. The following is the typical development cycle for synchronization:

1. Create and test your component.

   If you are using an existing component, you must still test it to verify that it runs without errors and returns the data you are expecting.

2. Create and test your Component Interface.

   If you are using an existing Component Interface, you must still test it to verify that it runs without errors and returns the data you are expecting.

3. Create and test your mobile pages.

   Once your Component Interface is running, *then and only then* should you make the Component Interface synchronizable and build mobile pages based on it.

   Note. Remember that any changes made to the underlying Component Interface are *not* automatically reflected in any mobile pages built from that Component Interface.

   At this point, you should verify whether the search page associated with the component returns the data you need for your mobile application. If it does, and if you are not dealing with a large number of instances of the Component Interface that need to be synchronized, you are finished. There is no need to go any further and create PeopleCode for synchronization.

4. Refine the conflict resolution for your mobile application.

   The properties for the synchronizable Component Interface specify which version of the data should be used if there is a conflict (device always wins, database always wins, and so on.) Use the OnConflict event to refine the conflict rules if necessary.

5. Refine the data used with your mobile application.

   If the search record associated with the component is not adequate for returning the data you need for your mobile application, create PeopleCode associated with the OnSelect event to better define the data for your mobile device.

   Note. Remember, this event fires when *any* PeopleCode is associated with the event, even if the PeopleCode program is commented out.

   ---

6. Fine tune your mobile application.

   If your mobile application requires a medium to large number of objects, between 100–1000 instances of a Component Interface, you can use the OnValidate event to possibly enhance synchronization performance.

   If your mobile applications requires a large number of objects, over 1000 instances of a Component Interface, you can use the OnValidateSet event to possibly enhance synchronization performance.

   If you need to create references to Component Interface objects in PeopleCode, without defining the relationship in Application Designer, you can use the OnGetDefinition event.

 Copyright © 1988-2007, Oracle. All rights reserved.

Using the Synchronization Server Events

PeopleSoft Mobile includes synchronization server events to support mobile device synchronization and data distribution. If a component interface is synchronizable, these events exist in its associated PeopleCode.

This section discusses how to use:

- OnConflict
- OnSelect
- OnValidate

See Also

Chapter 25, "Optimization PeopleCode," page 915

Chapter 39, "SyncServer Class," Advanced Considerations for Synchronization, page 1605

Using OnConflict

The OnConflict event is invoked only when the conflict resolution for the synchronizable component interface is selected as Custom on the properties dialog box for the definition, on the Synchronization tab, and a conflict has occurred.

Property conflicts are generated when the property changed on both the server and the device, and the property value on the server is different than the property value on the device.

OnConflict PeopleCode is run for each conflict to decide whether the device update is accepted or rejected. The OnConflict PeopleCode attached to the level zero object of the Component Interface, if any, is always run after all other OnConflict PeopleCode has run. OnConflict PeopleCode can set the ConflictStatus property on the SyncServer object to %CONFLICTSTATUS_ NOCONFLICT, which causes the conflict to be ignored.

OnConflict PeopleCode can set the ConflictAlgorithm property on the SyncServer object to %CONFLICTALGORITHM_SERVERWINS or %CONFLICTALGORITHM_DEVICEWINS, which causes the device update to be rejected or accepted.

The ConflictAlgorithm property is checked after all OnConflict PeopleCode has run. The ConflictAlgorithm property overrides the ConflictStatus property. If the ConflictAlgorithm property is not set and if the ConflictStatus property is set to %CONFLICTSTATUS_NOCONFLICT for all of the conflicts, the device update is accepted, otherwise the device update is rejected.

Note. If a PeopleCode program exists in OnConflict but is commented out, the synchronization server ignores the associated conflict (that is, behaves as if it were resolved.) This may result in changes being applied to the server database that may be in error.

OnConflict PeopleCode Example: CONFLICTALGORITHM

In this example, if the mobile device is a personal digital assistant (PDA), this code causes changes on the mobile device to always be accepted.

```
If (%SyncServer.ClientPlatform = %MobileDevice_PDA) Then
   %SyncServer.ConflictAlgorithm = %CONFLICTALGORITHM_DEVICEWINS;
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

In this example, the program causes the property values from the mobile device owning the property to take precedence.

```
Function isOwner() Returns boolean
   Local boolean &bOwner;
   /* Use user profile information to determine if the
sync user is the owner of this synchronized Component Interface object... */
   Return &bOwner;
End-Function;

If (isOwner()) Then
   /* No conflict, always accept changes from the owner's
device during synchronization... */
   %SyncServer.ConflictAlgorithm = %CONFLICTALGORITHM_DEVICEWINS;
Else
   /* Reject changes from non-owners when a conflicts occur...  */
   %SyncServer.ConflictAlgorithm = %CONFLICTALGORITHM_SERVERWINS;
End-If;
```

Understanding the Merge Algorithm

The merge algorithm is used when the conflict resolution for the synchronizable component interface is selected as Device Winsor Custom on the properties dialog box for the definition, on the Synchronization tab.

When you select Custom, the merge algorithm is executed before the OnConflict PeopleCode is run. While it is possible for the OnConflict PeopleCode to change the outcome of the merge algorithm, it should not do so. The purpose of OnConflict PeopleCode is to decide whether the device update is accepted or rejected.

The following are the merge algorithm rules:

- Property values changed on the device replace property values on the server.
- Property values changed on the server and not on the device are left equal to the server values.
- Objects added on the server are left in place. Objects added on the device are added on the server.

See Also

Chapter 39, "SyncServer Class," ConflictAlgorithm, page 1600

Using OnSelect

The OnSelect event is invoked during synchronization to implement custom data distribution.

To determine which data to send to the mobile device during synchronization, the synchronization server searches for the results of the OnSelect event as it is defined for each Component Interface. In the Component Interface, you can develop PeopleCode in the OnSelect event to select the record rows based on filtering, such as the mobile device owner, person logged in, sales region, and so on.

The OnSelect event uses the SyncServer class property SyncIDs, which is an array of synchronization Ids (from the SYNCID field.)

Using a SQL statement, select the SYNCID field for each row or instance of data to be synchronized, and pass this array of SYNCIDs in the SyncIDsproperty array to the synchronization server for retrieval. The OnSelect event runs only if the Component Interface has a corresponding mobile page on the mobile device.

 Copyright © 1988-2007, Oracle. All rights reserved.

If the OnSelect event definition contains no PeopleCode, the default search record for the component interface is used to return an unfiltered array of Sync Ids. All data that satisfies the search record for the component interface is synchronized to the mobile device.

If a PeopleCode program exists in OnSelect but is commented out, the PeopleCode is not invoked and nothing is pushed into the SyncIDs array. However, because a PeopleCode program exists, the synchronization server assumes that the SyncIDs array is correctly populated with no SyncID values.

If PeopleCode exists within the OnSelect event definition, the PeopleCode is run and the resulting set of synchronization Ids is used to determine which data is returned to the mobile device. You can define PeopleCode functions containing data distribution rules for reuse or write code that is specific to each Component Interface.

Using the Common Component Function DistributeDataByRules

This function, contained in the FuncLib FUNCLIB_ECMOBIL, can be used in the OnSelect event to filter data to the mobile device.

OnSelect PeopleCode Examples

In this example, the following PeopleCode on an OnSelect event populates the SyncIDs array with all of the sync Ids from the record rows that have an active lead status (LEAD_STATUS), which is a component interface property. This could be used to supply each salesperson with only the leads that are assigned to that salesperson.

```
/* Select Active Leads */
&sql = CreateSQL("SELECT SYNCID FROM PS_RSF_LEAD WHERE LEAD_STATUS = :1", 'A');
While &sql.Fetch(&SyncID)
   %SyncServer.SyncIDs.Push(&SyncID);
End-While;
```

In the following example, the system variable %UserId associates an end user with the PERSON_ID field in the record row. The PeopleCode for the OnSelect event selects all of the synchronization Ids (and associated record rows) that are owned by that user.

```
/* Determine User's Person ID */
SQLExec("SELECT PERSON_ID FROM PSOPRALIAS WHERE OPRID = :1", %UserId, &PersonID);
/* Select Leads with the User's Person ID */
&sql = CreateSQL("SELECT SYNCID FROM PS_RSF_LEAD WHERE PERSON_ID = :1", &PersonID);
While &sql.Fetch(&SyncID)
   %SyncServer.SyncIDs.Push(&SyncID);
End-While;
```

OnSelect and Spidering

Through spidering, rows of data can be required for the mobile device in addition to those selected by the OnSelect event. Suppose that:

- A contact list on a mobile device includes three people: two from the USA states of California and Illinois and one from the Canadian province of Quebec.
- The OnSelect event for state calls for only states or provinces that are in the USA.

The synchronization server retrieves all of the USA states, including California and Illinois, but it also retrieves Quebec because Quebec is already present on the mobile device.

See Also

Chapter 39, "SyncServer Class," SyncIDs, page 1603

Copyright © 1988-2007, Oracle. All rights reserved.

Using OnValidate

The OnValidate event is invoked during out-of-date detection during full synchronization. Use it only when you have medium to large set of data, approximately 100–1000 instances of a Component Interface.

Generally, when a Component Interface instance is accessed for synchronization, the Compoent Interface keys are set and the Get method is called on the Component Interface. If you use OnValidate (or OnValidateSet), this call to the Get method is bypassed. This means that the normal initialization events associated with a Compoent Interface (PreBuild, RowInit, and so are) are also bypassed. This can improve performance. This implies that you may want to use OnValidate (or OnValidateSet) if the performance cost of doing the Get (and running the other events) is greater than running SQL to compute the current version of the instance.

Note. OnValidate is not invoked during upload changes or bootstrap synchronization. It is only invoked during update applications and update business data synchronization.

Use the OnValidate event to compute the current version for the synchronized Component Interface object using the SyncServer validateID property, and return the result to the synchronization server using the SyncServer validateVersion property.

If the Component Interface instance does not exist or is not relevant for some other reason (for example, security), not setting a value into version is interpreted as a validate status of No Relevance and the mobile object is removed from the mobile device.

Use the OnValidate event to get the current version for the object and either set that object to be relevant or remove it from the mobile device.

The OnValidate event uses the SyncServer properties validateID and validateVersion.

Considerations Using OnValidate

If you create PeopleCode for this event, you must maintain it. Anytime you expose an additional collection in the Component Interface, or remove a collection, you must change your OnValidate PeopleCode.

If you have any PeopleCode in the OnValidateSet event, even if it is commented out, the OnValidate event does not run.

OnValidate runs on every instance of the Component Interface. This means that any code in this event fires once per Component Interface instance.

OnValidate PeopleCode Example

The following PeopleCode sets validation for this instance of the QE_DEMO_CONTACT component interface.

```
SQLExec("SELECT LASTUPDDTTM FROM %Table(QE_DEMO_CONTACT) WHERE SYNCID=:1",
%SyncServer.validateID, %SyncServer.validateVersion);
```

See Also

Chapter 39, “SyncServer Class,” validateID, page 1604

Chapter 39, “SyncServer Class,” validateVersion, page 1604

 Copyright © 1988-2007, Oracle. All rights reserved.

Using Attachment Events

PeopleTools supports attachments for mobile. The attachment consists of header information (such as size, type, description) and the corresponding document object. The attachment events occur when the mobile user synchronizes the mobile device.

OnGetProperty Event

The synchronization server invokes the OnGetProperty event when the value for the attachment property is requested. The specific implementation of this event is defined by the developer, and based on the application requirements. It can be used to specify which attachments are downloaded to the mobile device.

The following is a sample of PeopleCode that could be used in the OnGetProperty event. In the example, the PropertyValue property is used in the GetAttachment function to specify the filename and location of the attachment file on the synchronization server.

```
If QE_MB_CUST_NEED.FTPSITE <> "" And
      QE_MB_CUST_NEED.ATTACHSYSFILENAME <> "" Then
   %SyncServer.PropertyValue = "c:\PSMobile_FileAttach\" |
       QE_MB_CUST_NEED.ATTACHSYSFILENAME;
   &Result = GetAttachment(QE_MB_CUST_NEED.FTPSITE, QE_MB_CUST_⇒
NEED.ATTACHSYSFILENAME,
      %SyncServer.PropertyValue);
End-If;
```

OnGetPropertyFilter Event

The synchronization server invokes the OnGetPropertyFilter event prior to invoking the OnGetProperty event to allow the application to apply filtering rules to an individual attachment.

If the SyncServer ExcludeProperty property is set to True, the synchronization server only sends the header of the attachment to the mobile device. It does not send the contents of the attachment.

Here is a sample of PeopleCode that could be used in the OnGetPropertyFilter event.

```
If ((%SyncServer.ClientPlatform = %MobileDevice_PDA) And
   (SERVICEORDER_DAMAGEDPART.SIZE > &nMaxAttachmentSize)) Then
   %SyncServer.ExcludeProperty = True;
End-If;
```

OnSetProperty Event

The synchronization server invokes the OnSetProperty event when the value for the attachment is modified. The specific implementation of this event is based on application requirements. It can be used to specify whether the attachment is uploaded from the mobile device.

Here is a sample of PeopleCode that could be used in the OnSetProperty event. The PutAttachment function returns the location of the attachment file in PropertyValue.

```
If QE_MB_CUST_NEED.FTPSITE <> "" And
      QE_MB_CUST_NEED.ATTACHSYSFILENAME <> "" Then
     &Result = PutAttachment(QE_MB_CUST_NEED.FTPSITE, QE_MB_CUST_⇒
NEED.ATTACHSYSFILENAME,
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
        %SyncServer.PropertyValue);
End-If;
```

Using Filtering

Use the CheckForChanges method along with the LastSyncTime and LastSyncRowCount properties to specify whether the component interfaces on the devices are updated. You would use this ability if your component interfaces are relatively static, meaning that no changes will have occurred to that component interface data during most full synchronization operations.

A filter attribute indicates whether a component interface is to be synchronized or not. When the mobile device receives the response, if a full synchronization is in progress, the device looks for the presence of the filter attribute. If it is present, and the value is true, subsequent processing of the component interface is streamlined.

The filtered synchronization processing occurs as follows:

- If a reference list of SyncIDs is present in the response, the synchronization server constructs lists of existing, new and deleted objects. The list of existing objects is ignored. No validate or get requests are issued for these objects. New objects are retrieved from the server, and deleted objects are removed from the device.
- If the select response contains no reference list of SyncIDs, or if all SyncIDs are found to exist on the device, no other processing occurs for that component interface.
- When a component interface is not filtered, the mobile device saves the timestamp and row count information once the processing for that component interface is complete. All component interface objects retain the timestamp and row count from the last time they were fully synchronized.

See Also

Chapter 39, “SyncServer Class,” CheckForChanges, page 1596

Chapter 39, “SyncServer Class,” LastSyncDateTime, page 1601

Chapter 39, “SyncServer Class,” LastSyncRowCount, page 1602

Scope of SyncServer Objects

A SyncServer object can only be instantiated as Local. SyncServer objects are declared using the SyncServer data type. For example:

```
Local SyncServer &MySyncServer;
```

You can also declare and set the SyncServer variable in the same line. For example:

```
Local SyncServer &MySyncServer = %SyncServer;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

SyncServer Class

The SyncServer object is accessed using the %SyncServer system variable.

SyncServer Class Methods

This section describes the SyncServer class methods.

AddReference

Syntax

AddReference(*SyncID*[, *CIDefnName*])

Description

Use the AddReference method to identify every instance of an ad-hoc referenced component interface that you identified using the AddReferenceType method. Use AddReference in one of the Component build events (PreBuild, PostBuild, RowInit).

Parameters

Parameter	Description
SyncID	This is a string containing the synchronization ID (SyncID).
CIDefnName	This is a string containing the name of the referenced Component Interface. Preface this string with the reserved word **CompIntfc**.

Returns

None.

Example

```
%SyncServer.AddReference(&SyncID, CompIntfc.RefCI);
```

See Also

Chapter 39, “SyncServer Class,” Advanced Considerations for Synchronization, page 1605

AddReferenceType

Syntax

AddReferenceType(*CIDefnName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AddReferenceType method in the OnGetDefinition event to identify every ad-hoc reference component interfaces. You must have one call for every instance of an an-hoc Component Interface type in the current component interface.

Parameters

Parameter	Description
CIDefnName	This is a string containing the name of the referenced Component Interface. Prefix this string with the reserved word **CompIntfc**.

Returns

None.

Example

```
%SyncServer.AddReferencedType(CompIntfc.RefCI);
```

CheckForChanges

Syntax

CheckForChanges(*SyncDateTime*, *TotalRowCount*[, *HasRefList*])

Description

Use the CheckForChanges method to determine if a component interface needs to be synchronized. This method determines the DateTime value of all rows that make up the object, as well as a total count of those rows. If the maximum DateTime value exceeds the device value or if the row counts are different, the component interface is identified as changed and the data in the component interface will be resynchronized.

If neither condition is true, an additional attribute is added to the result element of the response: `filter="true"`.

Note. If you want to filter which component interface objects are updated, you should use CheckForChanges even during a bootstrap, in order to seed the RowCount setting on the device.

If there are rules to apply to the data, they must be applied whenever CheckForChanges returns false. They might also be applied when CheckForChanges returns true, especially if the data is sensitive and the relevance rules are likely to change.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
SyncDateTime	Specify the date and time the object was last updated on the server (the maximum update DateTime across all rows that comprise the object), as a DateTime value.
TotalRowCount	Specify the number of rows in the object on the server, as a number.
HasRefList	Specify if the response must contain a reference list even when the component interface is being filtered. This parameter takes a Boolean value. The default value is false. If you specify true for the value, the relevance rules in the OnSelect event must always be applied, even when the CheckForChanges method returns True, to ensure that the rule or authorization changes take effect immediately. If the value of this parameter is false, and if CheckForChanges returns true (which means this component interface will not be processed), then no reference list is included in the response. This means the synchronization processor skips the get and delete processing of this component interface, and the device data is left as-is.

Returns

A Boolean value: true, the device data is identical to the existing data and so the component interface should not be processed, false indicates that the data has changed and the component interface should be resynchronized.

Example

In the following example, the component interface is automatically filtered and no processing is done.

```
%SyncServer.CheckForChanges(%SyncServer.LastSyncTime, %SyncServer.LastSyncRow⇒
Count);
```

In this example, applying the relevance rules is complex and slow, and you only want to do it when changes have occurred, so the code includes false as the third parameter for CheckForChanges. This example code would go in an OnSelect event.

```
Local datetime &SyncDttm;
Local number &RowCount;

/* The following logic enables filtering based on whether any data has changed in⇒
 the underlying table. */
SQLExec("SELECT MAX(SYNCDTTM), COUNT(*) FROM PS_LEVEL0", &SyncDttm, &RowCount);

If %SyncServer.CheckForChanges(&SyncDttm, &RowCount, False) = False Then
   /* Add logic here to populate %SyncServer.SyncIDs based on relevance rules. */
   /* code here */
End-If;
```

In the following example, the logic filters according to a customer defined synchronization frequency (in all days.) This type of code would go in the OnSelect event.

```
Local datetime &DateTime;
Local number &SyncFrequency;

If All(%SyncServer.LastSyncDateTime) Then
   /* Not bootstrap sync.  Determine the DateTime when we should next perform a⇒
 full sync. */
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   SQLExec("SELECT FREQUENCY FROM PS_SYNC_FREQ_TBL WHERE CINAME = :1", Comp⇒
Intfc.MYCI, &SyncFrequency);
   &DateTime = AddToDateTime(%SyncServer.LastSyncDateTime, 0, 0, &SyncFrequency,⇒
 0, 0, 0);
Else
   /* Bootstrap sync.  Seed &DateTime so the if-condition below will be true. */
   &DateTime = %Datetime;
End-If;

If &DateTime <= %Datetime Then
   /* Time for a full sync.  No need to call CheckForChanges, because
SyncServer will always use current date-time in the response,
and the row count is irrelevant. */
Else
   /* Time to filter.  Call CheckForChanges with request value to force it. */
   %SyncServer.CheckForChanges(%SyncServer.LastSyncDateTime, 0);
End-If;

/* This CI has no relevance rules.  Tell SyncServer to select all instances. */
%SyncServer.SelectAll();
```

See Also

Chapter 39, "SyncServer Class," LastSyncDateTime, page 1601 and Chapter 39, "SyncServer Class," LastSyncRowCount, page 1602.

Notify

Syntax

Notify(*message_text*, *severity* [, *object_type*, *object_id*])

Description

Use this method to pass an error message during synchronization.

Parameters

Parameter	Description
message_text	Specify the message text.
severity	Specify the severity of the message.
object_type	Specify the object type.
object_id	Specify the object ID.

Returns

None.

 Copyright © 1988-2007, Oracle. All rights reserved.

SelectAll

Syntax

```
SelectAll()
```

Description

Use the SelectAll method to perform the default select processing. If this method is called in an OnSelect event, all instances of the component interface are added to the reference list in the select response. This means that the SyncIDs array is ignored, even if it is not empty.

Warning! If the SelectAll method is not called in the OnSelect event, and if CheckForChanges does not suppress the reference list, the response has a reference list containing all elements of the SyncIDs array. If the array is empty, the reference list is also empty, therefore *all* instances of that component interface are removed from the device.

Parameters

None.

Returns

None.

See Also

Chapter 39, "SyncServer Class," CheckForChanges, page 1596

SyncServer Class Properties

This section describes the SyncServer class properties.

ClientPlatform

Description

This property specifies the type of mobile device.

Value	Description
%MobileDevice_PDA	A PDA.
%MobileDevice_Laptop	A laptop computer.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

ConflictAlgorithm

Description

This property is used to cause the device update to be rejected or accepted. This property is checked after all OnConflict PeopleCode has run. This property overrides the ConflictStatus property.

Value	Description
%CONFLICTALGORITHM_SERVERWINS	The server always wins, which causes the device update to be rejected.
%CONFLICTALGORITHM_DEVICEWINS	The mobile device always wins, which causes the device update to be accepted.

This property is read-write.

ConflictStatus

Description

This property sets the conflict status. This property is valid only with the OnConflict event.

Value	Description
%CONFLICTSTATUS_NOCONFLICT	Remove from the list of conflicts.
%CONFLICTSTATUS_CONFLICT	Do not remove from the list of conflicts.

This property is read-write.

ExcludeProperty

Description

This property indicates to synchronization server that the contents of an attachment should *not* be downloaded to the mobile device. (The header is still downloaded.)

This property is only used with attachments.

This property is read-write.

Example

```
If ((%SyncServer.ClientPlatform = %MobileDevice_PDA) And
   (SERVICEORDER_DAMAGEDPART.SIZE > &nMaxAttachmentSize)) Then
   %SyncServer.ExcludeProperty = True;
End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ExportLocation

Description

This property allows a developer to instruct the mobile device to export a copy of an attachment to a specific file on the device's disk during synchronization.

You can use this property to automatically extract product images (JPEGs) to a directory on the mobile device to make them accessible from HTML image tags included in HTML that is generated as part of an HTML Area.

This property takes a value of a relative or fully qualified path and filename. If you want to put files into the Res directory (all graphics must be in this directory to display correctly) you could use the following PeopleCode:

```
%SyncServer.ExportLocation = "..\res\filename.ext";
```

This property is used in the OnGetProperty event for an attachment reference file.

This property is read-write.

IsReferenceUnresolved

Description

When two or more new objects are uploaded during a synchronization, and they contain references to each other, it can create a deadlock condition on the server, since the references cannot be resolved in the database yet, and therefore neither of the objects can be successfully saved. To break the deadlock, SyncServer performs a tentative save on one of the objects without attempting to resolve any of it's references. This makes it possible for the other object's reference to be resolved, and it can be permanently saved. Once that is completed, the first object's reference can also now be resolved, so a second save is performed to add the appropriate key values from the resolved reference. The purpose of this enhancement is to provide a new

This property indicates whether or not a tentative save is in progress, and whether the object being processed has had it's references resolved.

This property returns a boolean value: true if a tentative save is in progress, false otherwise.

This property is read-only.

LastSyncDateTime

Description

This property returns the datetime from the last time this component interface was synchronized, as a DateTime value. This property is used only in the OnSelect event.

This property is read-only.

See Also

Chapter 39, "SyncServer Class," CheckForChanges, page 1596 and Chapter 39, "SyncServer Class," LastSyncRowCount, page 1602.

Copyright © 1988-2007, Oracle. All rights reserved.

LastSyncRowCount

Description

This property returns the number of rows that were available in this component interface the last time it was synchronized, as a number. This property is used only in the OnSelect event.

This property is read-only.

See Also

Chapter 39, "SyncServer Class," LastSyncDateTime, page 1601 and Chapter 39, "SyncServer Class," CheckForChanges, page 1596.

PropertyValue

Description

This property provides the path and filename of the attachment file on the synchronization server. It is used with the GetAttachment and PutAttachment functions.

In the OnGetProperty event, set PropertyValue in the GetAttachment function to the filename and location of the attachment file.

In the OnSetProperty event, the PutAttachment function returns the location of the attachment file in PropertyValue.

This property is read-write.

Example

In the code example below, the second If statement checks for whether the attachment was successfully retrieved, prior to setting the value for PropertyValue.

```
If QE_MB_CUST_NEED.FTPSITE <> "" And
      QE_MB_CUST_NEED.ATTACHSYSFILENAME <> "" Then
   %SyncServer.PropertyValue = "c:\PSMobile_FileAttach\" |
       QE_MB_CUST_NEED.ATTACHSYSFILENAME;
   &Result = GetAttachment(QE_MB_CUST_NEED.FTPSITE, QE_MB_CUST_⇒
NEED.ATTACHSYSFILENAME,
      %SyncServer.PropertyValue);
End-If;
```

SessionID

Description

This property returns the session ID. This is not unique for a device. It is used for internal chunking in OnValidateSet, and can only be used with this event.

This property is read-only.

See Also

Chapter 39, "SyncServer Class," Using OnValidateSet, page 1605

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following PeopleCode sets validation for batches of instances of the QE_DEMO_CONTACT Component Interface.

```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=0 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM >=
B.LASTUPDDTTM)", %SyncServer.SessionID, %SyncServer.TypeID);

SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=1 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM <⇒
 B.LASTUPDDTTM)',
%SyncServer.SessionID, %SyncServer.TypeID);
```

Synchronizing

Description

This property is true during synchronization and false otherwise. Use this to perform (or omit) tasks during synchronization. For example, you might limit the number of rows processed in a child scroll during synchronization.

This property is read-only.

SyncIDs

Description

This property is an array containing the synchronization Ids. Populate the SyncIDs array with the sync Ids of the synchronizable component interface objects that you want to be synchronized with the mobile objects on the mobile device. Use the array class Push method to add a SyncID to the SyncIDs array.

The SyncID array is used as a list of sync Ids for instances that satisfy a set of relevance criteria, such as only synchronize Customer objects that are associated to one salesperson. A SQL object that runs a select on a view might be used to calculate the set of sync Ids for Customer objects that are assigned to that salesperson.

This property can be used only with the OnSelect event.

This property is read-write.

See Also

Chapter 8, “Array Class,” page 215

TypeID

Description

This property contains the sync ID for the current component interface type (not just an instance, but the entire type.) This property is used for internal chunking in OnValidateSet, and can only be used with this event.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following PeopleCode sets validation for batches of instances of the QE_DEMO_CONTACT component interface.

```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=0 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM >=
B.LASTUPDDTTM)", %SyncServer.SessionID, %SyncServer.TypeID);

SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=1 FROM
PSSYNCSERVERVAL A, %Table(QE_DEMO_CONTACT) B WHERE A.SYNCSESSIONID =
:1 AND A.SYNCTYPEID = :2 AND (A.SYNCID = B.SYNCID AND A.SYNCDTTM <⇒
 B.LASTUPDDTTM)', %SyncServer.SessionID, %SyncServer.TypeID);
```

See Also

Chapter 39, “SyncServer Class,” Using OnValidateSet, page 1605

validateID

Description

This property contains the sync ID for the only current *instance* of the Component Interface. It is created using the SYNCID field for that component interface (that field is set in the record properties for that Component Interface.)

This property can be used only with the OnValidate event.

This property is read-write.

validateRowCount

Description

This property returns the total number of rows in a hierarchical synchronizable Component Interface object.

For example, a service order with 10 lines has a validateRowCount value of 11. The property default is 1.

This property can be used only with the OnValidate event.

This property is read-write.

validateVersion

Description

This property contains the version for the instance of the Component Interface. Set validateVersion using the SYNCDTTM field. If the component interface object does not exist, or if the object is not to be used for some other reason (such as security), then do not set the validateVersion property.

This property can be used only with the OnValidate event.

ValidateVersion is initially set to the device version (that is, `max(syncdttm)` across all rows that make up the object). When leaving OnValidate, ValidateVersion must be reset to reflect the current version of the object.

 Copyright © 1988-2007, Oracle. All rights reserved.

If you want the object removed from the device, don't set the validateVersion property.

This property is read-write.

Example

The following code example forces an object to be re-synched:

```
%SyncServer.ValidateVersion = %CurrentDateTime;
```

If you know the object hasn't changed and want to keep it from being re-synced, use the following code (assuming the VersionRowCount property has also been set correctly):

```
%SyncServer.ValidateVersion = %SyncServer.ValidateVersion;
```

Advanced Considerations for Synchronization

After you mobile application is running without errors, you might decide to further tune your application. Two events enable you to do this in specific circumstances.

This section discusses the following:

- Using OnValidateSet
- Using OnGetDefinition

Using OnValidateSet

You should only use OnValidateSet when you have a large number of Component Interface instances to be synchronized, more than 1000.

Before you try to create an OnValidateSet PeopleCode program, you should first create OnValidate PeopleCode. This enables you to determine the shape of the Component Interface instance, the number of child rows and collections. Having a thorough understanding of this helps you create the OnValidateSet PeopleCode.

Though the OnValidateSet PeopleCode may improve performance, there is additional maintenance with it. Every time you add an exposed collection, or remove one, you must change your PeopleCode program to reflect the new data structure.

Note. If you have any PeopleCode in OnValidateSet, including a program that has been commented out, the OnValidate event does not run.

When this event occurs, a table on the synchronization server (PSSYNCSERVERAL) is populated with all of the synchronization Ids for the component type. They table is keyed by Session ID (SYNCSESSIONID field) and Type ID (SYNCTYPEID field).

The status of the row is tracked in the status column (SYNCSTATUS field). All of the rows in the table begin with a status of 3, meaning that the corresponding Component Interface instance should be purged from the synchronization server. The other valid values for the status field are:

Copyright © 1988-2007, Oracle. All rights reserved.

Status	Description
0	Current. This causes the corresponding Component Interface instance to not be synchronized.
1	Out of date. This causes a Get to be called on the corresponding Component Interface instance, and the rest of the event flow to be joined.

The OnValidateSet event uses the SyncServer properties SessionID and TypeID. These should be populated by values from the SYNCSESSIONID and SYNCTYPEID fields, respectively.

Considerations Using OnValidateSet

If you set breakpoints in a Component Interface, or do a trace , you will see OnValidateSet running multiple times. This happens when the synchronization server sends a request to an application server for OnValidateSet to run and does not receive a response soon afterward. Then the synchronization server sends out another request. In theory, OnValidateSet only runs once per Component Interface Type. In practice, more than one OnValidateSet request runs concurrently in certain environments.

OnValidateSet PeopleCode Example

The following PeopleCode sets validation for batches of instances of the QE_DEMO_CONTACT component interface.

```
SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=0
WHERE SYNCSESSIONID =:1 AND SYNCTYPEID = :2
AND EXISTS (SELECT 'X' FROM %Table(QE_DEMO_CONTACT) B
WHERE PSSYNCSERVERVAL.SYNCID = B.SYNCID
AND PSSYNCSERVERVAL.SYNCDTTM >= B.LASTUPDDTTM)",
%SyncServer.SessionID, %SyncServer.TypeID);

SQLExec("UPDATE PSSYNCSERVERVAL SET SYNCSTATUS=1
WHERE SYNCSESSIONID =:1 AND SYNCTYPEID = :2
AND EXISTS (SELECT 'X' FROM %Table(QE_DEMO_CONTACT) B
WHERE PSSYNCSERVERVAL.SYNCID = B.SYNCID
AND PSSYNCSERVERVAL.SYNCDTTM < B.LASTUPDDTTM)",
%SyncServer.SessionID, %SyncServer.TypeID);
```

Using OnGetDefinition

Using the AddReference and AddReferenceType methods, you can create ad-hoc references to Component Interfaces using PeopleCode, instead of defining the reference and the relationship in Application Designer.

Use the OnGetDefinition event when you want to access the ad-hoc references specified with the AddReference method in the component events. In this event, you must specify every instance of an ad-hoc reference. The referenced Component Interfaces are defined by their SyncIDs. The referenced Component Interfaces objects can reference other referenced Component Interface objects.

 Copyright © 1988-2007, Oracle. All rights reserved.

When to Use OnGetDefinition

You should only consider using ad-hoc references when you want to dynamically establish read-only references of static data. Using these methods, you are not establishing a relationship between the Component Interfaces. You cannot navigate these references. You cannot make any changes to these instances.

OnGetDefinition PeopleCode Example

In the OnGetDefinition event, have one call to the AddReferenceType method for every referenced Component Interface type in the current Component Interface.

```
%SyncServer.AddReferencedType(CompIntfc.RefCI);
```

In one of the Component build events (PreBuild, PostBuild, RowInit) you must also include a call to the AddReference method for every instance (identified by the SyncID) of the referenced Component Interface.

```
%SyncServer.AddReference(&SyncID, CompIntfc.RefCI);
```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 40

Tree Classes

This chapter provides an overview of Tree classes and discusses the following topics:

- Relationships between different Tree classes.
- Collections in the Tree classes.
- Error handling with trees.
- Leaves and nodes insert verification.
- Data types for tree objects.
- Scope of the tree objects.
- Tree classes implementation.
- Tree classes reference.

Understanding Tree Classes

Using the Tree classes in your PeopleCode, you have access to all the functionality of PeopleSoft Tree Manager. Your application should instantiate the appropriate tree objects when it must work with the tree system database data, call the appropriate methods and properties, then close the objects when it is finished.

Creating or deleting a tree *object* does not create or delete tree system database information. You must call the method for that tree object directly to create or delete database information, that is, the Create or Delete method.

One instance of a tree or tree structure object can be created to work on multiple database entities. However, only one tree or tree structure can be open at a time. If you open a Tree before closing the one that's currently open, you receive an error. You must explicitly close a tree or tree structure (using the Close method) before you try to open a second one.

All of the classes, and most of the properties and methods that make up the Tree Classes have a GUI representation in PeopleSoft Tree Manager. This document assumes that the reader is familiar with PeopleSoft Tree Manager.

The following classes make up the Tree classes:

- Branch Collection
- Leaf
- Level
- Level Collection
- Node
- Tree

Copyright © 1988-2007, Oracle. All rights reserved.

- Tree Structure

With most of the classes of objects in PeopleTools, when you use a *Getxxx* method, you are fully instantiating an object. However, for the tree class, when you use GetTree (from the Session object), you get a closed tree. A closed tree is a tree object with just the key fields filled in. The rest of the data is not present. To open the tree, you must use the Open method. Working with closed trees can improve your performance. The same applies to a tree structure: it's closed when you get it from the Session object, and you must open it before you can access its properties or change it.

There aren't any built-in functions for the Tree classes: objects are instantiated from identifiers, from other objects, or from a Session object.

Using the GenerateTree function, you can produce a GUI representation of a tree in the PeopleSoft Pure Internet Architecture.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TreeDetailInNode

Chapter 36, "Session Class," page 1511

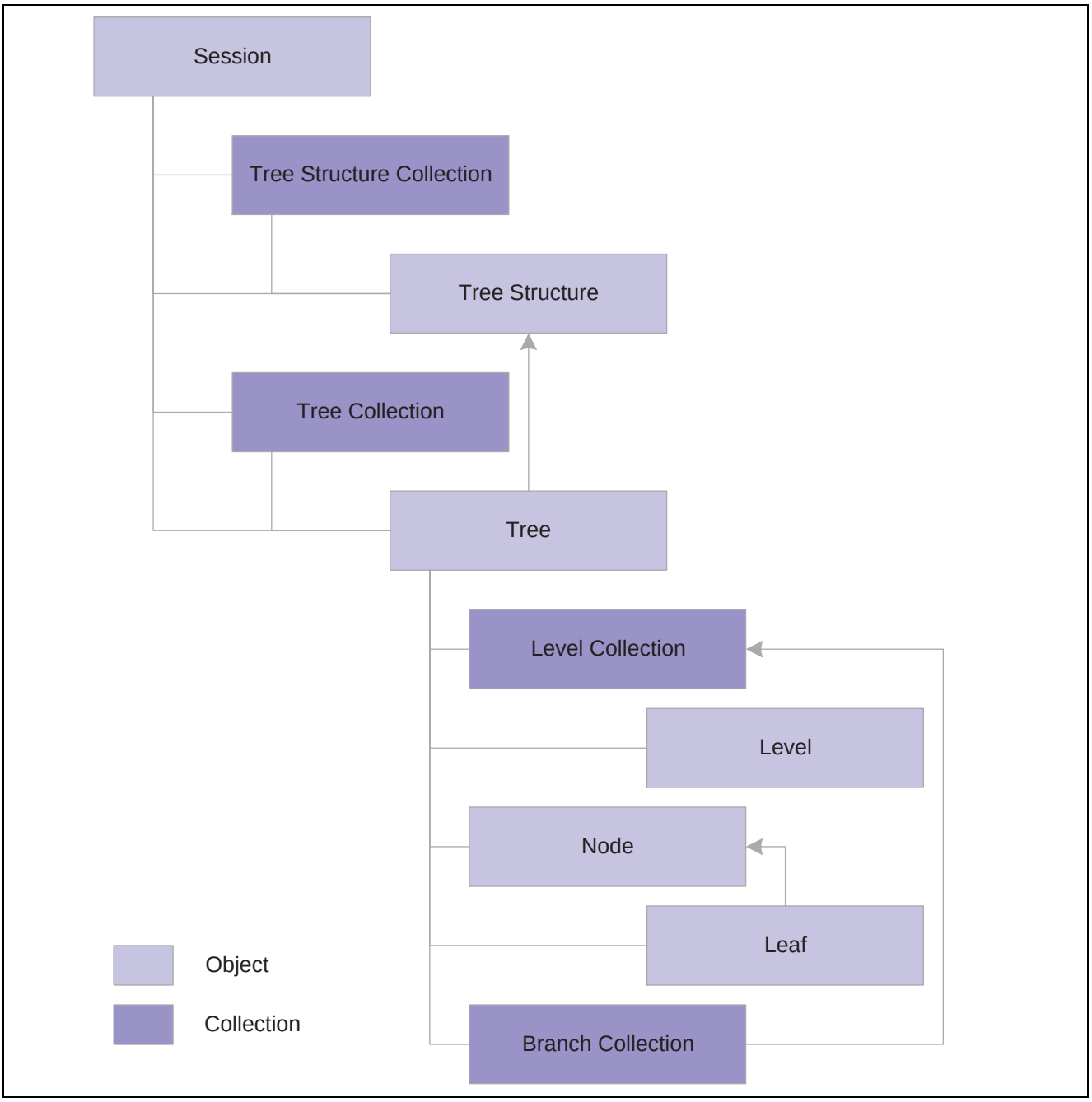
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Developer's Guide, "Understanding HTML Trees and the GenerateHTML Function," Using the GenerateTree Function

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Introduction to PeopleSoft Tree Manager"

Relationships Between Different Tree Classes

The following diagram shows the Tree classes and their relationships. Objects at the tip of the arrowhead are accessed or created from the object at the left of the arrowhead. Squares indicate objects. Ovals indicate collections.

 Copyright © 1988-2007, Oracle. All rights reserved.



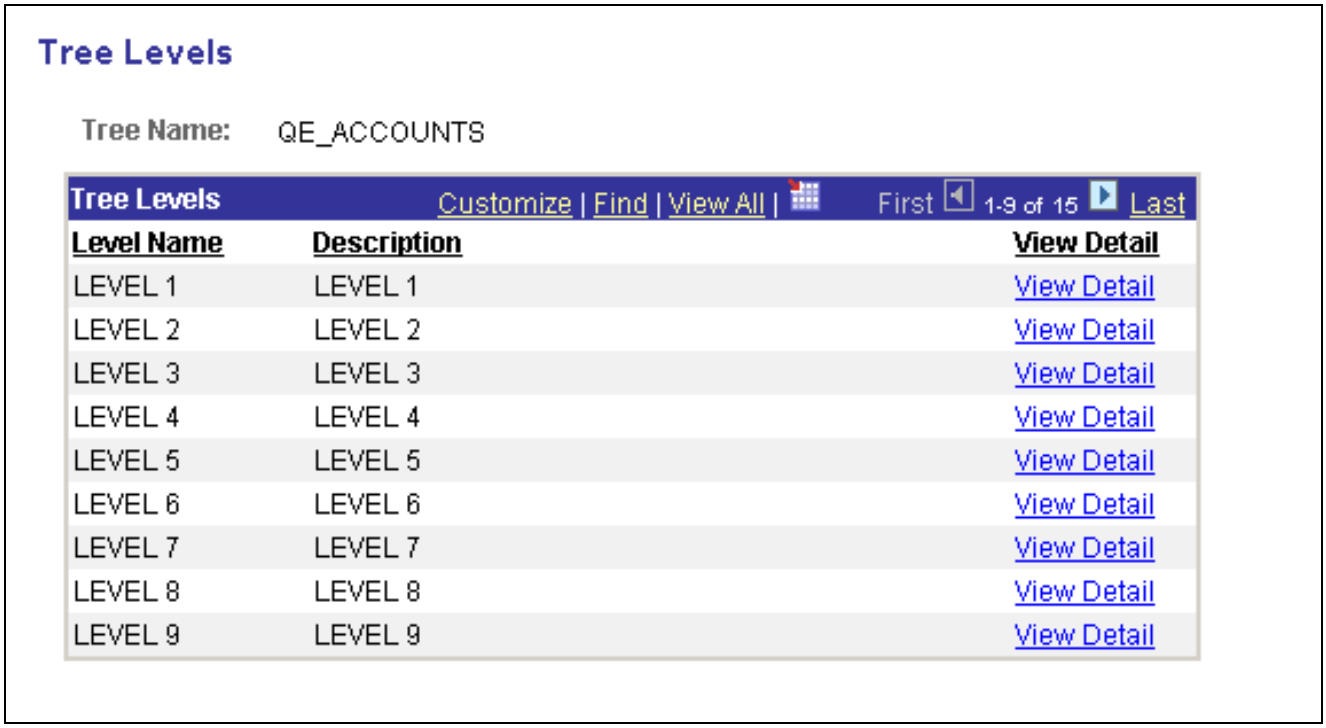
Tree Classes and their relationships

- From the session object, you can get identifiers for a tree or a tree structure.
- From the level collection object, you can instantiate a level object.
- From the branch collection object, you can get an identifier for a branch.
- From the tree object, you can instantiate a tree, a tree structure, a branch collection, a leaf, a node, and a level collection.
- From the node object, you can instantiate a leaf object or a node object.
- From the leaf object, you can instantiate a leaf object or a node object.

Copyright © 1988-2007, Oracle. All rights reserved.

Collections in the Tree Classes

A *collection* is a set of similar things, like a group of already existing branches or levels. Like everything else in the tree classes, collections have a GUI representation. For example, when you find an existing branch from PeopleSoft Tree Manager, you get a dialog that lets you select the branches you want. This dialog box data is represented in PeopleCode as the *tree level collection.*



Example of level collection

The following collections are part of the Tree classes:

- Branch collection
- Level collection

Error Handling With Trees

The Tree classes log descriptive information regarding errors and warnings to the PSMessages collection, instantiated from a session object.

In your program, use the PSMessages collection to identify and report to the user any errors that are encountered during processing.

The tree classes log errors "interactively", that is, as they happen. For example, suppose you had created a new tree, and were setting the effective date using the effective date property (EffDt). If you used an invalid value for the effective date, the error would be logged in the PSMessages collection as soon as you set the value, not when you saved the tree.

When you want to check for errors depends on your application. However, if you check for errors after every assignment, you may see a performance degradation.

Copyright © 1988-2007, Oracle. All rights reserved.

One way to check for errors is to check the number of messages in the PSMessages collection, using the Count property. If the Count is 0, no error or warning messages have been logged to the PSMessages collection.

```
Local ApiObject &MYSESSION;
Local ApiObject &ERRORCOL;
Local ApiObject &TREE, TREELIST;

&MYSESSION = %Session;

If &MYSESSION Then
   /* connection is good */

&MyTree = &Session.gettree();
    /* open a tree */
   &TreeReturn = &MyTree.Open("", "", "PERSONAL_DATA", "1999-06-01", "",true);
    /* Do error checking */

   &ERRORCOL = &MYSESSION.PSMessages;
   If (&ERRORCOL.Count <> 0) Then
      /* errors occurred - do processing */
   Else
      /* no errors */
   End-If;
   Else
   /* do processing for no connection */
End-If;
```

You can also do an error check based on the return value of your API calls

```
If ALL(&TreeReturn) then
   /* processing errors */
End-if:
```

See Also

Chapter 36, “Session Class,” Error Handling, page 1512

Leaves and Nodes Insert Verification

InsertChildLeaf, InsertSibNode, InsertChildNode, and so on, can return False or a Null object reference, depending upon the type of error encountered. You may want to declare references to new leaves or nodes as type ANY until after you verify they were actually created. You can do this by using the None or All PeopleCode functions to determine whether a valid Leaf or Node Object was created. If a Leaf or Node object was created ALL()returns True and None() returns False.

```
&NewLeaf = &RootNode.InsertChildLeaf("8000", "8999");
If NONE(&NewLeaf) Then
   /* Leaf not inserted, do error processing */
   &Messages = &Session.PSMessages;
   If &Messages.WarningPending Then
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
      /* do error processing */
   End-if;
End-if;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," None

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All

Data Types for Tree Objects

All tree objects, that is, trees, tree structures, nodes, levels, and so on, are declared as type ApiObject. For example:

```
Local ApiObject &MYTREE;

Global ApiObject &MYNODE;
```

All tree objects can be declared as Local, Component or Global.

Scope of the Tree Objects

All tree objects, that is, trees, tree structures, nodes, leaves, and so on, can *only* be instantiated from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in an application class, Application Engine PeopleCode, record field PeopleCode, and so on.

You can instantiate a tree or tree structure object only from a session object. You have to instantiate the session object, and connect to the database, before you can instantiate a tree or tree structure.

```
Local ApiObject &TREELIST;
Local ApiObject &MYSESSION;

&MYSESSION = %Session;

If &MYSESSION Then
   /* connection is good */
Else
   /* do error processing */
End-if;
```

Note. Tree Classes are accessible only through PeopleCode.

 Copyright © 1988-2007, Oracle. All rights reserved.

Tree Classes Implementation

You will often want to create a new tree. The following procedure discusses this action in more detail.

The TreeMover Application Engine program uses the Tree API (and File Layouts) for importing a tree from a flat file or exporting a tree to a flat file.

To create a new tree:

In this example, you are creating a new tree based on an existing tree structure. The following is the complete code sample: the steps explain each line.

```
Local ApiObject &Session;
Local ApiObject &TreeList, &MyTree;
Local ApiObject &LvlColl;

&Session = %Session;

&MyTree = &Session.GetT‘ree();
   /* create new tree */

If All(&MyTree) Then
  &TreeReturn = &MyTree.Create("", "", "PERSONAL_DATA2", "1999-06-01", "PERSONAL_⇒
DATA");

   If &TreeReturn <> 0 then
      /* check PSMessages collection */
   End-if;

   &MyTree.description = "test tree";

   /* add level */
   &LvlColl = &MyTree.levels;
   &Level = &LvlColl.add("FIRST LVL");
   &Level.description = "First Level";

   /* add root node */
   &RootNode= &MyTree.insertroot("00001");

If ALL(&RootNode) Then
 /* insert a leaf */
       &NewLeaf = &RootNode.InsertChildLeaf("8000", "8999");

   /* save new tree */
   &RSLT = &MyTree.Save();

   /* Do error checking */
   If &RSLT <> 0 Then
      /* errors occurred = do error checking */
      &ERRORCOL = &Session.PSMessages;
      For &I = 1 To &ERRORCOL.count
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
         /* do error processing */
      End-For;
   Else
      /* no errors - saved correctly - do other processing */
   End-If;
End-if;

End-if;
```

1. Get a session object.

   Before you can get a tree, you have to get a session object. The session controls access to the tree, provides error tracing, enables you to set the runtime environment, and so on. Use the %Session system variable to return a reference to the current PeopleSoft session.

   ```
   &Session = %Session;
   ```

2. Get a tree object.

   Use the GetTree method specifying a null value (" ") to return a closed tree object.

   ```
   &MyTree = &Session.GetT‘ree();
   ```

3. Create the Tree.

   The Create method creates a new tree with the name PERSONAL_DATA2. To ensure that you have a valid tree, use the All built-in function. Description is a required property (if you don’t specify something for Description you cannot save the tree.)

   ```
   &TreeReturn = &MyTree.Create("", "", "PERSONAL_DATA2", "1999-06-01", "PERSONAL_⇒
   DATA");

   &MyTree.description = "test tree";
   ```

4. Add a level.

   To add a level, you have to instantiate a level collection. Although there aren’t any levels in the tree, you can still access this collection. Use the Add method with the level collection to add a new level. Remember, the level name must be 8 characters or less. Description is a required property (if you don’t specify something for Description you cannot save the tree.)

   ```
   &LvlColl = &MyTree.levels;
   &Level = &LvlColl.add("FIRST LVL");
   &Level.description = "First Level";
   ```

5. Add the root node.

   Because this is a new tree, you must first add the root node.

   ```
   &RootNode = &MyTree.insertroot("00001");
   ```

6. Add a leaf.

   To add a new leaf, you must have a reference to the parent node object. Using the All built-in function ensures that there is a root node before you try to insert the leaf with the InsertChildLeaf method.

   ```
   If ALL(&RootNode) Then
      &NewLeaf = &RootNode.InsertChildLeaf("8000", "8999");
   ```

 Copyright © 1988-2007, Oracle. All rights reserved.

7. Save the tree.

   When you execute the Save method, the new tree is saved to the database.

```
&RSLT = &MyTree.Save();
```

   Note. If you're running the tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

8. Check for errors.

   You can check if there were any errors using the PSMessages property on the session object.

```
If All (&RSLT) Then
   /* errors occurred = do error checking */
   &ERRORCOL = &Session.PSMessages;
   For &I = 1 To &ERRORCOL.count
      /* do error processing */
   End-For;
Else
   /* no errors - saved correctly - do other processing */
End-If;
```

   If there are multiple errors, all errors are logged to the PSMessages collection, not just the first occurrence of an error.

Note. If you've called the Tree API from an Application Engine program, all errors are also logged in the application engine error log tables.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Using TreeMover"

Session Class Methods

Tree Classes don't have any built-in functions. Instead, they're instantiated from the Session Class.

In this section, we discuss the Session class methods. The methods are discussed in alphabetical order.

GetTree

Syntax

GetTree()

Description

The GetTree method returns a closed tree object. Before you can use some of the methods or any of the properties, you must use the Open method to open a tree object.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a closed tree object.

Example

To create a new tree, use the following:

```
&MYTREE = %Session.GetTree();
&MYTREE.Create("","","PERSONAL_NEW", "05-05-1997", "PERSONAL_DATA");
```

To determine if a tree already exists in the database, use the following code:

```
&MYTREE = %Session.GetTree();
If &MYTREE.Exists("","","PERSONAL_OLD", "05-05-1997", "PERSONAL_DATA") = 0 then
     /* Do processing */
End-If;
```

To open an existing tree, use the following code:

```
&MYTREE = %Session.GetTree();
&MYTREE.Open("","","PERSONAL_OLD", "05-05-1997", "PERSONAL_DATA", True);
```

GetTreeStructure

Syntax

GetTreeStructure()

Description

The GetTreeStructure method returns a closed tree structure object. Before you can use some of the methods or any of the properties, you must use the Open method to open the tree structure object.

Branch Collection

A branch collection is returned from the Branches property, used with an open tree object.

```
&MYBRANCHCOLL = &MYTREE.Branches;
```

&MYTREE must be a branched tree. To verify whether a tree is branched, you can check the value of the IsBranched property of a tree object (Boolean value: True or False.)

See Chapter 40, “Tree Classes,” Branches, page 1676.

Branch Collection Property

In this section, we discuss the Branch collection properties. The properties are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

Count

Description

Returns the number of branches for the branch collection object.

Leaf Class

Leaf objects are instantiated from other tree classes as follows:

- From a tree object with the FindLeaf method.
- From a node object with the FirstChildLeaf property.
- From another leaf object with the NextSib and PrevSib properties.

See Chapter 40, “Tree Classes,” FindLeaf, page 1661; Chapter 40, “Tree Classes,” FirstChildLeaf, page 1648; Chapter 40, “Tree Classes,” NextSib, page 1627 and Chapter 40, “Tree Classes,” PrevSib, page 1628.

Leaf Class Methods

In this section, we discuss the Leaf class methods. The methods are discussed in alphabetical order.

Cut

Syntax

```
Cut()
```

Description

The Cut method cuts the leaf and puts it into a buffer (the clipboard.) You can then use the PasteSib method to add the leaf back as a sibling of a different leaf. You can also use the PasteChild node method to add the leaf back as a child of a different node.

You can only have one object at a time on the clipboard. If you cut another leaf or node, the leaf in the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if the cut is successful.

See Also

Chapter 40, “Tree Classes,” Cut, page 1635, Chapter 40, “Tree Classes,” PasteSib, page 1625 and Chapter 40, “Tree Classes,” PasteChild, page 1643.

Copyright © 1988-2007, Oracle. All rights reserved.

Delete

Syntax

Delete()

Description

The Delete method deletes the leaf object executing the method *from the database*.

Returns

A number; 0 if the delete is successful.

Example

```
&MYLEAF = &MYTREE.FindLeaf("8200", "8300");
&MYLEAF.Delete();
```

DeleteByRange

Syntax

DeleteByRange(*RangeFrom*, *RangeTo*)

Description

The DeleteByRange method deletes the specified leaf object from the database. The leaf specified by the parameters does *not* have to match the leaf executing the method. That is, if &MYLEAF is associated with a range 8200-8300, you can specify a different range with the DeleteByRange method. For example:

```
&MYLEAF = &MYTREE.FindLeaf("8200", "8300");
/* some processing */
&RET_VALUE = &MYLEAF.DeleteByRange("8400", "8500");
```

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the leaf to be deleted. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf to be deleted. This parameter takes a string value.

Returns

A number; 0 if the delete is successful.

Example

```
&RET_VALUE= &MYLEAF.DeleteByRange("8400", "8500");
```

InsertDynSib

Syntax

InsertDynSib()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The InsertDynSib method inserts a dynamic leaf as a sibling leaf to the leaf object executing the method. The leaf is a *new*node.

A leaf object associated with the new leaf is returned. The new leaf is inserted as the next sibling leaf. If the new leaf isn't inserted successfully, the method returns Null.

Note. To insert a sibling leaf with a specific range, use the InsertSib method.

Parameters

None.

Returns

A reference to the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLEAF.InsertDynSib();
```

See Also

Chapter 40, "Tree Classes," InsertSib, page 1621
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None.

InsertSib

Syntax

InsertSib(*RangeFrom*, *RangeTo*)

Description

The InsertSib method inserts a new leaf as a sibling leaf under the leaf currently executing the method. The leaf specified by the range parameters must be a *new* leaf. You receive an error if the leaf already exists.

A leaf object associated with the new leaf is returned. The new leaf is inserted as the next sibling leaf, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields.

Note. To insert a dynamic sibling leaf (that is, without specifying a range) use the InsertDynSib method.

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the new leaf. This parameter takes a string value.
RangeTo	Specify the ending range of the new leaf. This parameter takes a string value.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A leaf object associated with the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLEAF.InsertSib("10090", "10100");
```

See Also

Chapter 40, "Tree Classes," InsertDynSib, page 1620
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None.

MoveAsChild

Syntax

MoveAsChild(*Node*)

Description

The MoveAsChild method moves the leaf executing the method to a different node in the tree. The leaf becomes the first child leaf under the node, even though there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields. The specified node becomes the new parent to the leaf.

Parameters

Parameter	Description
Node	Specify a node object. This parameter value must be an object, not a string or a name.

Note. To specify a node name, not a node object, use the MoveAsChildByName method.

Returns

A number; 0 if the move is successful.

Example

The following example moves leaf 8001 from node 10100 (old parent) to node 00001 (new parent)

```
&MY_LEAF = &MY_TREE.FindLeaf("8001", "8001");
&NEW_PARENT = &MY_TREE.FindNode("00001", "");
If &NEW_PARENT <> Null Then
   &RET_VALUE = &MY_LEAF.MoveAsChild(&NEW_PARENT);
End-If;
```

See Also

Chapter 40, "Tree Classes," MoveAsChildByName, page 1623

 Copyright © 1988-2007, Oracle. All rights reserved.

MoveAsChildByName

Syntax

MoveAsChildByName(*NodeName*)

Description

The MoveAsChildByName method moves the leaf executing the method to a different node in the tree. The leaf becomes the first child leaf under the node, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields. The specified node becomes the new parent to the leaf. The node specified by *NodeName* must be a valid node name.

Parameters

Parameter	Description
NodeName	Specify the name of a node. *NodeName* must be a valid node for the existing tree. This parameter takes a string value.

Note. To specify a node object, not a node name, use the MoveAsChild method.

Returns

A number; 0 if the move is successful.

Example

The following example moves leaf 8001 from node 10100 (old parent) to node 00001 (new parent)

```
&MY_LEAF = &MY_TREE.FindLeaf("8001", "8001");
If &MY_LEAF <> Null Then
   &RET_VALUE = &MY_LEAF.MoveAsChildByName("00001");
End-If;
```

See Also

Chapter 40, “Tree Classes,” MoveAsChild, page 1622

MoveAsSib

Syntax

MoveAsSib(*Leaf*)

Description

The MoveAsSib method moves the leaf executing the method to a new place in the tree. The current leaf becomes the next sibling leaf under the specified leaf object, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Leaf	Specify a leaf object. This parameter value must be an object, not a string or a name.

Note. To specify a range, not a leaf object, use the MoveAsSibByRange method.

Returns

A number; 0 if the move is successful.

Example

```
&MYLEAF = &MYTREE.FindLeaf("8000", "8000");
&MYLEAF2 = &MYTREE.FindLeaf("9000", "9000");
&RET_VALUE = &MYLEAF.MoveAsSib(&MYLEAF2);
```

See Also

Chapter 40, "Tree Classes," MoveAsSibByRange, page 1624

MoveAsSibByRange

Syntax

MoveAsSibByRange(*RangeFrom*, *RangeTo*)

Description

The MoveAsSibByRange method moves the leaf executing the method to a new place in the tree. The current leaf becomes the next sibling leaf under the specified leaf object.

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the leaf that you want as the parent of the leaf executing the method. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf that you want as the parent of the leaf executing the method. This parameter takes a string value.

Note. To specify a leaf object, not a range, use the MoveAsSib method.

Returns

A number; 0 if the move is successful.

Example

```
&MYLEAF = &MYTREE.FindLeaf("8000", "8000");
&RET_VALUE = &MYLEAF.MoveAsSibByRange("9000", "9000");
```

See Also

Chapter 40, "Tree Classes," MoveAsSib, page 1623

 Copyright © 1988-2007, Oracle. All rights reserved.

PasteSib

Syntax

```
PasteSib()
```

Description

The PasteSib method makes the leaf from the buffer (clipboard) a sibling to the leaf executing the method. You must use the Cut leaf method before you can paste a leaf.

You can have only one object at a time on the clipboard. If you cut another leaf or node, the leaf in the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if the paste is successful.

See Also

Chapter 40, “Tree Classes,” Cut, page 1619 , Chapter 40, “Tree Classes,” PasteChild, page 1643 and Chapter 40, “Tree Classes,” Cut, page 1635. node method

UpdateRanges

Syntax

```
UpdateRanges(RangeFrom, RangeTo)
```

Description

The UpdateRanges method performs validation of edited ranges, updating as necessary.

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the leaf that you updated. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf you want updated. This parameter takes a string value.

Returns

A number; 0 if the update is successful.

Leaf Class Properties

In this section, we discuss the Leaf class properties. The properties are discussed in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

DisplayLevelNumber

Description

When using the Tree API to create a graphic representation of the tree (such as for HTML Tree Manager or other application pages) use this property to indicate the *display* level, that is, tell the user how many levels deep the leaf is. This is generally used for trees where levels aren't used and the LevelNumber property always returns 0. This property always returns a number.

This property is read-only.

See Also

Chapter 40, "Tree Classes," LevelNumber, page 1651

Dynamic

Description

This property specifies whether the leaf has a dynamic range or a specified range. If you set this property to True, the leaf has a dynamic range. If this is a new leaf, and you do not set this property, the value is automatically set is False.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Creating Trees"

HasNextSib

Description

This property returns True if the leaf has a next sibling, that is, it isn't the last leaf listed under the parent node.

This property is read-only.

HasPrevSib

Description

This property returns True if the leaf has a previous sibling, that is, it isn't the first leaf listed under the parent node.

This property is read-only.

ImageName

Description

This property sets the which image is used to display the leaf. This property takes a string value for the image name of the leaf.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

IsChanged

Description

This property returns True if the leaf has been edited or changed.

This property is read-only.

Example

```
If &MYLEAF.IsChanged Then
   &MYTREE.Save();
End-If;
```

IsCut

Description

This property returns True if the leaf has been cut from the displayed tree. This property is generally used with the HTML Tree Manager.

This property is read-only.

IsDeleted

Description

This property returns True if the leaf has been deleted from the tree but the tree hasn't been saved.

This property is read-only.

IsInserted

Description

This property returns True if the leaf has been inserted as a new leaf in the tree but the tree hasn't been saved.

This property is read-only.

NextSib

Description

This property returns a leaf object associated with the next sibling leaf. The next sibling of a leaf is the leaf appearing under the current leaf. If there is no next sibling and you try to assign it to a variable, you receive a runtime error.

This property is read-only.

Example

The following code traversed the leaves from top to bottom.

```
While &MYLEAF.HasNextSib
   &MYLEAF = &MYLEAF.NextSib;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   /* do some processing */
End-While;
```

Parent

Description

This property returns a node object associated with the parent node for the leaf.

This property is read-only.

Example

```
&PARENTNODE = &MYLEAF.Parent;
/* do node processing with node object */
```

PrevSib

Description

This property returns a leaf object associated with the previous sibling leaf. The previous sibling of a leaf is the leaf appearing above the current leaf. If there is no previous sibling and you try to assign it to a variable, you receive a runtime error.

This property is read-only.

Example

The following code traverses the leaves from bottom to top.

```
While &MYLEAF.HasPrevSib
   &MYLEAF = &MYLEAF.PrevSib;
   /* do some processing */
End-While;
```

RangeFrom

Description

This property returns the starting range, as a string, of the leaf.

This property is read-write.

RangeTo

Description

This property returns the ending range, as a string, of the leaf.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

TreeBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property is read-only.

TreeEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not effective-dated, this property returns a blank string.

This property is read-only.

TreeName

Description

This property returns the name of the tree as a string.

This property is read-only.

TreeSetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property is read-only.

TreeUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

This property is read-only.

Level Collection

Level collection is instantiated from a tree object with the Levels property.

See Chapter 40, "Tree Classes," Levels, page 1683.

Copyright © 1988-2007, Oracle. All rights reserved.

Level Collection Methods

In this section, we discuss the Level Collection methods. The methods are discussed in alphabetical order.

Add

Syntax

Add(*LevelName*)

Description

The Add method adds a new level called *LevelName* to the database. The specified level must be a new level, that is, it cannot exist in the database. *LevelName* takes a string value.

Note. *LevelName* must be 8 characters or less.

If no levels currently exist in the tree, the level is added at the top level. If levels already exist in the tree, the new level is added as the last level. You can change the level number of a level using the Number property.

If the new level is the first level, the AllValuesAudit property is automatically set to True.

The new level is notadded to the database until the tree is explicitly saved.

This method returns a reference to the new level object.

See Also

Chapter 40, “Tree Classes,” Number, page 1633 and Chapter 40, “Tree Classes,” AllValuesAudit, page 1632.

Item

Syntax

Item(*LevelName, LevelNumber*)

Description

The Item method returns a reference to the specified level in the level collection executing the method as an object. The *LevelName* parameter specifies the name of the level to access. This parameter takes a string value. The *LevelNumber* parameter specifies the number at which the level exists. This parameter takes a number value. For example, suppose your level collection contains the following levels, in this order:

1. CORPORATE
2. COMPANY
3. DIVISION
4. DEPARTMENT
5. BRANCH

You want to access the fourth level, DEPARTMENT. You would use the following code:

```
&MYLEVEL = &LVLCOLLECTION.Item("DEPARTMENT", 4);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Remove

Syntax

```
Remove()
```

Description

The Remove method deletes the current level from the database. You can use this method only after you have used the First, Next, or Item properties: otherwise the system doesn't know which level to delete from the tree. If no level has been indicated yet system tries to remove the last level. If the level to remove has nodes associated with it, the system doesn't remove the level. The rest of the levels in the collection after the deleted level are moved up in the list and renumbered so that the levels remain consecutively numbered.

Returns

A number; 0 if the level is successfully removed.

Level Collection Properties

In this section, we discuss the Level collection properties. The properties are discussed in alphabetical order.

Count

Description

Returns the number of levels for the level collection object.

First

Description

The First property returns a reference to the first level in the level collection executing the method as an open object.

Last

Description

The Last property returns a reference to the last level in the level collection executing the method as an open object.

Next

Description

The Next property returns a reference to the next level in the level collection executing the method as an open object. You can use this method only after you have used either the First or Item properties: otherwise the system doesn't know where to start in the tree.

Copyright © 1988-2007, Oracle. All rights reserved.

Level Class

Level objects are instantiated from the level class collection object with the Item method or one of the following properties:

- First
- Last
- Next

See Chapter 40, "Tree Classes," Item, page 1630; Chapter 40, "Tree Classes," First, page 1631; Chapter 40, "Tree Classes," Last, page 1631 and Chapter 40, "Tree Classes," Next, page 1631.

Level Class Methods

In this section, we discuss the Leaf class properties. The properties are discussed in alphabetical order.

Create

Syntax

```
Create(LevelName, LevelNumber)
```

Description

Note. This method has been deprecated. If you create a level using this method, there is no way of saving the level to the database. Use the Add level collection method instead.

Level Class Properties

In this section, we discuss the Level class properties. The properties are discussed in alphabetical order.

AllValuesAudit

Description

This property specifies whether PeopleSoft Tree Manager permits nodes to skip over this level. To allow nodes to skip this level, specify this parameter as True. If you are creating a new level, and this level is the first level in a tree, this property is automatically set to True. If the level isn't the first level, this property is set to False by default..

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Using PeopleSoft Tree Manager," Modifying Tree Definitions

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Description

This property returns the description of the level.

This property is read-write.

Name

Description

This property returns the name of the level.

This property is read-write.

Number

Description

This property returns the number of the level.

Note. Though you can use this property to set the level of the number, you should not do so, as level numbers are automatically assigned when the level is inserted to the level collection using the Add method.

This property is read-write.

TreeBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property is read-only.

TreeEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not-effective dated, this property returns a blank string.

This property is read-only.

TreeName

Description

This property returns the name of the tree as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

TreeSetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property is read-only.

TreeUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

Node Class

Node objects are instantiated from other tree classes, as follows:

- From a tree object with the FindNode method.
- From a leaf object with the Parent property.
- From another node object with the FirstChildNode, NextSib, PrevSib, and Parent properties.

See Chapter 40, "Tree Classes," FindNode, page 1661; Chapter 40, "Tree Classes," FirstChildNode, page 1648; Chapter 40, "Tree Classes," NextSib, page 1651 and Chapter 40, "Tree Classes," PrevSib, page 1652.

See Chapter 40, "Tree Classes," Parent, page 1628.

See Chapter 40, "Tree Classes," Parent, page 1651.

Node Class Methods

In this section, we discuss the Node class methods. The methods are discussed in alphabetical order.

Branch

Syntax

```
Branch()
```

Description

The Branch method branches the node executing this method, identifying all of its child nodes and leaves as part of that branch. *Branching* means taking a limb of a tree and creating another subtree to hold that limb. (Technically is it not creating a real tree.) The subtree is accessed through the Branches collection.

 Copyright © 1988-2007, Oracle. All rights reserved.

After you use the Branch method, you can no longer access the child nodes and leaves of the node that executed the method until you close the current tree, open the new branched tree, and find the node again. To unbranch the tree, use the Unbranch method.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("10900", "");
&MYNODE.Branch();
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Creating Trees," Working with Tree Branches

Cut

Syntax

Cut()

Description

The Cut method cuts the node executing the method and puts it into a buffer (the clipboard). You can then use the PasteSib method to add the node back as a sibling of a different node. You can also use the PasteChild node method to add the node back as a child of a different node.

You can have only one object at a time on the clipboard. If you cut another leaf or node, the node on the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if method is successful.

See Also

Chapter 40, "Tree Classes," PasteSib, page 1644 , Chapter 40, "Tree Classes," PasteChild, page 1643 and Chapter 40, "Tree Classes," Cut, page 1619. leaf method

Delete

Syntax

Delete()

Description

The Delete method deletes the node object executing the method *from the database.*

Returns

A number; 0 if method is successful.

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteByName

Syntax

DeleteByName(*NodeName*)

Description

The DeleteByName method deletes the specified node from the database. The *NodeName* parameter takes a string value. The node specified by *NodeName* does not have to match the node executing the method. That is, if &MYNODE is associated with node 100200, you can specify a different node with the DeleteByName method. For example:

```
&MYNODE = &MYTREE.FindNode("100200", "");
/* some processing */
&MYNODE.DeleteByName("100300");
```

The node name specified by *NodeName* must be a valid node in the existing open tree object. If *NodeName* doesn't exist, you receive a runtime error.

Returns

A number; 0 if method is successful.

Expand

Syntax

Expand(*ExpandType*)

Description

The Expand method gets the next level of nodes or leaves into memory from the selected node. What leaves or nodes get expanded depends on the *ExpandType*.

Parameters

Parameter	Description
ExpandType	Specifies which leaves or nodes get expanded. Values are:

Value	Description
0	Expand only nodes
1	Expand nodes and leaves
2	Expand only one level

Returns

A number; 0 if method is successful.

Example

```
If (&MYNODE.State = 2 AND &MYNODE.HasChildren);
   /* if node is collapsed */
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   &MYNODE.Expand(2);
End-if;
```

InsertChildLeaf

Syntax

InsertChildLeaf(*RangeFrom*, *RangeTo*)

Description

The InsertChildLeaf method inserts a new leaf (as specified by the range parameters) under the node object executing the method.

A leaf object associated with the new leaf is returned.

The new leaf is inserted as the child of the current node, although there is no explicit ordering of leaves: leaves take the order of the alphanumeric database sort on their range fields.

Note. You must specify a range with this method. To insert a dynamic range leaf (that is, one with no range) use the InsertDynChildLeaf method.

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the new leaf. This parameter takes a string value.
RangeTo	Specify the ending range of the new leaf. This parameter takes a string value.

Returns

A leaf object associated with the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLNODE.InsertChildLeaf("10090", "10100");
If None(&NEWLEAF) Then
   /* Error Processing */
End-If;
```

See Also

Chapter 40, "Tree Classes," InsertDynChildLeaf, page 1639
Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None.

InsertChildNode

Syntax

InsertChildNode(*NodeName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The InsertChildNode method inserts a new node (as specified by *NodeName*) as a child node under the node object executing the method. The node specified by *NodeName* must be a new node. You receive an error if the node already exists.

A node object associated with the new node is returned. If the new node isn't inserted successfully, the method returns False or Null.

Parameters

Parameter	Description
NodeName	Specify a name for the new node. This parameter takes a string value. *NodeName* must not already exist.

Returns

A reference to the new node. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWNODE = &MYNODE.InsertChildNode("100500");
If None(&NEWNODE) Then
   /* Do error processing */
End-If;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None.

InsertChildRecord

Syntax

InsertChildRecord(*NodeName*)

Description

The InsertChildRecord method inserts a new record (as specified by *NodeName*) as a node under the parent node executing this method. This method works only on trees that are Query Access Trees.

A node object associated with the new node is returned, otherwise this method returns False or Null.

Parameters

Parameter	Description
NodeName	Specify an existing record name. This parameter takes a string value. This record must not already be a node under the parent node (that is, a parent node can't have the same record as a child node more than once.)

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to the new node. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWNODE = &MYNODE.InsertChildRecord("PERSONAL_DATA");
If None(&NEWNODE) Then
   /* Do error processing */
End-if;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None.

InsertDynChildLeaf

Syntax

InsertDynChildLeaf()

Description

The InsertDynChildLeaf method inserts a dynamic leaf under the node object executing the method.

A leaf object associated with the new leaf is returned.

The new leaf is inserted as the child of the current node, although there is no explicit ordering of leaves.

Note. To insert a leaf with a specific range, use the InsertChildLeaf method.

Parameters

None.

Returns

A leaf object associated with the new leaf. If this method fails, it returns either False or a Null object reference, depending upon the type of error encountered. The best way to test whether the object was inserted is to test the result using either the All or None functions.

Example

```
&NEWLEAF = &MYLNODE.InsertDynChildLeaf();
If None(&NEWLEAF) Then
   /* Do error processing */
End-If;
```

See Also

Chapter 40, "Tree Classes," InsertChildLeaf, page 1637

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," All and *Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference*, "PeopleCode Built-in Functions," None.

Copyright © 1988-2007, Oracle. All rights reserved.

InsertSib

Syntax

InsertSib(*NodeName*)

Description

The InsertSib method inserts a new node (as specified by *NodeName*) as a sibling node to the node object executing the method. The node specified by *NodeName* must be a *new* node. You receive an error if the node already exists.

A node object associated with the new node is returned. If the new node isn't inserted successfully, the method returns Null.

Parameters

Parameter	Description
NodeName	Specify a name for the new node. This parameter takes a string value. *NodeName* must not already exist.

Returns

A reference to the new node. If this method fails, it returns False.

Example

```
&NEWNODE = &MYNODE.InsertSib("100500");
```

InsertSibRecord

Syntax

InsertSibRecord(*NodeName*)

Description

The InsertSibRecord method inserts a new record (as specified by *NodeName*) as a sibling node of the parent node executing this method. This method works only on trees that are Query Access Trees.

A node object associated with the new node is returned, otherwise this method returns Null.

Parameters

Parameter	Description
NodeName	Specify an existing record name. This parameter takes a string value. This record must not already be a sibling node for the node executing the object (that is, a node can't have the same record as a node more than once.)

Returns

A reference to the new node. If this method fails, it returns False.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&NEWNODE = &MYNODE.InsertSibRecord("PERSONAL_DATA");
```

MoveAsChild

Syntax

MoveAsChild(*Node*)

Description

The MoveAsChild method moves the node executing the method to a different node in the tree. The node becomes the first child node under the named node. The specified node becomes the new parent to the child node. The node specified by *Node* must be an existing node in the current tree. All child nodes and details are also moved with the node.

Parameters

Parameter	Description
Node	Specify a node object. This parameter value must be an object, not a string or a name.

Note. To specify a node name, not a node object, use the MoveAsChildByName method.

Returns

A number; 0 if method is successful.

Example

The following example moves node 10200 from node 10100 (old parent) to node 00001 (new parent).

```
&MY_NODE = &MY_TREE.FindNode("10200", "");
&NEW_PARENT = &MY_TREE.FindNode("00001", "");
If &NEW_PARENT <> Null Then
   &MY_NODE.MoveAsChild(&NEW_PARENT);
End-If;
```

See Also

Chapter 40, “Tree Classes,” MoveAsChildByName, page 1641

MoveAsChildByName

Syntax

MoveAsChildByName(*NodeName*)

Description

The MoveAsChildByName method moves the node executing the method to a different node in the tree. The node becomes the first child node under the parent node. The specified node becomes the new parent to the node. The node specified by *NodeName* must be a valid node name. All child nodes and details are also moved with the node.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
NodeName	Specify the name of a node. This parameter takes a string value. *NodeName* must be a valid node for the existing tree.

Note. To specify a node object, not a node name, use the MoveAsChild method.

Returns

A number; 0 if method is successful.

Example

The following moves node 10200 from node 10100 (old parent) to node 00001 (new parent)

```
&MY_NODE = &MY_TREE.FindNode("10200", "");
If &MY_NODE <> Null Then
   &MY_NODE.MoveAsChildByName("00001");
End-If;
```

See Also

Chapter 40, "Tree Classes," MoveAsChild, page 1641

MoveAsSib

Syntax

MoveAsSib(*Node*)

Description

The MoveAsSib method moves the node executing the method to a new place in the tree. The current node becomes the next sibling node under the specified node. All child nodes and details are also moved with the node.

Parameters

Parameter	Description
Node	Specify a node object. This parameter value must be an object, not a string or a name.

Note. To specify a node name, not a node object, use the MoveAsSibByName method.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("20000", "");
&MYNODE2 = &MYTREE.FindNode("20100", "");
&MYNODE.MoveAsSib(&MYNODE2);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 40, “Tree Classes,” MoveAsSibByName, page 1643

MoveAsSibByName

Syntax
MoveAsSibByName(*NodeName*)

Description

The MoveAsSibByName method moves the node executing the method to a new place in the tree. The current node becomes the next sibling node under the specified node. All child nodes and details are also moved with the node.

Parameters

Parameter	Description
NodeName	Specify the name of a node. This parameter takes a string value. *NodeName* must be a valid node for the existing tree.

Note. To specify a node object, not a node name, use the MoveAsSib method.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("20000", "");
&MYNODE.MoveAsSibByName("10100");
```

See Also
Chapter 40, “Tree Classes,” MoveAsSib, page 1642

PasteChild

Syntax
PasteChild()

Description

The PasteChild method makes the node or leaf from the buffer (clipboard) a child to the node executing the method. You must use the Cut node or leaf method before you can paste a node or leaf.

You can have only one object at a time on the clipboard. If you cut another node or leaf, the node or leaf on the clipboard is overwritten.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A number; 0 if method is successful.

See Also

Chapter 40, “Tree Classes,” Cut, page 1619 , Chapter 40, “Tree Classes,” PasteSib, page 1644 and Chapter 40, “Tree Classes,” Cut, page 1635. node method

PasteSib

Syntax

```
PasteSib()
```

Description

The PasteSib method makes the node from the buffer (clipboard) a sibling to the node executing the method. You must use the Cut node method before you can paste a node.

You can have only one object at a time on the clipboard. If you cut another leaf or node, the node on the clipboard is overwritten.

Parameters

None.

Returns

A number; 0 if method is successful.

See Also

Chapter 40, “Tree Classes,” Cut, page 1619 , Chapter 40, “Tree Classes,” PasteChild, page 1643 and Chapter 40, “Tree Classes,” Cut, page 1635. node method

RefreshDescription

Syntax

```
RefreshDescription()
```

Description

The RefreshDescription method enables you to change the description of a node on a page and have the update be displayed immediately. If you don’t use RefreshDescription, you must save the tree and reopen it for the change to be displayed.

Parameters

None.

Returns

A zero (0) if description is refreshed successfully, a different error number otherwise.

Example

```
&result= &NodeObj.refreshdescription();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Rename

Syntax

Rename(*NodeName*)

Description

The Rename method renames the node object executing the method without changing any other properties. The parameter *NodeName* takes a string value. The node specified by *NodeName* must be a new node. You receive an error if the node already exists.

Note. The Rename method does not rename the node in a user node component. Your application needs to do that.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("10900", "");
&MYNODE.Rename("10200");
```

SwitchLevel

Syntax

SwitchLevel(*NewLevelNumber*)

Description

The SwitchLevel methods enables you to move a node down from one level to another.

NewLevelNumber must specify a valid level number. For trees in which the Level Use parameter is set to Strictly Enforce Levels, the new level must be at least one level lower than the node's parent level.

If the level specified by *NewLevelNumber* doesn't exist, it's automatically generated and added to the tree.

Parameters

Parameter	Description
NewLevelNumber	Specify the level number to where you want the node moved.

Returns

A zero (0) if node is moved successfully, a different error number otherwise.

Unbranch

Syntax

Unbranch()

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Unbranch method is the opposite of the Branch method: that is, it unbranches the node executing the method if it is branched. All of the child node and leaves of the node become part of the current open tree and accessible in that tree. You can't unbranch the Root Node in a branched tree.

If the node executing the method isn't branched, you receive a runtime error. Use the IsBranched property to make sure a node is branched.

Returns

A number; 0 if method is successful.

Example

```
&MYNODE = &MYTREE.FindNode("10900", "");
&MYNODE.Unbranch();
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Creating Trees," Working with Tree Branches

Node Class Properties

In this section, we discuss the Node class properties. The properties are discussed in alphabetical order.

AllChildCount

Description

This property returns the number of all the child nodes and leaves of the node, that is, all the nodes and leaves below this node.

This property is read-only.

AllChildNodeCount

Description

This property returns the number of child nodes of the node, that is, all the nodes below this node.

To determine all the leaves below a node, subtract the AllChildNodeCount from the AllChildCount

This property is read-only.

Example

To determine all the leaves below a node, use the AllChildNodeCount with the AllChildCount property.

```
&AllCount = (&MyNode.AllChildCount - &MyNode.AllChildNodeCount);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

ChildLeafCount

Description

This property returns the number of immediate child leaves below this node.

This property is read-only.

ChildNodeCount

Description

This property returns the number of immediate child nodes below this node.

This property is read-only.

ColImageName

Description

This property enables you to specify the image name for a collapsed node.

This property is read-write.

Description

Description

This property returns the description of the node.

This property is read-only.

DisplayLevelNumber

Description

When using the Tree API to create a graphic representation of the tree (such as for HTML Tree Manager or other application pages) use this property to indicate the display level, that is, tell the user how many levels deep the node is. This is generally used for trees where levels aren't used and the LevelNumber property always returns 0.

This property is read-only.

See Also

Chapter 40, "Tree Classes," LevelNumber, page 1651

ExpImageName

Description

This property enables you to specify the expanded image name for a node.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

FirstChildLeaf

Description

This property returns a reference to the first child leaf of the node. This is the leaf that appears highest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&NEWLEAF = &MYNODE.FirstChildLeaf;
```

FirstChildNode

Description

This property returns a reference to the first child node of the node. This is the node that appears highest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&CHILDNODE = &MYNODE.FirstChildNode;
```

HasChildLeaves

Description

This property returns True if the node has immediate child leaves, False otherwise.

This property is read-only.

HasChildNodes

Description

This property returns True if the node has immediate child nodes, False otherwise.

This property is read-only.

HasChildren

Description

This property returns True if the node has either child leaves or child nodes anywhere in the subtree, not just immediately under the node.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

HasNextSib

Description

This property returns True if the node has a next sibling, that is, if it isn't the last node.

This property is read-only.

Example

```
While &MYNODE.HasNextSib
   &MYNODE = &MYNODE.NextSib;
   /* do some processing */
End-While;
```

HasPrevSib

Description

This property returns True if the node has a previous sibling, that is, if it isn't the first node.

This property is read-only.

IsBranched

Description

This property returns True if the node is branched, False otherwise.

This property is read-only.

IsChanged

Description

This property returns True if the node has been edited or changed but the current tree hasn't been saved.

This property is read-only.

IsCut

Description

This property returns True if the node has been cut from the displayed tree. This property is generally used with the HTML Tree Manager.

This property is read-only.

IsDeleted

Description

This property returns True if the node has been deleted but the current tree hasn't been saved.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

IsInserted

Description

This property returns True if the node was inserted as a new node into the tree but the tree hasn't been saved.

This property is read-only.

Example

```
If &MYNODE.IsInserted Then
   &MYTREE.Save();
End-if;
```

IsRoot

Description

This property returns True for both of the following:

- the node is the root node of an unbranched tree.
- the node is the top node of an opened tree branch.

Otherwise, the property returns False.

This property is read-only.

LastChildLeaf

Description

This property returns a reference to the last child leaf of the node. This is the leaf that appears lowest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&NEWLEAF = &MYNODE.LastChildLeaf;
```

LastChildNode

Description

This property returns a reference to the last child node of the node. This is the node that appears lowest in the list of children of the node in PeopleSoft Tree Manager.

This property is read-only.

Example

```
&CHILDNODE = &MYNODE.LastChildNode;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

LevelNumber

Description

This property returns the level number of the node. Values are 1-99 for Strict or Loose level trees. This property returns 0 for trees that don't have levels.

Note. PeopleSoft does not recommend using this property to set the level of the node. Instead, use the SwitchLevel method.

This property is read-write.

See Also

Chapter 40, "Tree Classes," SwitchLevel, page 1645

Name

Description

This property returns the name of the node (as a string.) You must set this property to a valid value if you are creating a new node.

Note. Do not use this property to change the name of an existing node. The change will not be reflected in the database. You must use the Rename method to change the name of an existing node.

This property is read-write.

See Also

Chapter 40, "Tree Classes," Rename, page 1645

NextSib

Description

This property returns a reference to the next sibling node. If there isn't a next sibling node, Null is returned.

This property is read-only.

Example

```
While &MYNODE.HasNextSib
   &MYNODE = &MYNODE.NextSib;
   /* do some processing */
End-While;
```

Parent

Description

This property returns a reference to the node that is the parent of the node executing the property. If the current node has no parent (is a root node) False is returned.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&PARENT = &MYNODE.Parent;
```

PrevSib

Description

This property returns a reference to the node that is the previous sibling node to the node executing the property. If there isn't a previous sibling, Null is returned.

This property is read-only.

Example

```
If &MYNODE.PrevSib Then
   /* do processing */
End-if;
```

State

Description

This property returns the *state* of the node, that is, does it have children, and are they expanded or collapsed.

See Chapter 40, "Tree Classes," Expand, page 1636.

The values for this property are:

Value	Description
0	Node has no children
1	Children are expanded
2	Children are collapsed
3	Leaves are collapsed

This property is read-only.

Example

```
&VALUE = &MYNODE.State;
If &VALUE = 2 Then
   &MYNODE.Expand(0);
End-if;
```

TreeBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

TreeEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not effective-dated, this property returns a blank string.

This property is read-only.

TreeName

Description

This property returns the name of the tree as a string.

This property is read-only.

TreeSetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property is read-only.

TreeUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

Type

Description

This property returns the type of the node. This property takes a string value. Values are:

- "G" (Group): normal unbranched node or record group
- "B" (Branched nodes)
- "R" (Query Record): used for Query Access Trees only

This property is read-only.

Tree Class

Tree objects are instantiated from a session object with the GetTree method.

Copyright © 1988-2007, Oracle. All rights reserved.

The following code sample gets a tree, then opens the tree structure associated with that tree:

```
Local string &TREE_NAME;
Local string &TREE_DT;
Local ApiObject &MYSESSION, &VC_TREE, &STRUCT;


/* Get and Open the Tree using the Tree API */
&MYSESSION = %Session;

/* Get the Tree Name and Effective Date from the level 0 record */
&TREE_NAME = "CUSTOMER";
&TREE_DT = "1900-01-01";

/* Get and Open the Tree  */

&VC_TREE = &MYSESSION.GetTree();

/* open the tree with read access only */

&VC_TREE.OPEN("BKINV", "", &TREE_NAME, &TREE_DT, "", False);

/* Get and Open Tree Structure */

&STRUCT = VC_TREE.Structure;
```

See Chapter 40, “Tree Classes,” GetTree, page 1617.

Tree Class Methods

In this section, we discuss the Tree class methods. The methods are discussed in alphabetical order.

Audit

Syntax

Audit()

Description

The Audit method audits the tree object executing the method to determine its validity.

The Audit method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can audit it.

Returns

A number: 0 if the tree passes all audits. If Audit returns a value not equal to 0, an error is logged.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 40, "Tree Classes," Open, page 1665

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

AuditByName

Syntax

AuditByName(*SetID, UserKeyValue, TreeName, EffDt, BranchName*)

Description

The AuditByName method audits the tree specified by the parameters passed to it. The AuditByName method can be used only with a tree identifier, it cannot be used on a fully instantiated, open tree. Before you use the AuditByName method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

A number: 0 if the tree passes all audits. If this method returns a value not equal to 0, an error is logged.

Example

```
&ISVALID = &MYTREE.AuditByName("", "", "PERSONAL_DATA", "05-05-1997", "");
```

See Also

Chapter 40, "Tree Classes," Close, page 1656 and Chapter 40, "Tree Classes," IndirectionMethod, page 1696.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

Copyright © 1988-2007, Oracle. All rights reserved.

Close

Syntax

```
Close()
```

Description

The Close method closes the tree, freeing the memory associated with that object, and discarding any changes made to the tree since the last save. The Close method can be used only on an open tree, not a closed tree. This means you must have opened the tree with the Open method before you can close it. To save any changes, you must use the Save method before using Close.

Returns

A number: 0 if the method completed successfully.

See Also

Chapter 40, “Tree Classes,” Open, page 1665

Copy

Syntax

```
Copy(FromSetId, FromUserKeyValue, FromTreeName, FromEffDt, FromBranchName, ToSetId,
ToUserKeyValue, ToTreeName, ToEffDt, ToBranchName)
```

Description

The Copy method copies from the tree identified by the *From* parameters and copies it to the tree identified with the *To* parameters. The tree specified by the *To* parameters must be a *new* tree. You receive an error if the tree already exists. The tree specified by the *From* parameters does *not* have to match the tree identifier executing the method.

The Copy method can be used only with a *closed* tree, it cannot be used on an open tree. Before you use the Copy method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

To access the new tree, you must use the Open method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
FromSetId	Specify the table indirection key for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them (("")) for this parameter.
FromUserKeyValue	Specify the User Key Value for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them (("")) for this parameter.
FromTreeName	Specify the name for the tree to be copied from. This parameter takes a string value.
FromEffDt	Specify the effective date for the tree to be copied from. This parameter takes a string value.
FromBranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them (("")) for this parameter.
ToSetId	Specify the table indirection key for the tree to be copied to. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them (("")) for this parameter.
ToUserKeyValue	Specify the User Key Value for the tree to be copied to. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B" you must enter a Null string (that is, two quotation marks with no blank space between them (("")) for this parameter.
ToTreeName	Specify the name for the tree to be copied to. This parameter takes a string value.
ToEffDt	Specify the effective date for the tree to be copied to. This parameter takes a string value.
ToBranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them (("")) for this parameter.

Returns

An integer: 0 if copied successfully.

Example

```
&MYTREE.Copy("","","PERSONAL_DATA", "05-05-1997", "", "","","PERSONAL_EDIT1", "05-⇒
05-1997", "");
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 40, "Tree Classes," Close, page 1656, Chapter 40, "Tree Classes," Open, page 1665 and Chapter 40, "Tree Classes," IndirectionMethod, page 1696.

Create

Syntax

Create(*SetID, UserKeyValue, TreeName, EffDt, StructureName*)

Description

The Create method creates a new tree, based on the parameters passed with the method. The specified tree must be a new tree. You receive an error if the tree already exists.

The Create method can be used only with a closed tree, it cannot be used on an open tree. Before you use the Create method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

After you create a new tree, you don't have to open it with the Open method. The existing tree object points to the new tree.

The tree structure specified with *StructureName* determines what database fields the new tree is based on, what pages you can use to create tree nodes and detail values, and what record definitions PeopleSoft Tree Manager saves tree-related data in. The tree structure must already exist.

The new tree must be saved (Save, SaveAs, and so on) for this tree to be saved to the database.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
StructureID	Specify the name of the tree structure to use for this tree. This parameter takes a string value.

Returns

An integer: 0 if created successfully.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MYTREE.Create("","","PERSONAL_NEW", "05-05-1997", "PERSONAL_DATA");
```

See Also

Chapter 40, "Tree Classes," Close, page 1656, Chapter 40, "Tree Classes," IndirectionMethod, page 1696 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager*, "Creating Trees".

Delete

Syntax

Delete(*SetID*, *UserKeyValue*, *TreeName*, *EffDt*, *BranchName*)

Description

The Delete method deletes the specified tree from the database. The Delete method can be used only with a closed tree, it cannot be used on an open tree. Before you use the Delete method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

The specified tree does not have to match the tree identifier executing the method.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	Specify the name of the branch for the tree. This parameter takes a string value. If the tree is unbranched, you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

An integer: 0 if deleted successfully.

Example

```
&MYTREE=&Session.GetTree();

If ALL(&MYTREE) Then
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   &RETVALUE = &MYTREE.Delete("","","PERSONAL_REN2", "05-05-1997", "");
End-If;
```

See Also

Chapter 40, “Tree Classes,” Close, page 1656 and Chapter 40, “Tree Classes,” IndirectionMethod, page 1696.

Exists

Syntax

Exists(*SetID*, *UserKeyValue*, *TreeName*, *EffDt*, *BranchName*)

Description

The Exists method finds an existing tree, as specified by the parameters. The parameters must specify a unique tree. If a tree matching the parameters is found, the method returns 0. This method works with either open or closed trees.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn’t have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn’t have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

An integer: 0 if tree exists.

Example

```
&RSLT = &MYTREE.Exists("", "", "VENDOR_PER_DATA", "05-03-1998", "");

If &RSLT = 0 Then
   /* tree exists, do processing */
Else
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
   /* tree doesn't exist, do error processing */
End-if;
```

See Also

Chapter 40, "Tree Classes," IndirectionMethod, page 1696

FindLeaf

Syntax

FindLeaf(*RangeFrom*, *RangeTo*)

Description

The FindLeaf method finds an existing leaf (specified by the *range* parameters) in the tree executing this method. If the leaf is found, a leaf object is returned. If the leaf isn't found, a Null is returned.

You can use wildcards with the *range* parameters. Use an asterisk (*) in place of a number of characters. Use a question mark (?) in place of one character. You can also provide one of the ranges, and specify a Null string for the other.

Suppose you wanted to find the leaf with the range 81005 to 82000. The following would be valid:

```
&MyLeaf = &MyTree.FindLeaf("81*", "82000");
```

This would also be valid:

```
&MyLeaf = &MyTree.FindLeaf("8100?", "82000");
```

You could also specify this:

```
&MYLEAF = &MYTREE.FindLeaf("81000", "");
```

If you specify the exact same value for both *RangeFrom* and *RangeTo*, the *RangeTo* parameter is ignored.

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the leaf to be found. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf to be found. This parameter takes a string value.

Returns

A reference to a leaf object.

Example

```
&MYLEAF = &MYTREE.FindLeaf("81000", "");
```

FindNode

Syntax

FindNode(*NodeName*, *Description*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The FindNode method returns a reference to the node object specified by the parameters passed to the method. If the node is found, a node object is returned. If the node isn't found, a Null is returned.

Note. If you've only specified a node name, and the node you're searching for is in an unexpanded portion of the tree, this method expands the tree down to that node. If you've only specified a node description the search, occurs only in the memory. If the node isn't found, the tree is not expanded.

You should specify either the node name other description, not both. You should specify a Null string for the other. If both are provided, only the node name is used and the description is ignored.

You can use wildcards with the parameters. Use an asterisk (*) in place of a number of characters. Use a question mark (?) in place of one character.

Suppose you wanted to find the node called 10200, with a description of "Human Resources". The following would be valid:

```
&MyNode = &MyTree.FindNode("", "Human*");
```

This would also be valid:

```
&MyNode = &MyTree.FindNode("1020?", "");
```

You could also use the following:

```
&MYNODE = &MYTREE.FindNode("10200", "");
```

Note. If you're searching using wildcared characters and more than one node matches the search conditions, only the first occurence of the node is returned

---

Parameters

Parameter	Description
NodeName	Specify the name of the node that you want to find. This parameter takes a string value. Specify either the node name or the description.
Description	Specify the description of the node that you want to find. This parameter takes a string value. Specify either the node name or the description.

Returns

A reference to a node object.

Example

```
&MYNODE = &MYTREE.FindNode("10200", "");
```

FindRoot

Syntax

```
FindRoot()
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

The FindRoot method returns a reference to the root node object of the tree object executing the method. This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can find the root node.

This method returns an error if the tree has no nodes associated with it.

Returns

A reference to a node object.

Example

```
&MYNODE = &MYTREE.FindRoot();
```

See Also

Chapter 40, “Tree Classes,” Open, page 1665

InsertRoot

Syntax

InsertRoot(*NodeName*)

Description

The InsertRoot method inserts a node at the top of the tree object executing the method. *NodeName* is the name of the node you are creating. This parameter takes a string value. The InsertRoot method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can insert any nodes.

You can insert the root node only in a new tree, that is, a tree that was just created with the Create method. Also, after you create a new tree, you must insert the root node before you can insert any other nodes.

In a strict level tree, the first level must be defined before you can insert the root node.

This method returns an error if the tree already has nodes in it.

Returns

A reference to a node object.

Example

```
&MyRootNode = &MYTREE.InsertRoot("000001");
```

See Also

Chapter 40, “Tree Classes,” Open, page 1665

LeafExists

Syntax

LeafExists(*RangeFrom*, *RangeTo*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The LeafExists method enables you to verify if the leaf specified by the parameters exists in the current tree.

Suppose you wanted to check the leaf with the range 81005 to 82000 exists. The following would be valid:

```
&MyResult = &MyTree.LeafExists("81005", "82000");
```

This would also be valid, and would check on leaves up to 82000:

```
&MyResult = &MyTree.LeafExists("", "82000");
```

If you specify the exact same value for both *RangeFrom* and *RangeTo*, the *RangeTo* parameter is ignored.

Parameters

Parameter	Description
RangeFrom	Specify the starting range of the leaf you're verifying. This parameter takes a string value.
RangeTo	Specify the ending range of the leaf you're verifying. This parameter takes a string value.

Returns

An integer: 0 if the leaf exists.

See Also

Chapter 40, "Tree Classes," NodeExists, page 1668 and Chapter 40, "Tree Classes," FindLeaf, page 1661.

LockTree

Syntax

LockTree()

Description

Use the LockTree method to check out a tree for editing. Use this method to prevent saving conflicts when you have multiple users accessing the same tree, using both the Tree API and PeopleSoft Tree Manager. To release a tree, use the UnlockTree method.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

To verify if a tree is already locked, use the LockStatus property.

This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method.

Parameters

None.

Returns

An integer: 0 if locked successfully.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 40, “Tree Classes,” TreeLocksNumber, page 1673 , Chapter 40, “Tree Classes,” UnlockTree, page 1674 , Chapter 40, “Tree Classes,” HasLockedBranches, page 1679 , Chapter 40, “Tree Classes,” LockStatus, page 1684 and Chapter 40, “Tree Classes,” UseUpdateReservation, page 1689.

Open

Syntax

Open(*SetID, UserKeyValue, TreeName, EffDt, BranchName, Update*)

Description

The Open method opens the tree object specified by the parameters. The Open method can be used only with a closed tree, it cannot be used on an open tree. You cannot read or set any properties of a tree until after you open it. When you open a tree, you can specify whether you want to open it for update, or in read-only mode, with the *Update* parameter. If you try updating or writing to a tree opened in read-only mode, you receive an error.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn’t have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn’t have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
EffDt	Specify the effective date for the tree. This parameter takes a string value. The format must be in the correct format for the database platform the code is executing on.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
Update	Specify if you want to open the tree for update or in read-only mode. This parameter takes a Boolean value. Values for *Update* are: • True: open tree structure in update mode (read-write). • False: open tree structure in read-only mode. If you try updating or writing to a tree structure opened in read-only mode, you receive an error.

Returns

An integer: 0 if opened successfully.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following example opens a tree object, then tests to see if the tree was actually opened:

```
&RSLT = &MYTREE.Open("","","PERSONAL_DATA", "05-05-1997", "", True);

If &RSLT = 0 Then
   /* normal processing */
Else
   /* error processing tree isn&rsquo;t open */
End-if;
```

See Also

Chapter 40, "Tree Classes," Close, page 1656 and Chapter 40, "Tree Classes," IndirectionMethod, page 1696.

OpenForExport

Syntax

OpenForExport(*SetID*, *UserKeyValue*, *TreeNameEffDt*)

Description

Use the OpenForExport method to open a tree for export processes (using TreeMover.)

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
EffDt	Specify the effective date for the tree. This parameter takes a string value. The format must be in the correct format for the database platform the code is executing on.

Returns

An integer: 0 if opened successfully.

Example

```
If &TREE.OpenForExport(&SETID, &USERKEYVALUE, &TREE_NAME, &TREE_EFFDT) <> 0 Then
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
/* processing */

End-if;
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Using TreeMover," File Layout Details

OpenWholeTree

Syntax

OpenWholeTree(*SetID*, *UserKeyValue*, *TreeName*, *EffDt*)

Description

The OpenWholeTree method enables you to open a tree as if it were an unbranched tree. With this kind of tree, branched nodes cannot be distinguished from unbranched nodes. This means you can traverse the entire tree. It also means that branches have a node icon, not a branch icon. This method can be used only with a closed tree, it cannot be used on an open tree. You cannot read any properties of a tree until after you open it. This tree is always opened as read-only.

Note. You can't open a tree using this method and update it. The tree is opened in read-only mode.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for the tree. This parameter takes a string value.
EffDt	Specify the effective date for the tree. This parameter takes a string value. The format must be in the correct format for the database platform the code is executing on.

Returns

An integer: 0 if opened successfully.

Example

```
&pSession = %Session;

&pTreeObj = &pSession.GetTree();
&status = &pTreeObj.OpenWholeTree(&Setid, &SetCntrlValue, &TreeName, String(&Eff⇒
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Dt));
```

See Also
Chapter 40, "Tree Classes," Close, page 1656 and Chapter 40, "Tree Classes," IndirectionMethod, page 1696.

NodeExists

Syntax
NodeExists(*NodeName*)

Description

The NodeExists method enables you to verify if the node specified by *NodeName* exists in the current tree.

Note. If the node you're searching for is in an unexpanded portion of the tree, this method expands the tree down to that node.

Parameters

Parameter	Description
NodeName	Specify the name of the node that you want to find. This parameter takes a string value.

Returns

An integer: 0 if the node exists.

See Also
Chapter 40, "Tree Classes," FindLeaf, page 1661, Chapter 40, "Tree Classes," FindNode, page 1661 and Chapter 40, "Tree Classes," LeafExists, page 1663.

Rename

Syntax
Rename(*FromSetId, FromUserKeyValue, FromTreeName, FromEffDt, FromBranchName, ToTreeName*)

Description

The Rename method renames a tree specified with the *From* parameters to the tree specified with *ToTreeName*. The tree specified by *ToTreeName* must be a new tree. You receive an error if the tree already exists. The tree specified by the *From* parameters does not have to match the tree executing the method.

The Rename method can be used only with a closed tree, it cannot be used on an open tree. Before you use the Rename method, you must explicitly close any open tree objects (with the Close method.) You receive an error if there are any open trees.

To access the new tree, you must use the Open method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
FromSetId	Specify the table indirection key for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
FromUserKeyValue	Specify the User Key Value for the tree to be copied from. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B" you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
FromTreeName	Specify the name for the tree to be copied from. This parameter takes a string value.
FromEffDt	Specify the effective date for the tree to be copied from. This parameter takes a string value.
FromBranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
ToTreeName	Specify the new name of the tree. This parameter takes a string value.

Returns

An integer: 0 if renamed successfully.

Example

```
&MYTREE.Rename("","","PERSONAL_DATA3", "05-05-1997","", "PERSONAL_REN2");
```

See Also

Chapter 40, "Tree Classes," Close, page 1656, Chapter 40, "Tree Classes," Open, page 1665, Chapter 40, "Tree Classes," Close, page 1656 and Chapter 40, "Tree Classes," IndirectionMethod, page 1696.

Save

Syntax

Save()

Description

The Save method writes any changes to the tree executing the method to the database. It also performs audits.

The Save method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can save it.

The tree object remains open after executing Save. You must execute the Close method on the object before it is closed and the memory freed.

Copyright © 1988-2007, Oracle. All rights reserved.

This method returns an optional Boolean value: True if the tree was saved with no errors, False if there are one or more errors.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

Parameters

None.

Returns

An integer: 0 if save successfully.

SaveAs

Syntax

SaveAs(*SetID*, *UserKeyValue*, *TreeName*, *EffDt*, *BranchName*)

Description

The SaveAs method writes any changes to the tree executing the method to the new tree specified with the parameters. It also performs audits.

The tree specified with the parameters must be a new tree. You receive an error if the tree already exists. The new tree must have at least one different key field from the original tree.

The SaveAs method automatically closes the tree that is associated with the tree object executing the method, and associates that tree object with the new tree. Any changes made to the original tree since the last time the tree was saved are discarded. For example, suppose you opened a tree named DEPENDENT_BENEF and associated it with the variable &MYTREE. If you executed the SaveAs method with &MYTREE, changing the name to DEPENDENT_BENEF2, &MYTREE would now be associated with DEPENDENT_BENEF2, and the original tree, DEPENDENT_BENEF, would be closed.

```
&MYTREE.Open("", "", "DEPENDENT_BENEF", "02-02-1999", "", False);
/* do some processing */
&MYTREE.SaveAs("", "", "DEPENDENT_BENEF2", "02-05-1999", "");
```

&MYTREE is now associated with DEPENDENT_BENEF2, not with the original open tree.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a COMMIT.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

This method returns an integer: 0 if saved successfully, a number greater than 0 otherwise.

Example

```
&MYTREE.SaveAs("", "", "DEPENDENT_BENEF2", "02-05-1999", "");
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

SaveAsDraft

Syntax

SaveAsDraft(*SetID*, *UserKeyValue*, *TreeName*, *EffDt*, *Branch'Name*)

Description

The SaveAsDraft method writes any changes to the tree executing the method to the new tree specified with the parameters. However, it does not perform audits. Trees that are saved in this way are marked as "draft" and will not be valid (that is, cannot be used with other PeopleTools) until the tree audit is passed during Save, SaveAs, or the Tree Audits PeopleSoft Application Engine Tree Utility program..

The tree specified with the parameters must be a new tree. You receive an error if the tree already exists. The new tree must have at least one different key field from the original tree.

The SaveAsDraft method automatically closes the tree that is associated with the tree object executing the method, and associates that tree object with the new tree. Any changes made to the original tree since the last time the tree was saved are discarded.

See SaveAs for more details.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a Commit.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.
BranchName	This parameter is required, but it is unused in this release. You must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.

Returns

This method returns an integer: 0 if saved successfully, a number greater than 0 otherwise.

Example

```
&MYTREE.  SaveAsDraft("","","PERSONAL_EDIT2", "05-05-1997", "");
```

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

SaveDraft

Syntax

SaveDraft()

Description

The SaveDraft method writes any changes to the tree executing the method to the database. However, it does not perform audits. Trees that are saved in this way are marked as "draft" and will not be valid (that is, cannot be used with other PeopleTools) until the tree audit is passed.

The SaveDraft method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method before you can save it.

 Copyright © 1988-2007, Oracle. All rights reserved.

The tree object remains open after executing SaveDraft. You must execute the Close method on the object before it is closed and the memory freed.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a Commit.

Parameters

None.

Returns

This method returns an integer: 0 if saved successfully, a number greater than 0 otherwise.

See Also

Chapter 40, "Tree Classes," Close, page 1656 and Chapter 40, "Tree Classes," Open, page 1665.

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

SetImportMode

Syntax

```
SetImportMode(Nodes_NumberLeaves_Number)
```

Description

Use the SetImportMode method to set up special memory allocation and simplified processing rules when running Tree Import.

This special mode also cuts back on the number of SQL calls by not getting node descriptions. This mode also blocks internal validation while inserting new node or leaf from an input file. All validation occurs during saving.

Parameters

Parameter	Description
Nodes_Number	Specify the number of nodes.
Leaves_Number	Specify the number of leaves.

Returns

An integer: 0 if set successfully.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Using TreeMover," File Layout Details

TreeLocksNumber

Syntax

```
TreeLocksNumber(setID, UserKeyValue, TreeName, EffDt)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the TreeLocksNumber method to determine how many users have checked out any branches of a current tree. This method should be used to check if a tree definition could be changed without affecting other users working in different branches of a tree.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

You can use the TreeLocksNumber method with a both an open as well as a closed tree.

Parameters

Parameter	Description
SetID	Specify the table indirection key for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "S", you must specify a SetID. If the tree structure doesn't have its IndirectionMethod specified as "S", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
UserKeyValue	Specify the User Key Value for the tree. This parameter takes a string value. If the tree structure the tree is based on has its IndirectionMethod specified as "U" or "B", you must specify a User Key Value. If the tree structure doesn't have its IndirectionMethod specified as "U" or "B", you must enter a Null string (that is, two quotation marks with no blank space between them ("")) for this parameter.
TreeName	Specify the name for this tree. This parameter takes a string value.
EffDt	Specify the effective date for this tree. This parameter takes a string value.

Returns

The number of locked branches. If the tree is not branched, but is checked out, the return value is 1.

See Also

Chapter 40, "Tree Classes," LockTree, page 1664 , Chapter 40, "Tree Classes," UnlockTree, page 1674 , Chapter 40, "Tree Classes," UnlockTree, page 1674 and Chapter 40, "Tree Classes," UseUpdateReservation, page 1689.

UnlockTree

Syntax

```
UnlockTree()
```

Description

Use the UnlockTree method to release a tree after editing. Use this method to prevent saving conflicts when you have multiple users accessing the same tree, using both the Tree API and PeopleSoft Tree Manager. To checkout a tree, use the LockTree method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

To verify if a tree is checked out, use the LockStatus property.

This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method.

Parameters

None.

Returns

An integer: 0 if the tree is released successfully.

See Also

Chapter 40, “Tree Classes,” LockTree, page 1664 , Chapter 40, “Tree Classes,” LockStatus, page 1684 and Chapter 40, “Tree Classes,” UseUpdateReservation, page 1689.

UpdateLock

Syntax

```
UpdateLock()
```

Description

Use the UpdateLock method to update the timestamp representing time when the user who has the tree checked out performed the last action on a tree.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

Use this method to indicate that user who has the tree checked out is still working with it. The tree won’t be released unless the Tree Inactivity Period specified on the People Tools Options page is expired.

This method should be called from PeopleCode in a response to any user action happened on a Tree Manager page.

This method can be used only on an open tree, not on a closed tree. This means you must have opened the tree with the Open method.

Parameters

None.

Returns

An integer: 0 if activity time was updated successfully.

See Also

Chapter 40, “Tree Classes,” LockTree, page 1664 , Chapter 40, “Tree Classes,” UnlockTree, page 1674 and Chapter 40, “Tree Classes,” UseUpdateReservation, page 1689.

Copyright © 1988-2007, Oracle. All rights reserved.

Tree Class Properties

In this section, we discuss the Tree class properties. The properties are discussed in alphabetical order.

AllValues

Description

When the tree is audited, the AllValues property specifies whether the audit should verify that the tree includes all user data detail values. This property returns a Boolean value: True, check all values, False, do not check all values.

If this is a new tree, and you do not set this property, a default value of False is automatically set.

This property can be used only with an open tree, that is, you must use the Open method on the tree before you can use this property.

This property is read-write.

Example

```
&MyTree.AllValues = False;
```

AuditDetails

Description

Set this property to False in cases when you do not want to perform Detail Audit. It could happend if you are reusing existing structure that contains Detail information for tree that does not use details('Winter' tree).

The default value for this property is True.

This property is read-write.

Branches

Description

This property returns a reference to a branch collection object. This property is used only with branched trees: it returns Null for trees that aren't branched.

This property can be used only with an open tree, that is, you must use the open the tree using the Open method before you can use this property.

This property is read-only.

BranchImageName

Description

Use this property to specify the image used to display branches on a page. This property takes a string value.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

BranchLevel

Description

This property returns the level number (as a number) where this branch occurs in the tree. This property is used only with trees with levels: it returns zero for trees that don't have levels.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
&LevelNumber = &MyTree.BranchLevel;
```

BranchName

Description

This property returns the name of the specific branch for the tree, as a string. This property is used only with branched trees: it returns an empty string for trees that aren't branched.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
&BranchName = &MyTree.BranchName;
```

Category

Description

This property returns the category name for the tree, as a string. The category is a high-level grouping under which you can organize your tree structures and tree definitions. If you are creating a new tree, this property is required.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&MyTree.Category = "PurchaseOrders";
```

Description

Description

This property returns the description of the tree, as a string. If you are creating a new tree, this property is required.

Copyright © 1988-2007, Oracle. All rights reserved.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&MyTree.Description = "Departmental Security";
```

DuplicateLeaves

Description

In most trees, you want each detail value to appear only once in the tree. To permit duplicate values, set this property to True.

If this is a new tree, and you do not set this property, a default value of False is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

EffDt

Description

This property returns the effective date of the tree, as a date.

Note. PeopleSoft does not recommnd using this property to change data, as changes are not stored in the database.

If this is a new tree, and you do not set this property, a default value of the current date is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&TreeEffDt = &MYTREE.EffDt;
```

HasDetailRanges

Description

This property returns True if the tree has any detail ranges.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

HasLockedBranches

Description

This property indicates whether any user other then current one has checked out any branch in a tree. This property returns a Boolean value: true if the tree has checked out branches, false otherwise.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

IsBranched

Description

This property indicates whether the tree is branched. Possible values for this property are True and False: True if the tree is branched, False if it isn't.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
If &MyTree.IsBranched Then
   /* Do branched processing */
Else
   /*Do unbranched processing */
End-If;
```

IsChanged

Description

This property returns True if the tree has been changed but not saved, False if the tree is unchanged.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
If &MyTree.IsChanged Then
   &MyTree.Save();
End-if;
&MyTree.Close();
```

Copyright © 1988-2007, Oracle. All rights reserved.

IsOpen

Description

This property returns True if the tree is open, False if the tree is closed.

This property is read-only.

Example

```
If Not &MyTree.IsOpen Then
   &MyTree.Open("","","PERSONAL_DATA", &Date, "", True);
End-If;
```

IsQueryTree

Description

This property returns True if the tree is a Query access tree. It returns False if the tree is not a Query Access Tree.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

IsValid

Description

This property returns True if the tree is a valid tree that has passed all audits. It returns False otherwise.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, "Auditing and Repairing Trees"

IsVersionChanged

Description

This property indicates if a tree version has been changed in the database. This property returns a Boolean value, true if the version was changed, false otherwise.

Use this function in a multi-user environment to check if a tree loaded in a read-only mode by User1 should be reloaded. A possible reason why it needs to be reloaded could be triggered by tree version change after User2, who had this tree checked out, released it.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

IsWholeTree

Description

This property indicates whether the entire tree is accessible. This property is set to True if a tree was opened using OpenWholeTree method. The OpenWholeTree method opens a tree in read-only mode without paying attention to the existing tree branches.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

Example

```
If &MyTree.IsWholeTree Then
   /* Do processing */
Else
   /* close tree and reopen using OpenWholeTree method */
End-If;
```

KeyBranchName

Description

This property returns the branch name of the tree as a string if the tree is branched. If not branched, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeyBranchName property to get the tree branch name

This property is read-only.

KeyEffDt

Description

This property returns the effective date of the tree as a string if the tree is effective-dated. If not effective-dated, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeyEffDt property to get the tree effective date

This property is read-only.

KeyName

Description

This property returns the name of the tree as a string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the Name property to get the tree name.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

KeySetId

Description

This property returns the SetID of the tree as a string if the tree has a SetID. If the tree doesn't have a SetID, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeySetId to get the tree Set Id.

This property is read-only.

KeyUserKeyValue

Description

This property returns the UserKeyValue of the tree as a string if the tree has a UserKeyValue. If the tree doesn't have a UserKeyValue, this property returns a blank string.

This property can be used with a closed tree, that is, before you use the Open method.

If the tree has already been opened, you can use the KeyUserKeyValue to get the tree UserKeyValue.

This property is read-only.

LeafCount

Description

This property returns the number of leaves in the tree.

This property can be used only with an open tree, that is, you must use the Open method on the tree before you can use this property.

This property is read-only.

LeafImageName

Description

Use this property to specify the image used to display leaves on a page. This property takes a string name.

This property is read-write.

LeafOnClipboard

Description

Use this property to determine if a leaf has been cut and is available to be 'pasted' into tree. This property returns a Boolean value: True if a leaf object exists on the clipboard, False otherwise.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

LevelCount

Description

This property returns the number of levels defined for the tree. If the tree doesn't have any levels, this property returns zero (0).

See the LevelUse property.

Valid value range is a number between 0 and 99.

This property can only be used with an open tree, that is, you must use the Open method on the tree before you can use this property.

This property is read-only.

See Also

Chapter 40, "Tree Classes," LevelUse, page 1683

Levels

Description

This property returns a reference to a level collection object. This property is used only with trees that have levels: it returns Null for trees that don't have levels.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

See Also

Chapter 40, "Tree Classes," Level Collection Methods, page 1630

LevelUse

Description

This property returns the level use of the tree. The values are:

Value	Description
"S"	Strict levels
"L"	Loose levels
"N"	No levels

If this is a new tree, and you do not set this property, the default value is Strict Levels.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
If &MyTree.LevelUse = "S" Then
/* add levels */
End-If;
```

LockOwner

Description

This property returns the user ID of the user who has checked out a specific tree or a tree branch, as a string.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

LockStatus

Description

This property indicates if an opened tree is checked out.

Note. In order to use this method, you must have Tree multi-user environment enabled. You can check if it is enabled by using UseUpdateReservation Tree property.

Possible values are:

Value	Description
0	Tree is not checked out by any user.
1	Tree is checked out by another user.
2	Tree is checked out by the current user and is editable.

This property is read-only.

See Also

Chapter 40, “Tree Classes,” LockTree, page 1664 , Chapter 40, “Tree Classes,” TreeLocksNumber, page 1673 , Chapter 40, “Tree Classes,” UnlockTree, page 1674 and Chapter 40, “Tree Classes,” LockOwner, page 1684.

Name

Description

This property returns the name of the tree, as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. Do not use this property to change the name of an existing tree. The change will not be reflected in the database. You must use the Rename method to change the name of an existing tree.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Example

```
&TreeName = &MyTree.Name;
```

See Also

Chapter 40, "Tree Classes," Rename, page 1668

NodeColImageName

Description

Use this property to specify the image displayed for a collapsed node. This property takes a string value.

This property is read-write.

NodeCount

Description

This property returns the number of nodes for the tree. A valid tree must have at least one node.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

NodeExpImageName

Description

Use this property to specify the displayed image of an expanded node. This property takes a string value.

This property is read-write.

NodeOnClipboard

Description

Use this property to determine if a node has been cut and is available to be 'pasted' into a tree. This property returns a Boolean value: True if a node object exists on the clipboard, False otherwise.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

ParentLevel

Description

This property returns the number of the parent branch level in a branched tree. If the tree does not have levels or is an unbranched tree, zero (0) is returned. Valid value range is a number between 0 and 100.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

ParentName

Description

This property returns the name of the parent branch in a branched tree, as a string. If the tree does not have levels, is an unbranched tree, or if the tree is the root level, Null is returned.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

PerformanceMethod

Description

This property sets the method to use for select and join statements used to generate SQL with the tree. This property is used with nVision layouts and Query.

The values for this property are:

Value	Description
"L"	Suppress Join: Use Literal Values.
"S"	Sub-SELECT Tree Selector.
"J"	Join to Tree Selector.
"D"	User Application Defaults.

If this is a new tree, and you do not set this property, the default value "D" is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager, “Introduction to PeopleSoft Tree Manager” and *Enterprise PeopleTools 8.49 PeopleBook: PS/nVision*, “Understanding PS/nVision”.

 Copyright © 1988-2007, Oracle. All rights reserved.

PerformanceSelector

Description

This property selects the tree selectors for nVision for the tree. This property is used with nVision layouts and Query.

The values for this property are:

Value	Description
"D"	Dynamic Selectors
"S"	Static Selectors

If this is a new tree, and you do not set this property, the default value "S" is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PS/nVision, "Understanding PS/nVision"

PerformanceSelectorOption

Description

This property selects the tree selector options for nVision for the tree. This property is used with nVision layouts and Query.

The values for this property are:

Value	Description
"S"	Single values
"R"	Ranges of values (>=…<=).
"B"	Range of values (BETWEEN).

If this is a new tree, and you do not set this property, the default value "R" is automatically set.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PS/nVision, "Understanding PS/nVision"

Copyright © 1988-2007, Oracle. All rights reserved.

SetID

Description

This property returns the SetID for the tree, as a string. This property is used only when the IndirectionMethod property for the tree structure the tree is based on has been set to "S". Otherwise, this property returns Null.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Note. PeopleSoft does not recommend setting the SetID for a tree, as there is no way to save this value to the database.

See Also

Chapter 40, "Tree Classes," IndirectionMethod, page 1696

Status

Description

This property returns the effective status of the tree. The values are:

Value	Description
"A"	Active
"I"	Inactive

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

Structure

Description

This property returns a reference to the tree structure of the tree. Before you can use some of the methods or any of the properties of a tree structure, you must use the Open method to open the tree structure.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-only.

StructureName

Description

This property returns the name (*structure ID*) of the tree structure the tree is based on, as a string. This property can be set only with a new tree.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. Even though this is a writable property, you will receive an error if you try to change the tree structure for an existing tree.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

See Also

Chapter 40, "Tree Classes," SetID, page 1688

TreeImageName

Description

Use this property to specify the displayed image for a tree. This property takes a string value.

This property is read-write.

UserKeyValue

Description

Use this property to read the key field value for the Node User Key Field specified from the NodeRecord on the associated tree structure.

This property is used only when the IndirectionMethod property for the tree structure has been set to "U" or "B". Otherwise, this property returns Null.

This property is read-write.

Note. PeopleSoft does not recommend setting the UserKeyValue for a tree, as there is no way to save this value to the database.

See Also

Chapter 40, "Tree Classes," NodeRecord, page 1701 and Chapter 40, "Tree Classes," IndirectionMethod, page 1696.

UseUpdateReservation

Description

This property indicates if multi-user environment features are used. This property returns a Boolean value, true if multi-user environments features are used, false otherwise.

This property can be used with either an open or a closed tree.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Tree Structure Class

Tree structure objects are instantiated from the following:

- From a session object with the GetTreeStructure method.
- From a tree object with the Structure property.

See Chapter 40, "Tree Classes," GetTreeStructure, page 1618 and Chapter 40, "Tree Classes," Structure, page 1688.

Tree Structure Class Methods

In this section, we discuss the Tree Structure class methods. The methods are discussed in alphabetical order.

Close

Syntax

```
Close()
```

Description

The Close method closes the tree structure object executing the method, freeing the memory associated with that object, and discarding any changes made to the tree structure. The Close method can be used only on an open tree structure, not on a closed tree structure. This means you must have opened the tree structure with the Open method before you can save it. To save any changes, you must use the Save method before using Close.

Returns

A number; 0 if the method completed successfully.

Copy

Syntax

```
Copy(FromStructId, ToStructId)
```

Description

The Copy method copies the tree structure specified with *FromStructId* to the tree structure specified with *ToStructId*. Both parameters take string values. The tree structure specified by *ToStructId* must be a *new* tree structure. You receive an error if the tree structure already exists.

The Copy method can only be used with a closed tree structure, it cannot be used on an open tree structure. Before you use the Copy method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

To access the new tree structure, you must use the Open method.

Returns

A number; 0 if the method completed successfully.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
&MyTreeStruct = &MySession.GetTreeStructure();
&RetValue = &MyTreeStruct.Copy(PERSONAL_DATA, PERSONAL_DATA2);
```

Create

Syntax

Create(*StructId*, *Type*)

Description

The Create method creates a tree structure with the name *StructId*, and of the type *Type*. Both parameters take a string value.

Note. If the tree structure already exists, it is overwritten with the newly created tree structure.

The Create method can be used only with a closed tree structure, it cannot be used on an open tree structure. Before you use the Create method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

The new tree structure must be saved (using the Save method) for this structure to be saved to the database.

After you create a new tree structure, you can open it with the Open method.

The values for *Type* are:

- "D": create a detail tree structure.
- "S": create a summary tree structure.

Currently, only the value of "D" is supported for the *Type* parameter. You can specify an empty string for *Type*. If you need to create a summary structure follow the example below:

```
&MYSTRUCT=&SESSION.GETSTRUCTURE();
IF ALL( &MYSTRUCT) THEN
   &RETVALUE=&MYSTRUCT.CREATE('DEPARTMENT_SUM',' ');
     IF NONE(&RETVALUE) THEN
        &MYSTRUCT.TYPE = 'S';
     END-IF;
   &RETVALUE=&MYSTRUCT.SAVE();
   &RETVALUE=&MYSTRUCT.CLOSE();
END-IF;
```

Delete

Syntax

Delete(*StructId*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

The Delete method deletes the tree structure specified by *StructId* from the database. The Delete method can be used only with a tree structure identifier, it cannot be used on a fully instantiated, open tree structure. Before you use the Delete method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

The tree structure specified by *StructId* does not have to match the tree structure identifier executing the method.

Returns

A number: 0 if the method completed successfully.

Open

Syntax

Open(*StructId*, *Update*)

Description

The Open method opens the tree structure specified by *StructId*. *StructId* takes a string value. You cannot read or set any properties of a tree structure until after you open it.

When you open a tree structure, you can specify whether you want to open it for update, or in read-only mode, with the *Update* parameter. *Update* takes a Boolean value. Values for *Update* are:

- "True": open tree structure in update mode (read-write).
- "False": open tree structure in read-only mode.

If you try updating or writing to a tree structure opened in read-only mode, your changes are ignored.

Returns

A number: 0 if the method completes successfully.

Example

```
&MyTreeStruct = &MySession.GetTreeStructure();
&REtValue = &MyTreeStruct.Open("ACCESS_GROUP", True);
```

Rename

Syntax

Rename(*FromStructId*, *ToStructId*)

Description

The Rename method renames the tree structure specified with *FromStructId* to the new name specified by *ToStructId*. Both parameters take a string value. The tree structure specified by *ToStructId* must be a *new* tree structure. You receive an error if the tree structure already exists. The change is automatically reflected in the database.

 Copyright © 1988-2007, Oracle. All rights reserved.

The Rename method can only be used with a closed tree structure, it cannot be used on an open tree structure. Before you use the Rename method, you must explicitly close any open tree structure objects (with the Close method.) You receive an error if there are any open tree structures.

Returns

A number: 0 if the method completes successfully.

Save

Syntax

```
Save()
```

Description

The Save method writes any changes to the tree structure executing the method to the database.

The Save method can be used only on an open tree structure, not on a closed tree structure. This means you must have opened the tree structure with the Open method before you can save it.

The tree structure object remains open after executing Save. You must execute the Close method on the object before it is closed and the memory freed.

Note. If you're calling the Tree API from an Application Engine program, the data won't actually be committed to the database until the Application Engine program performs a Commit.

Returns

A number: 0 if the method completes successfully.

Tree Structure Class Properties

In this section, we discuss the Tree Structure class properties. The properties are discussed in alphabetical order.

Description

Description

This property returns the description for the tree structure, as a string. The description is the text that displays in list boxes or reports.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

Example

```
&MYTREESTRUCT.Description = "Department Security Chart";
```

Copyright © 1988-2007, Oracle. All rights reserved.

DetailComponent

Description

This property returns the name of the detail component for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

DetailField

Description

This property returns the name of the detail field for the tree structure, as a string. This property is valid only if you define a detail tree structure. If you create a node-oriented tree structure, this property is not required. This field must exist on the specified DetailRecord.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

Example

The following example sets the DetailPage, DetailRecord, and DetailField properties for a tree structure. Note that the names of the page, record, and field are all capitalized: this is required for all PeopleSoft page, record, and field names.

```
&MYTREESTRUCT.DetailPage = "BUS_UNIT_TREE_INV";
&MYTREESTRUCT.DetailRecord = "BUS_UNIT_TBL_IN";
&MYTREESTRUCT.DetailField = "BUSINESS_UNIT";
```

See Also

Chapter 40, “Tree Classes,” DetailRecord, page 1696

DetailMenu

Description

This property returns the name of the detail menu for the tree structure, as a string. If the page you use for detail values (DetailPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the DetailMenu and DetailMenuBar properties.

This property is valid only if you define a detail tree structure. If you create a node-oriented tree structure, this property isn’t required.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

DetailMenuBar

Description

This property returns the name of the detail menu bar for the tree structure, as a string. If the page you use for detail values (DetailPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the DetailMenu and DetailMenuBar properties.

This property is valid only if you're define a detail tree structure. If you create a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must use the Open method on the tree structure before you can use this property.

This property is read-write.

DetailMenuItem

Description

This property returns the name of the detail menu item for the tree structure as a string.

This property can be used only with an open tree, that is, you must open the tree using the Open method before you can use this property.

This property is read-write.

DetailMultiNavigate

Description

This property sets or returns a Boolean value indicating whether the detail mulit-navigation tree structure navigation option has been selected or not: True, the Detail Multi-Navigation check box has been selected, False otherwise.

This property is read-write.

See Also

Chapter 40, "Tree Classes," NodeMultiNavigate, page 1701 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager*, "Setting Multi-Navigation Paths".

DetailPage

Description

This property returns the name of the detail page for the tree structure as a string. If the page you use for detail values (DetailPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the DetailMenu and DetailMenuBar properties.

This property is valid only if you're defining a detail tree structure. If you're creating a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

Example

The following example sets the DetailPage, DetailRecord, and DetailField properties for a tree structure. Note that the names of the page, record, and field are all capitalized: this is required for all PeopleSoft page, record, and field names.

```
&MYTREESTRUCT.DetailPage = "BUS_UNIT_TREE_INV";
&MYTREESTRUCT.DetailRecord = "BUS_UNIT_TBL_IN";
&MYTREESTRUCT.DetailField = "BUSINESS_UNIT";
```

DetailRecord

Description

This property returns the name of the detail record for the tree structure as a string. This property is valid only if you define a detail tree structure. If you create a node-oriented tree structure, this property isn't required.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

The following example sets the DetailPage, DetailRecord, and DetailField properties for a tree structure. Note that the names of the page, record, and field are all capitalized: this is required for all PeopleSoft page, record, and field names.

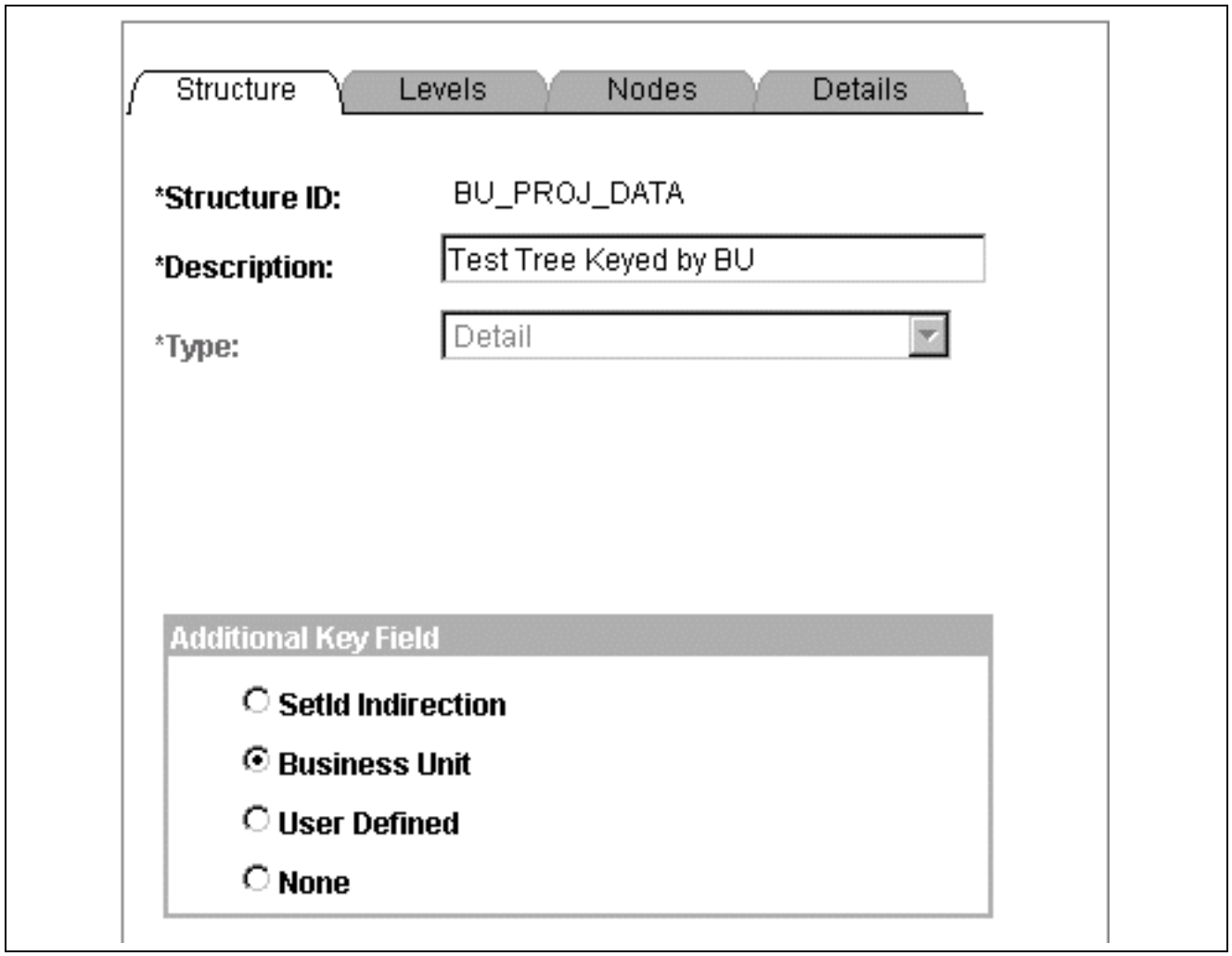
```
&MYTREESTRUCT.DetailPage = "BUS_UNIT_TREE_INV";
&MYTREESTRUCT.DetailRecord = "BUS_UNIT_TBL_IN";
&MYTREESTRUCT.DetailField = "BUSINESS_UNIT";
```

IndirectionMethod

Description

When you set up a tree structure, you can set the indirection at the bottom of the Structure tab, with the radio buttons in the Additional Key Field.

 Copyright © 1988-2007, Oracle. All rights reserved.



Structure tab for Tree Structure setup

In the Tree Classes API, you set these same values with the IndirectionMethod property. The values are:

- "S": SetID Indirection.
- "B": Business Unit.
- "U": User Defined Node Key.
- "N": None.

Both the *User Defined Node Key* and the *Business Unit* are the key field set with the NodeRecord property for this tree structure. If *Business Unit* is set, the key field set with the NodeRecord property is a business unit.

If you are creating a new tree structure, and you do not explicitly set this property, None ("N") is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also

Chapter 40, “Tree Classes,” IndirectionMethod, page 1696 and Chapter 40, “Tree Classes,” NodeRecord, page 1701.

Copyright © 1988-2007, Oracle. All rights reserved.

KeyName

Description

This property returns the name of the tree structure as a string.

This property can be used with a closed tree structure.

This property is read-only.

LevelComponent

Description

This property returns the name of the level component for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelMenu

Description

This property returns the name of the level menu for the tree structure as a string. If the page you use for level values (LevelPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the LevelMenu and LevelMenuBar properties.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelMenuBar

Description

This property returns the name of the level menu bar for the tree structure as a string. If the page you use for level values (LevelPage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the LevelMenu and LevelMenuBar properties.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelMenuItem

Description

This property returns the name of the level menu item for the tree structure as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelPage

Description

This property returns the name of the level page for the tree structure object as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_LEVEL is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

LevelRecord

Description

This property returns the name of the level record for the tree structure object as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_LEVEL_TBL is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Name

Description

This property returns the name of the tree structure as a string. This is also called the *structure ID*. Use this property to set the name (structure ID) of a tree structure. This property is required if you are creating a new tree structure.

Note. Do not use this property to change the name of an existing tree structure. The change will not be reflected in the database. You must use the Rename method to change the name of an existing tree structure.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

```
&Name = &MYTREESTRUCT.Name;
```

See Also

Chapter 40, “Tree Classes,” Rename, page 1692

Copyright © 1988-2007, Oracle. All rights reserved.

NodeComponent

Description

This property returns the name of the node component for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeField

Description

This returns the name of the field used for storing information about nodes for the tree structure as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_NODE is the default value, and is automatically set. This field must exist on the specified NodeRecord.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Example

```
&MYTREESTRUCT.NodeField = "DEPT_ID";
```

NodeMenu

Description

This property returns the name of the node menu for the tree structure as a string. If the page you use for node values (NodePage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the NodeMenu and NodeMenuBar properties.

If you create a new tree structure, and you do not explicitly set this property, TREE_MANAGER is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeMenuBar

Description

This property returns the name of the node menu bar for the tree structure as a string. If the page you use for node values (NodePage property) is part of a component, and you want to have access to the other pages in that group when you edit a node value, enter the name of the menu and menu bar for the component in the NodeMenu and NodeMenuBar properties.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

NodeMenuItem

Description

This property returns the name of the node menu item for the tree structure as a string.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeMultiNavigate

Description

This property sets or returns a Boolean value indicating whether the node mulit-navigation tree structure navigation option has been selected: True, the Node Multi-Navigation check box has been selected, False otherwise.

This property is read-write.

See Also

Chapter 40, "Tree Classes," DetailMultiNavigate, page 1695 and *Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Tree Manager*, "Setting Multi-Navigation Paths".

NodePage

Description

This returns the name of the page used for storing information about nodes for the tree structure as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_NODE is the default value, and is automatically set.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

NodeRecord

Description

This returns the name of the record used for storing information about nodes for the tree structure as a string. If you create a new tree structure, and you do not explicitly set this property, TREE_NODE_TBL is the default value, and is automatically set.

If the IndirectionMethod property of this tree structure has been set to "U" or "B", use this property to specify the record that the NodeUserKeyField property will use.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 40, “Tree Classes,” IndirectionMethod, page 1696

NodeUserKeyField

Description

Use this property to set the key field from the record specified with NodeRecord for use with the IndirectionMethod. This property is identical to the UserKeyValue property for a tree.

This property is used only when the IndirectionMethod property for the tree structure has been set to "U" or "B". Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also
Chapter 40, “Tree Classes,” IndirectionMethod, page 1696 and Chapter 40, “Tree Classes,” UserKeyValue, page 1689.

SummarySetId

Description

This property returns the SetID for the detail tree whose detail values the summary tree structure as a string summarizes. This property is valid only for Summary tree structures. Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

SummaryLevelNumber

Description

This property returns the level number for the detail tree whose detail values the summary tree structure summarizes. The level number indicates the level in the detail tree whose nodes the summary tree will use as detail values. The top level in the detail tree is 1, the next is level two, and so on. Valid values are numbers between 1 and 99.

This property is only valid for Summary tree structures. Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

SummaryTreeName

Description

This property returns the name of the detail tree whose detail values the summary tree structure summarizes, as a string. This property is valid only for Summary tree structures. Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

SummaryUserKeyValue

Description

This property returns the name of the tree that the node records the tree structure is summarized to, as a string. This property is valid only for Summary tree structures. This property is used only when the IndirectionMethod property for the tree structure has been set to "U" for *User Defined Keys* or "B" for *Business Unit.* Otherwise, this property returns Null.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

See Also

Chapter 40, “Tree Classes,” IndirectionMethod, page 1696

Type

Description

This property returns the type of the tree structure as a string, whether it is a detail or summary tree structure. The values are:

- "D": detail tree structure.
- "S": summary tree structure.

This property can be used only with an open tree structure, that is, you must open the tree structure with the Open method before you can use this property.

This property is read-write.

Traverse Tree Hierarchy Example

The following is an example to transverse a tree hierarchy and write the information to a file.

```
Local ApiObject &TreeObj, &RootNodeObj;
Local number &Res;
Local string &sSetId, &sSetCntrlValue, &sTreeName, &sBranchName;
Local File &MyFile;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
Function ProcessNodeChildren(&ParentNodeObj As ApiObject)

   Local ApiObject &ChildNodeObj, &ChildLeafObj;
   Local boolean &First;

   If &ParentNodeObj.HasChildren Then

      &ChildLeafObj = &ParentNodeObj.FirstChildLeaf;
      While All(&ChildLeafObj)
         &MyFile.WriteLine("Process Leaf Range From : " | &ChildLeafObj.Range⇒
From);
         &ChildLeafObj = &ChildLeafObj.NextSib;
      End-While;

      If &ParentNodeObj.HasChildNodes Then
         &First = True;
         &ChildNodeObj = &ParentNodeObj.FirstChildNode;
         While &First Or
               &ChildNodeObj.HasNextSib

            If &First Then
               &First = False;
            Else
               &ChildNodeObj = &ChildNodeObj.NextSib;
            End-If;
            &MyFile.WriteLine("Process Node : " | &ChildNodeObj.Name);
            ProcessNodeChildren(&ChildNodeObj);

         End-While;
      End-If;
   End-If;
End-Function;

&TreeObj = %Session.GetTree();

If None(&TreeObj) then
   Error("Cannot get a tree");
End-if;

&sSetId = "";
&sSetCntrlValue = "";
&sTreeName = "QE_JOBCODES";
&TreeEffDt = Date("1999-01-01");
&sBranchName = "";

&Res = &TreeObj.OPEN(&sSetId, &sSetCntrlValue, &sTreeName, &TreeEffDt, &sBranch⇒
Name, False);
If All(&Res) Then
   Error("Cannot open a tree");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
End-If;

&MyFile = GetFile("TreeDump.txt", "A");
&RootNodeObj = &TreeObj.FindRoot();

&MyFile.WriteLine("Process Root Node : " | &RootNodeObj.Name);

ProcessNodeChildren(&RootNodeObj);
&MyFile.Close();
```

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 41

Universal Queue Classes

This chapter provides an overview of the universal queue classes and discusses the following topics:

- MCFFactory class hierarchy
- Scope of universal queue classes
- Data types of universal queue classes
- How to import universal queue classes
- How to create universal queue class objects
- Universal queue class reference

Understanding Universal Queue Classes

The universal queue classes provide the means by which applications can inspect objects that are processed by the queue server, such as tasks, agents, and queues. The classes also enable tasks to reenter the queue with their original task ID, as well as keep task times based on its original entry into the system.

All of the classes, and most of the properties and methods that make up the universal queue classes have a GUI representation in the PeopleSoft Multi-Channel Framework. This document assumes that the reader is familiar with PeopleSoft Multi-Channel Framework.

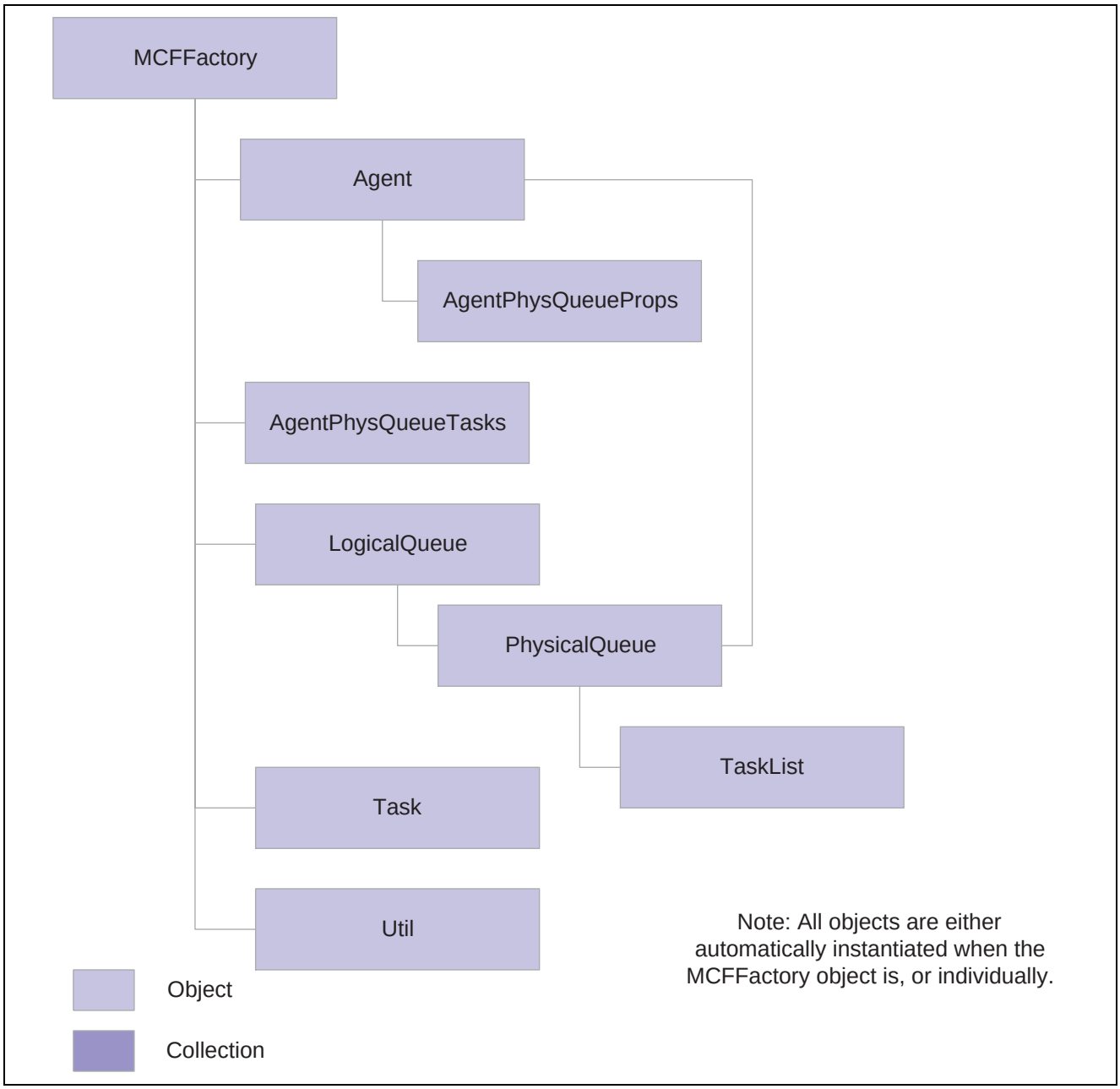
See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft MultiChannel Framework, "Understanding PeopleSoft MultiChannel Framework"

MCFFactory Class Hierarchy

The MCFFactory class is the root class from which the other classes are derived. Instantiate an object in this class, then traverse down the properties to inspect other individual objects, such as tasks, agents or queues. You can also instantiate individual MCFFactory objects directly and inspect their properties.

Copyright © 1988-2007, Oracle. All rights reserved.



Universal queue classes hierarchy

Scope of the Universal Queue Classes

The universal queue classes can be instantiated only from PeopleCode.

The universal queue classes can be called from a component, an internet script, or an Application Engine program.

Universal queue classes can be of Local, Global, or Component scope.

Copyright © 1988-2007, Oracle. All rights reserved.

Data Types of the Universal Queue Classes

Every universal queue class is its own data type, that is, Agent objects are declared as data type Agent, LogicalQueue objects are declared as type LogicalQueue, and so on.

The following are the data types of the universal queue classes:

- Agent
- AgentPhysQueueProps
- AgentPhysQueueTasks
- Broadcast
- LogicalQueue
- MCFFactory
- PhysicalQueue
- Task
- TaskList
- Util

How to Import Universal Queue Classes

The universal queue classes are *not* built-in classes, like rowset, field, record, and so on. They are application classes. You must import the universal queue application package before you can use the classes in your PeopleCode program.

An import statement names either all the classes in a package or one particular application class. For importing the universal queue classes, PeopleSoft recommends that you import all the classes in the application package.

The application package PT_MCF_UQAPI contains the following sub-packages:

- Agent
- AgentPhysQueueProps
- AgentPhysQueueTasks
- Broadcast
- LogicalQueue
- MCFFactory
- PhysicalQueue
- Task
- TaskList
- Util

The import statements you should use are as follows:

```
import PT_MCF_UQAPI:Agent:*;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
import PT_MCF_UQAPI:AgentPhysQueueProps:*;
import PT_MCF_UQAPI:AgentPhysQueueTasks:*;
import PT_MCF_UQAPI:Broadcast:*;
import PT_MCF_UQAPI:LogicalQueue:*;
import PT_MCF_UQAPI:MCFFactory:*;
import PT_MCF_UQAPI:PhysicalQueue:*;
import PT_MCF_UQAPI:Task:*;
import PT_MCF_UQAPI:TaskList:*;
import PT_MCF_UQAPI:Util:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are *not* made available.

See Also

Chapter 6, “Application Classes,” page 153

How to Create a Universal Queue Object

After you’ve imported the universal queue classes, you instantiate an object of one of those classes using the constructor for the class and the *Create* function.

The following example creates a new instance of the MCFFactory class, as the variable &MyMCFFactory, with local scope:

```
Local MCFFactory &myMCFFactory  = Create MCFFactory ();
```

See Also

Chapter 41, “Universal Queue Classes,” Universal Queue Classes Constructors, page 1710

Universal Queue Classes Constructors

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the universal queue Classes.

Agent

Syntax

Agent(*AgentID*)

Description

Use the Agent constructor to instantiate an Agent object.

All objects encapsulated by this object are created when the constructor is called.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. When an MCFFactory object is created, Agent objects are automatically created. In this case, the number of tasks in the list is limited by the *MAX_TASKLIST_ITEMS* parameter used when creating the MCFFactory object.

Parameters

Parameter	Description
AgentID	Specify a valid AgentID as a string.

Returns

An Agent object.

Example

```
Create Agent(&AgentID);
```

See Also

Chapter 41, "Universal Queue Classes," Agent Class, page 1717

AgentPhysQueueProps

Syntax

AgentPhysQueueProps(*AgentID, PhysicalQueueID*)

Description

Use the AgentPhysQueueProps constructor to instantiate an agent physical queue properties object.

Parameters

Parameter	Description
AgentId	Specify as valid AgentId as a string.
PyhscialQueueId	Specify a PhyscialQueueId as a string.

Returns

An agent physical queue properties object.

See Also

Chapter 41, "Universal Queue Classes," AgentPhysQueueProps Class, page 1722

AgentPhysQueueTasks

Syntax

AgentPhysQueueTasks(*AgentID, PhysicalQueueID*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the AgentPhysQueueTasks constructor to instantiate an agent physical queue task object.

Parameters

Parameter	Description
AgentID	Specify an agent ID as a string.
PhysicalQueueID	Specify a PhysicalQueueID as a string.

Returns

An agent physical queue task object.

See Also

Chapter 41, “Universal Queue Classes,” AgentPhysQueueTasks Class, page 1723

Broadcast

Syntax

```
Broadcast()
```

Description

Use the Broadcast constructor to instantiate a Broadcast object.

Parameters

None.

Returns

A Broadcast object.

See Also

Chapter 41, “Universal Queue Classes,” Broadcast Class, page 1725

LogicalQueue

Syntax

```
LogicalQueue(LogicalQueueID)
```

Description

Use the LogicalQueue constructor to instantiate a LogicalQueue object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
LogicalQueueID	Specify the ID of a logical queue as a string.

Returns

A LogicalQueue object.

Example

```
Local LogicalQueue &MyLogQueue = Create LogicalQueue(&LogicalQueueID);
```

See Also

Chapter 41, "Universal Queue Classes," LogicalQueue Class, page 1727

MCFFactory

Syntax

MCFFactory([*MAX_TASKLIST_ITEMS*])

Description

Use the MCFFactory constructor to instantiate an MCFFactory object.

Parameters

Parameter	Description
MAX_TASKLIST_ITEMS	Specify the maximum number of tasks loaded on a TaskList.

Returns

An MCFFactory object.

Example

```
Local MCFFactory &myMCFFactory = Create MCFFactory(MAX_TASKLIST_ITEMS);
```

See Also

Chapter 41, "Universal Queue Classes," MCFFactory Class, page 1728

PhysicalQueue

Syntax

PhysicalQueue(*PhysicalQueueID*)

Description

Use the PhysicalQueue constructor to instantiate a PhysicalQueue object. All objects encapsulated by this object are created when the constructor is called.

Copyright © 1988-2007, Oracle. All rights reserved.

Note. When an MCFFactory object is created, the PhysicalQueue objects are automatically created. In this case, the number of tasks in the list is limited by the *MAX_TASKLIST_ITEMS* parameter used when creating the MCFFactory object.

Parameters

Parameter	Description
PhysicalQueueID	Specify a physical queue ID as a string.

Returns

A PhysicalQueue object.

Example

```
Local PhysicalQueue &myPhysicalQueue = Create PhysicalQueue(&PhysicalQueueID);
```

See Also

Chapter 41, "Universal Queue Classes," PhysicalQueue Class, page 1728

Task

Syntax

Task(*TaskNumber*)

Description

Use the Task constructor to instantiate a task object.

Only persistent tasks (email and generic tasks) are available for introspection. Each persistent task goes through a lifecycle that is recorded by the following statuses:

State	Indicated Action	Status of Task Object
Enqueued by the universal queue server	Enqueued	ENQ
Universal queue assigns to physical queue	Assigned	ASGN
Agent accepts task	Accepted	ACPT
Agent completes task	Done	DONE
Task is not accepted within the overflow (action) time period	Overflowed	OVFL

 Copyright © 1988-2007, Oracle. All rights reserved.

State	Indicated Action	Status of Task Object
Agent does not complete task or task not assigned within the escalation (complete) time period	Escalated	ESCL
Inderminate Task (task not found in system)	Indeterminate	IND

Parameters

Parameter	Description
TaskNumber	Specify a task number.

Returns

A task object if the task exists, or an error message if the task number is not in the expected format. If the task number is properly formatted, but the task is not found, the task object is returned with a task status of IND.

Example

```
Local string &tasknumber, &status;

Local Task &myTask = Create Task(&tasknumber);

&status = &myTask.Status;
```

See Also

Chapter 41, “Universal Queue Classes,” Task Class, page 1733

TaskList

Syntax

TaskList(*Status*, *TaskType*, *PhysicalQueueID*, *AgentID*)

Description

Use the TaskList constructor to instantiate a task list.

If the queue is not provided, a list of tasks for all physical queues for that *TaskType* and *Status* is returned. To get all tasks on all queues do not specify any of the optional parameters.

Parameters

Parameter	Description
Status	Specify a task status, as an uppercase string. If you do not specify a valid status, an empty tasklist is returned. Valid values are:

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
ACPT	Accepted
ASGN	Assigned
ENQ	Enqueued
ESCL	Escalated
OVFL	Overflowed

Parameter	Description
TaskType	Specify a task type as a lowercase string. Valid values are:

Value	Description
chat	Chat tasks.
email	Email tasks.
generic	Generic type tasks.
voice	Voice type tasks.

Parameter	Description
PhysicalQueueID	Specify a physical queue ID as a string.
AgentID	Specify an agent ID as a string.

Returns

A task list object.

Example

```
Local string &sStatus;

/* get all enqueued task on the UQ server*/

&sStatus = "ENQ";
Local TaskList &myTaskList = Create TaskList (&sStatus, "", "", "");
```

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

Util

Syntax

Util()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the Util constructor to instantiate a util object.

Parameters

None.

Returns

A Util object.

Example

```
Create Util();

Local &myUtil = create Util();
```

See Also

Chapter 41, "Universal Queue Classes," Util Class, page 1740

Agent Class

Use the agent class to refresh or view the properties, such as the agent ID or buddy list associated with a particular agent.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft MultiChannel Framework, "Configuring PeopleSoft MCF Agents"

Agent Methods

In this section, we discuss the agent class methods. The methods are discussed in alphabetical order.

Delete

Syntax

Delete()

Description

Use the Delete method to delete an agent from the database. You cannot delete an agent that has still has open tasks.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A number. The values are:

Value	Description
0	Agent deleted successfully.
1	Invalid agent ID.
2	Agent has accepted task and cannot be deleted.
3	Other SQL error in executing deletion. Possibly there are errors in the child tables.

Example

```
import PT_MCF_UQAPI:*;


   &testAgent = create PT_MCF_UQAPI:Agent("QEMGR");

   /* Delete an UQ Agent definition */
   &retCode = &testAgent.Delete();
```

Refresh

Syntax

```
Refresh()
```

Description

Use the Refresh method to refresh the properties of the agent.

An agent on a queue is processing tasks as they are assigned to them. Use this method to see realtime statistics on how much load an agent has, or possibly before running any metrics.

If you're interested in how tasklists have changed, use the RefreshQTaskList method instead.

Parameters

None.

Returns

None.

Example

The following refreshes all the properties of the agent:

```
&MyAgent.Refresh();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 41, "Universal Queue Classes," RefreshQTaskList, page 1719

RefreshQTaskList

Syntax

RefreshQTaskList(*PhysQ*)

Description

Use the RefreshQTaskList method to refresh the task list properties for the specified physical queue.

You might use this method if you're interested in seeing the realtime statistics for a particular tasklist. If you are only interested in an agent's work, use the Refresh method.

Parameters

Parameter	Description
PhysQ	Specify a physical queue ID as a string.

Returns

None.

Example

The following refreshes all task lists on the first physical queue of the agent:

```
&MyAgent.RefreshQTaskList(&myAgent.PhysicalQueueID [1]);
```

See Also

Chapter 41, "Universal Queue Classes," Refresh, page 1718

Agent Properties

In this section, we discuss the agent class properties. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

AgentProps

Description

This property returns the agent properties associated with this agent as an array of AgentPhysQueueProps objects.

This property is read-only.

Example

The following example returns the workload for physical queue SALES of agent properties object:

```
Local number &i = 1;

While &i < &myAgent.TotalPhysicalQueues
   If &myAgent.AgentProps[&i].PhysicalQueueID = "SALES" Then
      &AgentWorkLoad = &myAgent.AgentProps[&i].WorkLoad
      break;
   End-If;
End-While;
```

See Also

Chapter 41, “Universal Queue Classes,” AgentPhysQueueProps Class, page 1722

AgentTasks

Description

This property returns the agent tasks associated with this agent as an array of AgentPhysQueueTasks objects.

This property is read-only.

Example

The following example returns the task number for the second task on physical queue SALES in the accepted task list:

```
Local Number &i = 1;

While &i < &myAgent.TotalPhysicalQueues
   If &myAgent.AgentProps[&i].PhysicalQueueID = "SALES" then
      &Tasknumber = &myAgent.AgentsTasks [&i].AcceptedTaskList.Task[2].TaskNumber
      break;
   End-If;
End-While;
```

See Also

Chapter 41, “Universal Queue Classes,” AgentPhysQueueTasks Class, page 1723

 Copyright © 1988-2007, Oracle. All rights reserved.

Buddy

Description

This property returns a list of buddies for this agent, as an array of string.

This property is read-only.

Example

The following example returns the total number of buddies on the list:

```
&BuddyTotal = &myAgent.Buddy.Len;
```

Language

Description

This property returns the list of languages associated with the agent as an array of string.

This property is read-only.

Example

The following code example returns the first language code in the list of languages associated with the agent:

```
&Language = &myAgent.Language[1];
```

Name

Description

This property returns the name of the agent as a string.

This property is read-only.

NickName

Description

This property returns the nick name of the agent as a string.

This property

PhysicalQueueID

Description

This property returns the physical queues to which this agent is assigned as an array of string.

This property is read-only.

Example

The following example returns the first physical queue of agent.

Copyright © 1988-2007, Oracle. All rights reserved.

```
&PhysicalQueueID = &myAgent.PhysicalQueueID[1];
```

See Also

Chapter 41, “Universal Queue Classes,” PhysicalQueue Class, page 1728

TotalPhysicalQueues

Description

This property returns the total number of physical queues to which this agent is assigned as a number.

This property is read-only.

Example

```
&TotalPhysicalQueues = &myAgent.TotalPhysicalQueues;
```

See Also

Chapter 41, “Universal Queue Classes,” PhysicalQueue Class, page 1728

AgentPhysQueueProps Class

Use the AgentPhysQueueProps class to view the properties of the physical queue assigned to an agent.

AgentPhysQueueProps Properties

In this section, we discuss the agent physical queue properties class properties. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue ID as a string.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

The following code example returns the physical queue ID for the physical queue of agent.

```
&PhyscalQueueID = &myAgentPhysQueueProps.PhysicalQueueID;
```

See Also

Chapter 41, "Universal Queue Classes," PhysicalQueue Class, page 1728

SkillLevel

Description

This property returns the skill level as a number.

This property is read-only.

Example

The following code example returns the skill level on physical queue.

```
&SkillLevel = &myAgentPhysQueueProps.SkillLevel;
```

WorkLoad

Description

This property returns the maximum work load that can be assigned to the agent, as an integer.

This property is read-only.

AgentPhysQueueTasks Class

Use the AgentPhysQueueTasks class to refresh or view the tasks associated with the physical queue assigned to the agent.

AgentPhysQueueTasks Method

In this section, we discuss the agent physical queue tasks class method Refresh.

Refresh

Syntax

Refresh(*TaskList*)

Description

Use the Refresh method to refresh the specified task list.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TaskList	Specify the task list you want refereshed. You must specify a task list for this parameter.

Returns

None.

Example

The following code example refresehs the accepted task list of the physical queue (&MyAgentPhysQueueTasks) for the agent.

```
&myAgentPhysQueueTasks.Refresh(&myAgentPhysQueueTasks.AcceptedTaskList);
```

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

AgentPhysQueueTasks Properties

In this section, we discuss the agent physical queue tasks class properties. The properties are discussed in alphabetical order.

AcceptedTaskList

Description

This property returns the accepted task list as a task list object.

This property is read-only.

Example

The following code example returns the task number of the first accepted task:

```
&TaskNumber = &myAgentPhysQueueTasks.AcceptedTaskList.Task[1].TaskNumber;
```

AssignedTaskList

Description

This property returns the assigned task list as a task list object.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue ID as a string.

This property is read-only.

Example

The following code example returns the physical queue ID for the physical queue of agent.

```
&PhyscalQueueID = &myAgentPhysQueueTasks.PhysicalQueueID;
```

See Also

Chapter 41, “Universal Queue Classes,” PhysicalQueue Class, page 1728

Broadcast Class

The Broadcast class has a single method, Broadcast. Use the Broadcast method to broadcast a notification message. You can specify whether to send the message to agents, to a queue, or even system wide.

Broadcast Class Method

The following is the description of the Broadcast class method Broadcast.

Broadcast

Syntax

Broadcast(*ClusterID*,*QueueID*, *ChannelID*, *AgentState*, *AgentPresence*, *Message*, *MessageSetNumber*, *MessageNumber*, *DefaultMessage*, *SecurityLevel*, *ImportanceLevel*, *SenderId*, *NameValueString*)

Description

Use the Broadcast function to broadcast a notification message. You can specify whether to send the message to agents, to a queue, or even system wide.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ClusterID	Specify the name of the cluster that you want to broadcast the message to, such as, RENCLSTR_001, as a string.
QueueID	Specify the name of the physical or logical queue that you want to broadcast the message to, such as, SALES, as a string.
ChannelID	Specify the name of the channel, or task, for the broadcast, such as Email, Chat, Voice or Generic, as a string.
AgentState	Specify the state of the agents you want to broadcast the message to, such as Available, as a string.
AgentPresence	Specify the presence of the agents you want to broadcast the message to, such as Active, as a string.
Message	Specify the text of the message you want to broadcast, as a string.
MessageSetNumber	Specify the message set number of a message from the message catalog if you want to broadcast a message from the message catalog. You must also specify values for the *MessageNumber* and *DefaultMessageText* parameters if you want to broadcast this type of message. Specify the message set number as a number.
MessageNumber	Specify the message number of a message from the message catalog if you want to broadcast a message from the message catalog. You must also specify values for the *MessageSetNumber* and *DefaultMessageText* parameters if you want to broadcast this type of message. Specify the message number as a number.
DefaultMessageText	Specify the text to be used if the specified message catalog message isn't found. Use the *MessageSetNumber* and *MessageNumber* parameters to specify the catalog message. Specify the default message text as a string.
SercurityLevel	Specify the security level for the broadcast message, as a string.
ImportanceLevel	Specify the importance level of the broadcast message, as a string.
SenderID	Specify the ID of the sender of the broadcast message, as a string.
NameValueString	Specify a string containing name-value pairs specific to your application.

Returns

None.

Example

The following example would broadcast a message to a specific logical queue:

```
import PT_MCF_UQAPI:Broadcast:*;
Local PT_MCF_UQAPI:Broadcast &BC;

&BC.Broadcast("", "SALES", "", "", "Best of Luck!", "", "", "Default Message",⇒
 "PRIV1", "URGENT", "Admin", "EffDate, 2005-10-25:12:00:45");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," MCFBroadcast

LogicalQueue Class

A logical queue is an application level queue that receives work requests (tasks) relating to an application area, such as chat requests regarding sales information, and routes them to agents capable of handling the work. Use the LogicalQueue class to view the properties for a logical queue.

LogicalQueue Properties

In this section, we discuss the logical queue class properties. The properties are discussed in alphabetical order.

LogicalQueueID

Description

This property returns the logical queue ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns all the physical queues associated with this logical queue as an array of PhysicalQueue objects.

This property is read-only.

Example

The following code example returns the physical queue ID for the fourth physical queue.

```
&PhyscalQueueID = &myLogicalQueue.PhysicalQueue[4].PhyscalQueueID;
```

Copyright © 1988-2007, Oracle. All rights reserved.

MCFFactory Class

The MCFFactory class is the base class of the universal queue classes. Instantiating an object in this class automatically instantiates all the associated universal queue classes. The MCFFactory contains all the logical queues operating on the MCF universal queue. The logical queues are comprised of physical queues. Agents log onto these physical queues. Tasks are queued to physical queues, and later, assigned and accepted by agents. If agents aren't available on a physical queue to accept the tasks on that queue, the tasks are either escalated or put into overflow.

MCFFactory Property

In this section, we discuss the MCFFactory class property LogicalQueue.

LogicalQueue

Description

This property returns all the logical queues with which this MCFFactory object is associated, as an array of LogicalQueue objects.

This property is read-only.

Example

The following code example returns the physical queue ID for the first physical queue on the logical queue, as well as the total of overflowed taskes on the first physical queue on the logical queue.

```
Local number &TotalTasks;

Local MCFFactory &myMCFFactory  = Create MCFFactory ();

&PhyscalQueueID = &myMCFFactory.LogicalQueue [1].PhysicalQueue [1].PhysicalQueueID;

&TotalTasks  = &myMCFFactory.LogicalQueue [1].PhysicalQueue [1].OverflowedTask⇒
List.Total;
```

PhysicalQueue Class

Logical queues can be partitioned into physical queues for scalability. Use the PhysicalQueue class to refresh and view the properties of a physical queue.

 Copyright © 1988-2007, Oracle. All rights reserved.

PhysicalQueue Methods

In this section, we discuss the physical queue class methods. The methods are discussed in alphabetical order.

Refresh

Syntax

Refresh()

Description

Use the Refresh method to refresh all the objects in the physical queue.

Parameters

None.

Returns

None.

Example

```
&myPhysicalQueue.Refresh();
```

RefreshTaskList

Syntax

RefreshTaskList(*TaskList*)

Description

Use the RefreshTaskList method to refresh the specified task list.

Parameters

Parameter	Description
TaskList	Specify the task list you want refereshed. You must specify a task list for this parameter.

Returns

None.

Example

The following code example refreshes the accepted task list for the physical queue.

```
&myPhysicalQueue.RefreshTaskList(&myPhysicalQueue.AcceptedTaskList);
```

Copyright © 1988-2007, Oracle. All rights reserved.

PhysicalQueue Properties

In this section, we discuss the physical queue class properties. The properties are discussed in alphabetical order.

AcceptedTaskList

Description

This property returns the accepted task list as a task list object.

This property is read-only.

Example

The following code example returns the task number of the first accepted task:

```
&TaskNumber = &myPhysicalQueue.AcceptedTaskList.Task[1].TaskNumber;
```

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

AssignedTaskList

Description

This property returns the assigned task list as a task list object.

This property is read-only.

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

Agent

Description

This property returns all the agents assigned to the physical queue as an array of agent objects.

This property is read-only.

Example

The following example returns the name of the third agent.

```
&AgentName = &MyPhysicalQueue.Agent[3].Name;
```

BrowserURL

Description

This property returns the URL of the browser as a string. The format is that of an absolute URL.

 Copyright © 1988-2007, Oracle. All rights reserved.

The URL for the REN server that serves this MCF cluster for external clients and for agent chat. The Browser URL may be different from the InternalURL, which should not have to go through any firewall, reverse proxy server or other outward-facing security barrier.

This property is read-only.

See Also

Chapter 41, “Universal Queue Classes,” InternalURL, page 1731

EnqueuedTaskList

Description

This property returns the enqueued task list as a task list object.

This property is read-only.

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

EscalatedTaskList

Description

This property returns the escalated task list as a task list object.

This property is read-only.

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

InternalURL

Description

This property returns the internal URL as a string. The format is that of an absolute URL.

This property is read-only.

See Also

Chapter 41, “Universal Queue Classes,” BrowserURL, page 1730

IsActive

Description

This property returns true if the physical queue is active, false otherwise.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
Local LogicalQueue &myLogicalQueue  = Create LogicalQueue (&LogicalQueueID);

For &I = 1 to &myLogicalQueue.PhysicalQueue.Len

&PhyscalQueueID = &myLogicalQueue.PhysicalQueue[&I].PhyscalQueueID;
&myPhysicalQueue = Create PhysicalQueue (&PhysicalQueueID);

   If &myPhysicalQueue.IsActive Then
      /* do work */
   Else
      /* return error */
   End-If;

End-For;
```

LogicalQueueID

Description

This property returns the logical queue ID associated with the physical queue as a string.

This property is read-only.

Example

```
&LogicalQueueID = &myPhysicalQueue.LogicalQueueID;
```

OverflowedTaskList

Description

This property returns the overflowed task list as a task list object.

This property is read-only.

See Also

Chapter 41, “Universal Queue Classes,” TaskList Class, page 1738

PhysicalQueueID

Description

This property returns the physical queue ID of this physical queue.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

RENURLID

Description

This property returns the ID of the REN server for the MCFFactory as a string.

This property is read-only.

TotalAgents

Description

This property returns the total number of agents belonging to this physical queue as a number. They may or may not be logged to the queue at that time.

This property is read-only.

Task Class

Use the task class to manipulate tasks or to view the properties of a specific task.

Task Methods

In this section, we discuss the task class methods. The methods are discussed in alphabetical order.

Close

Syntax

`Close`(*`Comment`*)

Description

Use the Close method to close the task. Only use this method with overflowed or escalated tasks.

Note. This method does *not* resubmit the task. Use the Enqueue method to resubmit the task, then use this method to close it.

Parameters

Parameter	Description
Comments	Specify any comments you want associated with this closed task. You can specify a null value (`""`) for this parameter.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 41, “Universal Queue Classes,” Enqueue, page 1734

Enqueue

Syntax

Enqueue(*LogicalQueueID*,*AgentID*, *Timeout*, *ResponseTime*, *Cost*, *Priority*, *MinSkill*)

Description

Use the Enqueue method to add a task to the logical queue that the task was first assigned to. Every task enters the universal queue by being assigned to a logical queue.

Considerations Specifying Time

If no value (zero) is provided for the time parameters (such as *Timeout* or *ResponseTime*) the Timeout or ResponseTime properties from the task are used, respectively. This may cause the time to never occur as it creates a moving deadline. To avoid such a problem the time can be calculated for each task prior to enqueue as shown below.

Note. This logic is not included in the task’s enqueue function, so as to allow you the choice for how you want your application to handle this situation.

```
&Utilobj = create Util();
&CurrentTask = create Task(&Task_Number);
&TimeDelta = &Utilobj.GetTimeDiff(&currenttime, &CurrentTask.EscalationTime) / 60;
&AgentAcceptTime = &Utilobj.GetTimeDiff(&CurrentTask.EnqueueTime, &Current⇒
Task.OverFlowTime) / 60;
    If AddToDateTime(&currenttime, 0, 0, 0, 0, &AgentAcceptTime, 0) >= &Current⇒
Task.EscalationTime Then
       If &TimeDelta > 0 Then
          &AgentAcceptTime = &TimeDelta;
       Else
/* The current time already exceeds the Escalationn time*/
       End-If;
    End-If;
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
LogicalQueueId	Specify the logical queue that you want to add this task to, as a string. You can specify a null value ("") for this parameter. If no value is provided , the logicalQueueID that is a property of task is used.
AgentID	Specify the agent ID for this task, as a string. You can specify a null value ("") for this parameter. If you specify a null value, the task is given to any available agent. You might expect to use a null for this parameter generally, as most tasks are to be reassinged for random agent allocation. However, there may be cases where a particular agent (such as one who previously worked on this task) might be required to take this task. In this case an agentID is provided
Timeout	Specify the time out period for this task as a number of minutes. You can specify a zero for this parameter. If no value is provided, the Timeout property of the task is used. The acceptance timeout is the period of time that an agent has to accept an assigned task (click on the flashing icon on the MultiChannel console). If the task is not accepted within this time, the task is re-enqueued.
ResponseTime	Specify the response time for this task as a number of minutes. You can specify a zero for this parameter. If no value is provided, the ResponseTime property of the task is used. The response time refers to the when a task should be completed by in order to satisfy customer contracts (such as, a platinum customer who requires a response in one day for a email. If by the end of 24 hours the task is not completed, it is escalated or the response time is exceeded.)
Cost	Specify the cost of this task as a number. You can specify a zero for this parameter. This is the cost of the task as set by the application when the task was enqueued. If no cost is set at that time, then the default cost is used.
Priority	Specify the priority of the task as a number. One is the lowest possible priority. Higher numbers indicate high priorities. If no priority is set, the default priority for the task is used.
MinSkill	Specify the minimum skill level required to do this task, as a number. You can specify a zero for this parameter.

Returns

None.

Example

```
&CurrentTask = create Task(&Task_Number);
&CurrentTask.Enqueue(&CurrentTask.PhysicalQueueID, "", &AgentAcceptTime, &Task⇒
CompleteTime, &CurrentTask.Cost, &CurrentTask.Priority, &CurrentTask.SkillLevel);
```

Refresh

Syntax

Refresh()

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the Refresh method to refresh the task properties. If you only want to refresh the status, use the RefreshStatus method.

Parameters

None.

Returns

None.

Example

The following refreshes all the properties of the task:

```
&MyTask.Refresh();
```

See Also

Chapter 41, “Universal Queue Classes,” RefreshStatus, page 1736

RefreshStatus

Syntax

```
RefreshStatus()
```

Description

Use the RefreshStatus method to refresh only the status of the task. If you want to refresh all the task properties, use the Refresh method.

Parameters

None.

Returns

None.

Example

The following refreshes the status of the task:

```
&MyTask.RefreshStatus();
```

See Also

Chapter 41, “Universal Queue Classes,” Refresh, page 1735

 Copyright © 1988-2007, Oracle. All rights reserved.

Task Properties

In this section, we discuss the task class properties. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

ApplicationData

Description

This property returns a string in which data relevent to the task is passed. Typically, for a task, it contains the URL to show when an agent access this task, the subject, and so on. This data is not really useful to the end-user, but only for administrators and for test purposes.

This property is read-only.

Comments

Description

Use this property to either set or return the comments associated with this task, as a string.

This property is read-write.

EnqueueTime

Description

This property returns the datetime when the task was submitted to the universal queue as a datetime value.

This property is read-only.

EscalationTime

Description

This property returns the datetime when the task will be removed from the queue and placed in the escalated pool.

This property is read-only.

Language

Description

This property returns the language associated with the task as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

OriginalTime

Description

This property returns the datetime when the task was first submitted to the universal queue as a datetime value. This value is maintained for a task through its various stages and cannot change.

This property is read-only.

OverFlowTime

Description

This property returns the datetime when the task has reached the stipulated time during which the universal queue server could not find an agent to accept the task.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue associated with the task as a string.

This property is read-only.

TiedAgentID

Description

This property returns the agent ID of the agent this task is enqueued for as a string.

This property is read-only.

TaskList Class

Use the task list class to refresh or view a task list.

TaskList Method

This section discusses the task list method Refresh.

 Copyright © 1988-2007, Oracle. All rights reserved.

Refresh

Syntax

```
Refresh()
```

Description

Use the Refresh method to refresh the task list properties.

Parameters

None.

Returns

None.

Example

The following refreshes all the properties of the task list:

```
&MyTaskList.Refresh();
```

TaskList Properties

This sections discusses all the properties for the task list class. The properties are discussed in alphabetical order.

AgentID

Description

This property returns the agent ID as a string.

This property is read-only.

PhysicalQueueID

Description

This property returns the physical queue associated with the task list as a string.

This property is read-only.

Task

Description

This property returns all of the tasks in the task list as an array of task objects.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

```
For &I = 1 to &MyTaskList.Total

   &MyTask = &MyTaskList[&I];

   /* do work on every task in list */

End-For;
```

TaskType

Description

This property returns the task type as a string. A tasklist only contains tasks of a single type. The following are the valid task types for a tasklisk:

Value	Description
ACPT	Accepted
ASGN	Assigned
ENQ	Enqueued
ESCL	Escalated
OVFL	Overflowed

This property is read-only.

Total

Description

This property returns the total number of tasks in the list as a number.

This property is read-only.

Util Class

The Util class is used to perform certain useful tasks, such as calculating times (so a task will timeout).

Util Methods

This section discusses the Util class methods. The methods are discussed in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

GetLogicalQueue

Syntax

GetLogicalQueue(*PhysicalQueueID*)

Description

Use the GetLogicalQueue to get the logical queue ID for the specified physical queue.

Parameters

Parameter	Description
physicalQueueID	Specify the physical queue ID that you want to find the logical queue for, as a string.

Returns

A string containing the logical queue ID.

Example

The following code example returns the logical queue ID for a given physical queue ID .

```
&LogicalQueueID = &testUtil.GetLogicalQueue(&PhysicalQueueID);
```

GetTimeDiff

Syntax

GetTimeDiff(*DateTime1, DateTime2*)

Description

Use the GetTimeDiff method to determine the difference between two datetime values.

Parameters

Parameter	Description
DateTime1	Specify the starting datetime as a datetime value.
DateTime2	Specify the ending datetime as a datetime value.

Returns

A number specifying the difference between the two datetime values (in seconds.)

Example

In the following example, &nPeriod is 15 minutes.

```
&DTM1 = DateTimeValue("10/09/97 10:35:36");
&DTM2 = DateTimeValue("10/09/97 10:50:36");

Local number &nPeriod = &myUtil.GetTimeDiff(&DTM1, &DTM2) / 60;
```

Copyright © 1988-2007, Oracle. All rights reserved.

MCFFactory Example

The following is an example of creating an MCFFactory, then traversing the branches to reach elements from different points. It is the loops that are important—any code within the loops is just an example of retrieval.

```
&testMCFFactory = create MCFFactory(100); /* set a maximum number of tasks in a⇒
 task  list to 100 */

Local integer &i, &j, &k;
&i = 1;
&j = 1;
&k = 1;
&l = 1;
&m = 1;
&n = 1;

While &j < &testMCFFactory.LogicalQueue.Len + 1
   /*looping through the logical queues*/
   While &i < &testMCFFactory.LogicalQueue [&j].PhysicalQueue.Len + 1
      /*looping through the physical queues*/

      /*  e.g. Tasklists by physical queues*/
      &PhysQID = &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Physical⇒
QueueID;
      &nOvflTaskTotal = &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Over⇒
FlowedTaskList.Total;

      While &k < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].OverFlowed⇒
TaskList.Total + 1
         /*looping through the Tasklist on physical queues*/

         /*  e.g. Tasks by physical queues*/
         &tasknum = &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Over⇒
FlowedTaskList.Task [&k].TaskNumber;

         &k = &k + 1;
      End-While; /* &k loop */
      /*Task List by Agents on a Physical Queue */
      While &l < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].TotalAgents⇒
 + 1
         While &m < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Agent ⇒
[&l].TotalPhysicalQueues + 1
            /*  e.g. Workload of agents on physical queues.  NOTE: &m  and &i are⇒
 not the same value for the same physical queue.*/
            &AgentMaxWorkload = &testMCFFactory.LogicalQueue [&j].PhysicalQueue ⇒
[&i].Agent [&l].AgentProps [&m].Workload;
            While &n < &testMCFFactory.LogicalQueue [&j].PhysicalQueue [&i].Agent ⇒
[&l].AgentsTasks [&m].AcceptedTaskList.Total + 1
               /* e.g. tasks on Agent's accepted list  NOTE: &m  and &i are not⇒
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
 the same value for the same physical queue.*/
                &tasknum = &testMCFFactory.LogicalQueue [&j].PhysicalQueue ⇒
[&i].Agent [&l].AgentsTasks [&m].AcceptedTaskList.Task [&n];
                &n = &n + 1;
             End-While; /* &n Loop - tasks on Phys Q of Agents  */
             &n = 1;
             &m = &m + 1;
          End-While; /* &m loop - Phys Q of Agents  - convoluted*/
          &m = 1;
          &l = &l + 1;
       End-While; /* &l loop - Agents on Phys Q */
       &i = &i + 1;
       &k = 1;
    End-While; /* &i loop- Physical Queues*/
    &j = &j + 1;
    &i = 1;
End-While; /* &j loop - Logical Queues*/
```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 42

XmlDoc Class

This chapter provides an overview of XmlDoc and XmlNode classes and discusses the following topics:

- When to use an XmlDoc object
- XmlDoc object creation
- XmlNode class considerations
- Error handling
- SoapDoc objects consideration
- Scope of XmlDoc and XmlNode objects
- Data type of an XmlDoc or XmlNode object
- XmlDoc class built-in functions
- XmlDoc methods
- XmlDoc property
- XmlNode methods
- XmlNode properties

Understanding XmlDoc Class

The Extended Markup Language (XML) describes a class of data objects called XML documents. It also partially describes the behavior of computer programs which process them. The XmlDoc class is used to create and manipulate XML data.

The Extended Markup Language (XML) is a method for putting structured data in a text file. Like HTML, XML uses tags, that is, text delimited by brackets (< and >). However, HTML specifies what each tag is, and how it's supposed to be displayed in a browser. XML uses tags only to delimit data. The interpretation of that data is entirely up to the application.

For example, <UL> in HTML specifies an unordered (bulleted) list. However, with XML, it could specify an underlined link.

Each XML document has both a physical and a logical structure:

- Physically, the XML document is composed of units called *entities*. A document begins in a "root" or document entity.
- Logically, the XML document is composed of declarations, elements, comments, character references, and processing instructions

Copyright © 1988-2007, Oracle. All rights reserved.

The CreateXmlDoc function creates an XmlDoc object. An XmlDoc is composed of XmlNode objects. Each XmlNode represents an XML document *entity*. You can use PeopleCode to create the following types of entities:

- Attribute (specified both by a name and a NamespaceURI)
- CDATA Section
- Comment
- Element
- Entity References
- Process Instruction
- Text

See Also

http://www.w3.org

When to Use an XmlDoc Object

You can use the XmlDoc class to access inbound messages. You can also use the XmlDoc class to create XML messages to be sent out, either synchronously or asynchronously.

All messages must be associated with a message definition. An XmlDoc message must be associated with an *unstructured* or nonrowset-based message, that is, a message definition that's created without any record definitions.

Use the XmlDoc object if any of the following is true:

- Your message structure doesn't fit the rowset model.
- Your message data doesn't come from database records.
- Your third-party source or target node requires non-XML messages.
- Your message uses the SOAP protocol.

Do *not* use the XmlDoc object if your message contains cookies.

XmlDoc Object Creation

Use the CreateXmlDoc function to create an XmlDoc object. You can create either an empty object, and populate it with data, or you can specify an XML string that is then transformed into an XmlDoc object that you can then manipulate with PeopleCode.

If you're creating an empty XmlDoc object, and you also want to specify a particular document type declaration (DTD) to be used to validate your XML, immediately after you use the CreateXmlDoc function, you should use the CreateDocumentType method to specify the DTD, followed by the CreateDocumentElement method to associate the DTD with the root entity.

The following PeopleCode program:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local XmlDoc &inXMLDoc;
Local XmlNode &docTypeNode;
Local XmlNode &rootNode;

&inXMLDoc = CreateXmlDoc("");
&docTypeNode = &inXMLDoc.CreateDocumentType("Personal", "", "Personal.dtd");
&rootNode = &inXMLDoc.CreateDocumentElement("root", "", &docTypeNode);
```

produces the following XML:

```
<?xml version="1.0"?>
<!DOCTYPE Personal SYSTEM "Personal.dtd">
</root>
```

After you've created your XmlDoc object, use the GenXmlString method to create an XML string. You can then use an Internet Script (iScript) Response object method to send the string as an XML response.

See Also

Chapter 18, "Internet Script Classes (iScript)," page 585

Considerations Using a Unique Namespace

In the root tag, the attribute **xmlns** stands for the XML namespace. This allows you to define namespaces for tag names so that collisions can be avoided and validation logic can be run.

If you do not give a prefix for an XML namespace, but instead define it with the tag (xmlns) followed by a colon (:) and then a unique namespace, for example,

```
xmlns:psft
```

For example, this sort of functionality allows you to have two nodes named "Transaction", but one can be referenced by a "psft" namespace and another not, allowing for two nodes with the same name to exist, but each containing different data.

```
<?xml version="1.0"?>
<root xmlns:psft="http://www.peoplesoft.com">
    <psft:Transaction>Value</psft:Transaction>
    <Transaction>Another</Transaction>
</root>
```

Incorrect results can be returned when you have a namespace as in the above example, which belongs to the namespace defined in http://www.people.com. When the system tries to find the path of "root/Transaction", it may return multiple nodes when in fact the end user might only want to return one.

To avoid this, do one of the following:

- Do not use FindNode. Use GetElementByTagName instead. This does not use XPath to resolve entries in the DOM. Instead, it works at a node by node basis, for example, by getting the root node, then getting the transactions node. This code may be a bit more complex to write, but you loose the richness of XPath.
- Give every **xmlns** attribute a prefix. The system will use XPath correctly and find the node.

Copyright © 1988-2007, Oracle. All rights reserved.

Considerations Using Rowsets

Unstructured XML should be transformed to structured if you want to use the full rowset abilities. PeopleSoft recommends transforming the data into a rowset-based message.

If you do not want to transform the data, you need to break it up using the Transaction tag around the equivalent of each level zero rowset, as shown in the example.

```
<?xml version="1.0" ?>
 - <SAMPLE_MSG>
    - <Transaction>
       - <QE_SALES_ORDER class="R">
            <QE_ACCT_ID>26</QE_ACCT_ID>
            <QE_ACCOUNT_NAME>APG-65</QE_ACCOUNT_NAME>
            <QE_ADDRESS>F18 HORNET WAY</QE_ADDRESS>
            <QE_PHONE>(206)544-1264</QE_PHONE>
            <QE_FROMROWSET />
            <QE_TOROWSET />
            <QE_SEND_SOA_BTN />
            <QE_SEND_SOS_BTN />
            <QE_TRAN_SOA_BTN />
            <QE_SEND_SQ_BTN />
            <QE_TRAN_SOS_BTN />
            <QE_TRAN_APCODE_BTN />
            <QE_TRAN_SPCODE_BTN />
            <QE_PUBXMLDOC_BTN />
            <QE_CLEAR_BTN />
         <DESCRLONG />
       </QE_SALES_ORDER>
    </Transaction>
    - <Transaction>
       - <QE_SALES_ORDER class="R">
            <QE_ACCT_ID>27</QE_ACCT_ID>
            <QE_ACCOUNT_NAME>JASON ACCOUNT</QE_ACCOUNT_NAME>
            <QE_ADDRESS>THE ADDRESS</QE_ADDRESS>
            <QE_PHONE>(PHONE NUMBER</QE_PHONE>
            <QE_FROMROWSET />
            <QE_TOROWSET />
            <QE_SEND_SOA_BTN />
            <QE_SEND_SOS_BTN />
            <QE_TRAN_SOA_BTN />
            <QE_SEND_SQ_BTN />
            <QE_TRAN_SOS_BTN />
            <QE_TRAN_APCODE_BTN />
            <QE_TRAN_SPCODE_BTN />
            <QE_PUBXMLDOC_BTN />
            <QE_CLEAR_BTN />
         <DESCRLONG />
       </QE_SALES_ORDER>
    </Transaction>
 </SAMPLE_MSG>
```

 Copyright © 1988-2007, Oracle. All rights reserved.

XmlNode Class Considerations

In general, the XmlNode Class methods and properties can be broken up into the following categories:

Category	Description
Add	The Addxxx methods add an entity of the specified type to the end of the list of child nodes, and returns a reference to the newly created node.
Insert	The Insertxxx methods insert an entity of the specified type at the specific location and returns a reference to the newly created node.
Get	The Getxxx methods return a reference to the specified entity. The Getxxx methods may return a single reference or an array of references.
Other	The other methods enable you to find a particular entity in an XmlDoc (FindNode), copy data from one node into another, and remove nodes.

Accessing and Traversing an XmlNode Object

If you're creating an XmlDoc object, use the CreateDocumentElement, as well as the different Add and Insert methods to create XmlNode objects. These methods return a reference to the newly created node if successful.

Use the XmlNode properties for traversing the data structure (ParentNode, PreviousSibling, NextSibling, and so on.)

You can also use the ChildNodeCount property for looping through all the child nodes of a node.

Use the NodeType property to get the type of the node (element, processing instruction, comment, and so on).

Error Handling

Use the IsNull property to verify if the XmlNode returned by a method is a valid node. In the following example, a particular node is checked to see if it has another node after it. If it doesn't, a node is added.

```
Local XmlNode &usernode;
Local string &userName;

&usernode = &inXMLDoc.DocumentElement.FindNode("/Request/Company/Location/User");

If Not (&MyNode.IsNull) Then
   &userName = &usernode.NodeValue;
Else
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
   /* Do error processing */
End-If;
```

SOAPDoc Object Considerations

A SOAP Message is an ordinary XML document that consists of the following parts:

- a mandatory SOAP envelope
- an optional SOAP header
- a mandatory SOAP body

Most of the methods and properties that you use with an XmlDoc or XmlNode can also be used with a SOAPDocument. However, there are a few XmlNode methods that cannot be used with a SOAPDocument. The following methods *cannot* be used with a SOAPDocument.

- AddCDataSection
- AddProcessInstruction
- CopyToPSFTMessage
- CreateDocumentType
- InsertCDataSection
- InsertProcessInstruction
- LoadIBContent

See Also

Chapter 37, “SOAPDoc Class,” page 1539

Scope of XmlDoc and XmlNode Objects

XmlDoc and XmlNode objects can be instantiated only from PeopleCode.

This object can be used anywhere you have PeopleCode, that is, in Application Engine PeopleCode, record field PeopleCode, and so on.

Data Type of an XmlDoc or XmlNode Object

XmlDoc objects are declared type XmlDoc.

XmlNode objects are declared type XmlNode.

For example:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Component XmlDoc &MyDoc;
Local XmlNode &MyNode;
```

Note. XmlNode objects can be declared only as type Local.

XmlDoc Class Built-in Functions

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateSOAPDoc

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateXmlDoc

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," GetNRXmlDoc

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," ReValidateNRXmlDoc

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Transform

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransformEx

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," TransformExCache

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," UpdateXmlDoc

XmlDoc Methods

In this section, we discuss the XmlDoc methods. The methods are discussed in alphabetical order.

CopyRowset

Syntax

CopyRowset(*&InRowset*[, *MessageName*][, *MessageVersion*])

Description

Use the CopyRowset method to copy rowset data to an XmlDoc object.

Warning! If the XmlDoc is *not* empty, the existing structure and data are replaced with the data and structure from *InRowset* . If *MessageName* is not specified, this function makes the best possible XML structure given the passed rowset. Not passing the message name should only occur when using a nonrowset-based message or when using a standalone rowset. For best performance, PeopleSoft recommends to always specify the message name.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&InRowset	Specify the variable of an already instantiated and populated rowset to copy data from.
MessageName	Specify the name of the message that the rowset belongs to, as a string. If the message name is not specified, the best possible XML structure is created.
MessageVersion	Specify the message version, as a string. If you do not specify a message version, the default message version is used.

Returns

A Boolean value: True, the rowset data copied successfully, False otherwise.

Example

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.CopyRowset(&rs, "<insert message name>", "<insert message⇒
 version");
```

See Also

Chapter 42, “XmlDoc Class,” CopyToRowset, page 1754

Chapter 33, “Rowset Class,” page 1421

CopyToPSFTMessage

Syntax

CopyToPSFTMessage(*targetDoc*, *srcPath*, *targetPath*)

Description

Use the CopyToPSFTMessage method to copy an XmlDoc to a PeopleSoft message. The *srcPath* and *targetPath* parameters enable you to map the data of the XmlDoc (using the XmlDoc structure) to the message (following the message structure.)

The XmlDoc object generally has the following structure:

```
<?xml version="1.0"?>
<MESSAGE_NAME>
  <FieldTypes>
    ...
  </FieldTypes>
  <MsgData>
    <Transaction>
      ...
    </Transaction>
  </MsgData>
```

 Copyright © 1988-2007, Oracle. All rights reserved.

</MESSAGE_NAME>

Parameters

Parameter	Description
targetDoc	Specify the XmlDoc that contains the message structure.
srcPath	Specify an array of string that contains the mapping path information.
targetPath	Specify an array of string that contains the mapping path information.

Returns

A number. The values are:

Value	Description
0	Ok
1	Missing PSCAMA structure
2	Missing FieldType structure
3	General Error

Example

```
Local XmlDoc &srcDoc, &targetDoc;
Local array of string &srcPath, &targetPath;

&srcPath = CreateArrayRept("", 0);
&targetPath = CreateArrayRept("", 0);


&srcDoc = CreateXmlDoc("");
&targetDoc = CreateXmlDoc("");
&ret = &srcDoc.ParseXmlFromURL("c:/temp/source.xml");
&ret = &targetDoc.ParseXmlFromURL("c:/temp/target.xml");

&srcPath.Push("QE_EMPLOYEE");
&srcPath.Push("QE_EMPLOYEE/QE_ACCOUNT_TBL");
&srcPath.Push("QE_EMPLOYEE/QE_JOBCODE");
&srcPath.Push("QE_EMPLOYEE/DEPTID");
&srcPath.Push("EMAIL_MSG_RCD");
&srcPath.Push("EMAIL_MSG_RCD/EMAIL_FILE_RCD");

&targetPath.Push("QE_EMPLOYEE_TG");
&targetPath.Push("QE_EMPLOYEE_TG/QE_ACCOUNT_TBL_TG");
&targetPath.Push("QE_EMPLOYEE_TG/QE_JOBCODE_TG");
&targetPath.Push("QE_EMPLOYEE_TG/DEPTID_TG");
&targetPath.Push("EMAIL_MSG_RCD_TG");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&targetPath.Push("EMAIL_MSG_RCD/EMAIL_FILE_RCD_TG");

&ret = &srcDoc.CopyToPSFTMessage(&targetDoc, &srcPath, &targetPath);
```

See Also

Chapter 42, “XmlDoc Class,” CopyRowset, page 1751, Chapter 42, “XmlDoc Class,” CopyToRowset, page 1754 and Chapter 8, “Array Class,” page 215.

Chapter 22, “Message Classes,” page 729

CopyToRowset

Syntax

CopyToRowset(*&Rowset*, *Message_Name*, [*Message_Version*])

Description

Use the CopyToRowset method to copy data from the XmlDoc to an already instantiated rowset.

The rowset must be based on a message object that has the same *structure* as the XmlDoc. That is, if you have a record at level two in your message and you want that data, you must have the same level zero and level one records in your message as in your XmlDoc.

For example, suppose your XmlDoc had the following structure:

```
<?xml version="1.0"?>
<QE_SALES_ORDER>
    <QE_ACCT_ID/>
    <QE_ACCOUNT_NAME/>
    <QE_ADDRESS/>
    <QE_PHONE/>
    <QE_FROMROWSET/>
    <QE_TOROWSET/>
    <QE_SEND_SOA_BTN/>
    <QE_SEND_SOS_BTN/>
    <QE_TRAN_SOA_BTN/>
    <QE_SEND_SQ_BTN/>
    <QE_TRAN_SOS_BTN/>
    <QE_TRAN_APCODE_BTN/>
    <QE_TRAN_SPCODE_BTN/>
    <QE_PUBXMLDOC_BTN/>
    <QE_CLEAR_BTN/>
    <DESCRLONG/>
</QE_SALES_ORDER>
```

If you wanted to include the information in the QE_SALES_ORDER record, you would have to have *at least* the following record structure in your message:

```
QE_SALES_ORDER
```

Any records or fields that are in the XmlDoc that aren’t in the message (and vice-versa) are ignored.

Rowsets should be created using the following pseudo code:

 Copyright © 1988-2007, Oracle. All rights reserved.

```
Local Message &msg;
Local Rowset &rs;
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&msg = CreateMessage(OPERATION.<insert message name>);
&rs = &msg.GetRowset();
&inXMLDoc = CreateXmlDoc("<insert XML structure here>");
&ret = &inXMLDoc.CopyToRowset(&rs, "<insert message name>", "<insert message⇒
 version>");
```

XmlDoc objects have to follow the Peoplesoft message format:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>xxx</Field1>
                <Field2>yyy</Field2>
                ...
                <Fieldn>nnn</Fieldn>
                <Record2>
                ...
                </Record2>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

Warning! If *MessageName* is not specified, this function makes the best possible flat structure. Not passing the message name should only occur when using a nonrowset-based message or when using a standalone rowset. For best performance, PeopleSoft recommends to always specify the message name.

Parameters

Parameter	Description
&InRowset	Specify the variable of an already instantiated rowset to copy data to.
Message_Name	Specify the message name of the message. If the message name is not specified, the best possible flat structure is created.
Message_Version	Specify the message version, as a string. If the message version is not specified, the default message version is used.

Returns

This function always returns a True value, regardless of the success of the operation.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 42, "XmlDoc Class," CopyRowset, page 1751

Chapter 33, "Rowset Class," page 1421

Chapter 22, "Message Classes," page 729

CreateDocumentElement

Syntax

```
CreateDocumentElement(TagName [, NamespaceURI ] [, &DocType])
```

where *NamespaceURI* can have *one* of the following forms:

```
URL.URLname
```

OR a string URL, such as

```
http://www.peoplesoft.com/
```

Description

Use the CreateDocumentElement method to create the root element of the document.

For *&DocType* you must specify an already instantiated document type node using CreateDocumentType.

Parameters

Parameter	Description
TagName	Specify the name of the tag to be used for the document element.
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc, as a string.
&DocType	Specify the document type node. This must already be instantiated using CreateDocumentType.

Returns

A reference to an XmlNode object representing the root element.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &docTypeNode;
Local XmlNode &rootNode;

&inXMLDoc = CreateXmlDoc("");
&docTypeNode = &inXMLDoc.CreateDocumentType("Personal", "", "Personal.dtd");
&rootNode = &inXMLDoc.CreateDocumentElement("root", "", &docTypeNode);
```

produces the following XML:

```
<?xml version="1.0"?>
<!DOCTYPE Personal SYSTEM "Personal.dtd">
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
<root/>
```

See Also

Chapter 42, "XmlDoc Class," CreateDocumentType, page 1757

CreateDocumentType

Syntax

CreateDocumentType(*Name*, *PublicID*, *SystemID*)

Description

Use the CreateDocumentType method to create the document type declaration (a document type node) in the XmlDoc.

You can specify only *one* document type node for an XmlDoc.

You need to create a document type node only if you have a document type definition (DTD). You don't need to specify a document type declaration in your XML. Generally, the DTD file is used for validating the XML.

If you do specify this type of node when creating an XmlDoc in PeopleCode, you *must* specify it immediately following the creation statement. Then you must specify the document type when you create the root node. This means your code should start with the CreateXmlDoc function, creating the XmlDoc object, then use the CreateDocumentType method, followed by the CreateDocumentElement method, creating the root node.

Considerations Using Public and System IDs

You can specify both a public and a system ID. However, if you want to use *only* a system ID, you must specify a Null string ("") for the public ID. To use only a public ID, you must specify a Null string ("") for the system ID.

To use the System.dtd as a system ID, you must use the following code:

```
&DocType = &MyDoc.CreateDocumentType("Personal", "", "Personal.dtd");
```

This produces the XML:

```
<!DOCTYPE Personal PUBLIC "Personal.dtd">
```

Parameters

Parameter	Description
Name	Specify the name of the document type declaration as a string.
PublicID	Specify the public ID of the document type declaration as a string.
SystemID	Specify the system ID of the document type declaration as a string.

Returns

A reference to an XmlNode object representing the document type element.

Example

The following example creates a document type declaration using the DTD named "Personal".

Copyright © 1988-2007, Oracle. All rights reserved.

```
Local XmlDoc &inXMLDoc;
Local XmlNode &docTypeNode;
Local XmlNode &rootNode;

&inXMLDoc = CreateXmlDoc("");
&docTypeNode = &inXMLDoc.CreateDocumentType("Personal", "", "Personal.dtd");
&rootNode = &inXMLDoc.CreateDocumentElement("root", "", &docTypeNode);
```

This produces the following XML:

```
<?xml version="1.0"?>
<!DOCTYPE Personal SYSTEM "Personal.dtd">
<root/>
```

See Also

Chapter 42, "XmlDoc Class," CreateDocumentElement, page 1756

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," CreateXmlDoc

GenFormattedXmlString

Syntax

GenFormattedXmlString()

Description

Use the GenFormattedXmlString method to return an XML string representing the XmlDoc object.

The GenFormattedXmlString method produces an XML string with new line characters and indents. If you want an unformatted XML string, that is, with all the data on a single line, use the GenXmlString method instead.

Parameters

None.

Returns

A formatted XML string.

Example

```
Local XmlDoc &inXMLDoc;
Local string &outStr;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><PSMessage/>");
&outStr = &inXMLDoc.GenFormattedXmlString();
```

See Also

Chapter 42, "XmlDoc Class," GenXmlString, page 1759

 Copyright © 1988-2007, Oracle. All rights reserved.

GenXmlString

Syntax

GenXmlString()

Description

Use the GenXmlString method to return an XML string representing the XmlDoc object.

The GenXmlString method produces an XML string with all the data on a single line. If you want a formatted XML string, that is, with new line characters and indents, use the GenFormattedXmlString method instead.

Parameters

None.

Returns

An XML string.

Example

```
Local XmlDoc &inXMLDoc;
Local string &outStr;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><PSMessage/>");
&outStr = &inXMLDoc.GenXmlString();
```

See Also

Chapter 42, "XmlDoc Class," GenFormattedXmlString, page 1758

GetElementsByTagName

Syntax

GetElementsByTagName(*TagName*)

Description

Use the GetElementsByTagName method to return an array of XmlNode objects that match the specified tag name.

If you specify an invalid tag name, an array is still returned, however, it has 0 elements in it.

Parameters

Parameter	Description
TagName	Specify the tag name that you want to look for.

Returns

An array of XmlNode objects.

Copyright © 1988-2007, Oracle. All rights reserved.

Example

Using the following Xml structure:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following code finds all of the *Field1* elements:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.GetElementsByTagName("Field1");

If &field1List.Len = 0 Then
   /* do error processing */
Else
   /* do processing */
End-If;
```

See Also

Chapter 8, “Array Class,” page 215

LoadIBContent

Syntax

LoadIBContent(*Content*[, *RootTagName*])

Description

Use LoadIBContent to load data into an XmlDoc object. This function is primarily used for display purposes, the main use is found in the Integration Broker Monitor for viewing message structures.

When *Content* is an XML string, the XmlDoc contains the DOM representation of the XML.

When *Content* is not an XML string, LoadIBContents generates a DOM wrapper.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Content	Specify the content of the XmlDoc.
RootTagName	Specify the root tag name to be used in the XmlDoc object used as the wrapper, if the data is non-Xml. If a root tag name is not specified, and the data is non-Xml, the contents are not wrapped with the default of root tag name of PSMessage.

Returns

This method returns a Boolean value that is always True.

Example

Using the following data:

```
John Q. Public,1234 West Eastland,925-987-0909
Jane Doe,423 Someplace,234-992-9383
```

The following PeopleCode program specifies a root name:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &inStr;

&inStr = "John Q. Public,1234 West Eastland,925-987-0909" | Char(13) | "Jane⇒
 Doe,423 Someplace,234-992-9383";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.LoadIBContent(&inStr, "Root");
```

This produces the following:

```
<?xml version="1.0"?>
<Root>
  <data PsNonXml="Yes">
    <![CDATA[John Q. Public,1234 West Eastland,925-987-0909
Jane Doe,423 Someplace,234-992-9383]]>
  </data>
</Root>
```

The following PeopleCode program does not specify a root name:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &inStr;

&inStr = "John Q. Public,1234 West Eastland,925-987-0909" | Char(13) | "Jane⇒
 Doe,423 Someplace,234-992-9383";

&inXMLDoc = CreateXmlDoc("");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&ret = &inXMLDoc.LoadIBContent(&inStr);
```

This produces the following:

```
<?xml version="1.0"?>
<PSMessage>
  <data PsNonXml="Yes">
    <![CDATA[John Q. Public,1234 West Eastland,925-987-0909
Jane Doe,423 Someplace,234-992-9383]]>
  </data>
</PSMessage>
```

The following is an example of a valid XML document:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &inStr;

&inStr = "<?xml version='1.0'?><root/>";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.LoadIBContent(&inStr);
```

This produces the following:

```
<?xml version="1.0"?>
<root/>
```

ParseXmlFromURL

Syntax

ParseXmlFromURL(*path* [, *DTDValidation*])

Where *path* can have *one* of the following forms—a string URL, containing the filename and extension:

```
http://www.peoplesoft.com/filename.ext
```

Or an absolute file path, including the filename and extension:

```
c:\directory\filename.ext
```

Note. HTTPS is not a supported protocol for *path*.

In Tru64 platforms, this method only accepts local file paths ("/tmp/test.xml"), not HTTP paths (http://myserver.com/test.xml).

Description

Use the ParseXmlFromURL method to convert the XML file located at *path* into an XmlDoc object that you can then manipulate using PeopleCode. The XmlDoc object executing the method is populated with the XML string after it's been converted. Any data already existing in the XmlDoc object is overwritten.

Using this method also does basic validation of the XML string, comparing it to the document type declaration (DTD) if the DOCTYPE for the DTD is provided in the XML string.

 Copyright © 1988-2007, Oracle. All rights reserved.

Note. PeopleSoft only supports UTF-8 encoding. Therefore, if the input file is encoded, it must be encoded in UTF-8.

Parameters

Parameter	Description
path	Specify the URL or file path to the XML that file you want to manipulate. If you specify a string URL, the URL must be contained in quotation marks. **Note.** HTTPS is not a supported protocol for *path*.
DTDValidation	Specify whether to validate a document type definition (DTD.) This parameter takes a boolean value. If you specify true, the DTD validation occurs if a DTD is provided. If you specify false, and if a DTD is provided, it is ignored and the XML isn't validated against the DTD. The default value for this parameter is false. In the case of application messaging, if a DTD is provided, it's always ignored and the XML isn't validated against the DTD. If the XML cannot be validated against a DTD, an error is thrown saying that there was an XML parse error.

Returns

A Boolean value: True, the XML file converted successfully and was validated, False otherwise. (If the XML file is not valid, the PeopleCode program terminates.)

Example

The following PeopleCode program loads a file from a file path:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.ParseXmlFromURL("c:\temp\in.xml");
```

The following PeopleCode program loads a file from a URL:

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.ParseXmlFromURL("http://www.peoplesoft.com/xmlfile.xml");
```

See Also

Chapter 42, "XmlDoc Class," ParseXmlString, page 1763

ParseXmlString

Syntax

ParseXmlString(*XmlString*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the ParseXmlString method to convert the XML string into an XmlDoc object that you can then manipulate using PeopleCode. The XmlDoc object executing the method is populated with the XML string after it's been converted. Any data already existing in the XmlDoc object is overwritten.

Using this method also does basic validation of the XML string, comparing it to the document type declaration (DTD) if the DOCTYPE for the DTD is provided in the XML string.

Parameters

Parameter	Description
XmlString	Specify an XML string that you want to be able to manipulate using PeopleCode.

Returns

A Boolean value: True, the XML string converted successfully and was validated, False otherwise. (If the XML is not valid, the PeopleCode program terminates.)

Example

```
Local XmlDoc &inXMLDoc;
Local boolean &ret;
Local string &str;

&str = "<?xml version='1.0'?><root/>";

&inXMLDoc = CreateXmlDoc("");
&ret = &inXMLDoc.ParseXmlString(&str);

If &ret Then
   /* do processing */
Else
   /* do error processing */
End-If;
```

See Also

Chapter 42, "XmlDoc Class," ParseXmlFromURL, page 1762

XmlDoc Properties

In this section, we discuss the XmlDoc properties. The properties are discussed in alphabetical order.

DocumentElement

Description

This property returns a reference to the root element of the document as an XmlNode.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

See Also
Chapter 42, "XmlDoc Class," XmlNode, page 1765

IsNull

Description

This property returns a Boolean value, indicating whether the doc is a valid object. This property returns True if the XmlDoc object is valid, False otherwise.

This property is read-only.

XmlNode

Unlike C++ and Java programming model, PeopleCode interface uses the root object XmlNode to represent all 11 different derived objects. An XmlNode object can be defined only as local.

XmlNode Methods

Add methods create a new node of the specific type, append this new node to the child list, and return this newly created node, while Insert methods add the node to the specified position.

In this section, we discuss the XmlNode methods. The methods are discussed in alphabetical order.

AddAttribute

Syntax
```
AddAttribute(Name, Value);
```

Description

Use the AddAttribute method to add an attribute to an XmlNode. The new attribute is appended to the list of child nodes. A reference to the newly created attribute is returned.

If you specify an attribute name that already exists, the new value you use *replaces* the existing value, and a reference to the attribute is returned.

Parameters

Parameter	Description
Name	Specify the name of the Attribute that you want to create, as a string.
Value	Specify the value of the Attribute that you want to create, as a string.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

Here is a set of XML response code.

```
<?xml version="1.0"?>
<myroot>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?"/>

         </user>
      </candidate>
   </postreqresponse>
</myroot>
```

Here's the PeopleCode that builds it.

```
Local XmlDoc &inXMLDoc;
Local XmlNode &postReqNode;
Local XmlNode &candidatesNode;
Local XmlNode &userNode;
Local XmlNode &locationNode;
Local XmlNode &sceneryAtt;
Local XmlNode &densityAtt;
Local XmlNode &blankAtt;
Local boolean &ret;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");

&postReqNode = &inXMLDoc.DocumentElement.AddElement("postreqresponse");
&candidatesNode = &postReqNode.AddElement("candidates");
&userNode = &candidatesNode.AddElement("user");
&locationNode = &userNode.AddElement("location");

&locationNode.AddAttribute("scenery", "great");
&locationNode.AddAttribute("density", "low");
&locationNode.AddAttribute("blank", "eh?");
```

See Also

Chapter 42, "XmlDoc Class," IsNull, page 1798

AddAttributeNS

Syntax

AddAttributeNS(*NamespaceURI*, *AttributeName*, *Value*)

where *NamespaceURI* can have *one* of the following forms:

 Copyright © 1988-2007, Oracle. All rights reserved.

URL.*URLname*

OR a string URL, such as
`http://www.peoplesoft.com/`

Description

Use the AddAttributeNS method to add a namespace attribute to an XmlNode. The new attribute is appended to the list of child nodes.

To insert a Namespace attribute at a particular place in the list of nodes, use the InsertAttributeNS method instead.

A reference to the newly created attribute is returned.

If you specify an attribute name that already exists, the new value you use *replaces* the existing value, and a reference to the attribute is returned.

Parameters

Parameter	Description
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc, as a string.
AttributeName	Specify the name of the Attribute that you want to create, as a string.
Value	Specify the value of the Attribute that you want to create, as a string.

Returns

None.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&childNode.AddAttributeNS("http://www.peoplesoft.com", "scenery", "great");
```

See Also

Chapter 42, "XmlDoc Class," IsNull, page 1798

AddCDataSection

Syntax

AddCDataSection(*Data*)

Description

Use the AddCDataSection to add a CDATA section to an XmlNode.

CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup.

Copyright © 1988-2007, Oracle. All rights reserved.

To insert a CDATA section at a particular place in the list of nodes, use the InsertCDataSection method instead.

A reference to the newly created CDATA section is returned.

Note. You cannot use this method with a SoapDoc object.

Parameters

Parameter	Description
Data	Specify the data to be included in the CDATA section as a string.

Returns

A reference to the newly created CDATA section.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &cdataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&cdataNode = &childNode.AddCDataSection("testing...");
```

produces the following:

```
<?xml version="1.0"?>
<myroot>
  <child>
    <![CDATA[testing...]]>
  </child>
</myroot>
```

See Also

Chapter 42, "XmlDoc Class," InsertCDataSection, page 1785

AddComment

Syntax

AddComment(*Text*)

Description

Use the AddComment method to add a comment to an XmlNode. The new attribute is appended to the list of child nodes.

To insert a comment at a particular place in the list of nodes, use the InsertComment method instead.

A reference to the newly created comment is returned.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Text	Specify the text to be used as the comment, as a string.

Returns

A reference to the newly created comment.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &commentNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&commentNode = &childNode.AddComment("This is an example of a comment.");
```

produces the following:

```
<?xml version="1.0"?>
<myroot>
  <child>
    <!--This is an example of a comment.-->
  </child>
</myroot>
```

See Also

Chapter 42, "XmlDoc Class," InsertComment, page 1786

AddElement

Syntax

AddElement(*TagName*)

Description

Use the AddElement method to add an element to the XmlNode. The new element is appended to the list of child nodes.

To insert an element at a particular place in the list of nodes, use the InsertElement method instead.

A reference to the newly created element is returned.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TagName	Specify the tag for the new element that you are adding, as a string.

Returns

A reference to the newly created element.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
```

produces the following:

```
<?xml version="1.0"?>
<myroot>
  <child/>
</myroot>
```

See Also

Chapter 42, "XmlDoc Class," InsertElement, page 1787, Chapter 42, "XmlDoc Class," GetElement, page 1781 and Chapter 42, "XmlDoc Class," GetElementsByTagName, page 1783.

AddElementNS

Syntax

AddElementNS(*NamespaceURI, TagName*)

Where *NamespaceURI* can have *one* of the following forms:

URL.*URLName*

Or a string URL, such as:

`http://www.peoplesoft.com/`

Description

Use AddElementNS to add a namespace element to an XmlNode. The new element is appendced to the list of child nodes.

To insert a Namespace element at a particular place in the list of nodes, use the InsertElementNS method instead. A reference to the newly created element is returned. If you specify an element name that already exists, the new value you use replaces the existing value, and a reference to the element is returned.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc, as a string.
TagName	Specify the name of the Element that you want to create, as a string.

Returns

A reference to the newly created element if successful. If not successful, the IsNull property is set to True.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElementNS("http://www.peoplesoft.com",⇒
 "child");
```

AddEntityReference

Syntax

AddEntityReference(*Name*)

Description

Use the AddEntityReference method to add an entity reference. An entity reference refers to the content of the named entity. The new entity reference is appended to the list of child nodes. In the generated XML, the entity reference is automatically prefaced with the '&' character and suffixed with a semi-colon.

The named entity must exist in the document type declaration (DTD).

To insert an entity reference at a particular place in the list of nodes, use the InsertEntityReference method instead.

Parameters

Parameter	Description
Name	Specify the name of the entity to which this reference refers to.

Returns

A reference to the newly created entity reference.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &entityRef;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&entityRef = &childNode.AddEntityReference("MyURL");
```

produces the following:

```
<?xml version="1.0"?>
<myroot>
  <child>
&MyURL;  </child>
</myroot>
```

See Also

Chapter 42, "XmlDoc Class," InsertEntityReference, page 1790

AddNode

Syntax

AddNode(*&XmlNode*)

Description

Use the AddNode method to add an XmlNode to the XmlDoc. The new node is appended to the list of child nodes.

To insert a node at a particular place in the list of nodes, use the InsertNode method instead.

A reference to the newly created node is returned.

Parameters

Parameter	Description
&XmlNode	Specify an already instantiated XmlNode that you want to add.

Returns

None.

Example

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><child><subchild/></child><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&childNode = &firstDoc.DocumentElement.FindNode("/child");
&inXMLDoc.DocumentElement.AddNode(&childNode);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also
Chapter 42, “XmlDoc Class,” InsertNode, page 1791

AddProcessInstruction

Syntax
AddProcessInstruction(*Target*, *Data*)

Description

Use the AddProcessInstruction method to add a processing instruction to an XmlNode. The new process instruction is appended to the end of the child list.

To insert a processing instruction at a particular place, use the InsertProcessInstruction method instead.

A reference to the newly created processing instruction is returned.

Note. You cannot use this method with a SoapDoc object.

Parameters

Parameter	Description
Target	Specify the application to which the instruction is directed, as a string.
Data	Specify the data used with the processing instruction.

Returns

A reference to the newly created processing instruction.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &procInst;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&procInst = &inXMLDoc.DocumentElement.AddProcessInstruction("xml-stylesheet",⇒
 "href=""book.css"" type=""text/css""");
```

produces the following:

```
<?xml version="1.0"?>
<myroot>
  <?xml-stylesheet href="book.css" type="text/css"?>
</myroot>
```

See Also
Chapter 42, “XmlDoc Class,” InsertProcessInstruction, page 1792

Copyright © 1988-2007, Oracle. All rights reserved.

AddText

Syntax

AddText(*Data*)

Description

Use the AddText method to add a text node to an XmlNode.

Note. A text node is *not* the same as a text declaration. The text declaration is added automatically with the CreateXmlDoc function. A text node just contains text within an XmlDoc.

To insert a text node at a particular place, use the InsertText method instead.

A reference to the newly created text node is returned.

Parameters

Parameter	Description
Data	Specify the data to be used as the text node, as a string.

Returns

A reference to the newly created text node.

Example

The following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &childNode;
Local XmlNode &textNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><myroot/>");
&childNode = &inXMLDoc.DocumentElement.AddElement("child");
&textNode = &childNode.AddText("This is text");
```

produces the following:

```
<?xml version="1.0"?>
<myroot>
  <child>This is text</child>
</myroot>
```

See Also

Chapter 42, "XmlDoc Class," InsertText, page 1794

CopyNode

Syntax

CopyNode(*&Node*)

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the CopyNode method to copy a node specified by *&Node* and *all* of its child nodes into the current node, as a child node. The new node is appended to the end of the list of child nodes. The specified node can belong to a different XmlDoc.

If you just want to copy a top-level node, without copying its children, you can just create a new node with the name of the node you want.

Parameters

Parameter	Description
&Node	Specify an already instantiated node that you want to copy.

Returns

None.

Example

Suppose that your XmlDoc has the following structure:

```
<?xml version="1.0"?>
<myroot>
  <child>
    <subchild/>
  </child>
</myroot>
```

The following PeopleCode copies the node *child* and copies it into another doc.

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><child><subchild/></child><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&childNode = &firstDoc.DocumentElement.FindNode("/child");
&inXMLDoc.DocumentElement.CopyNode(&childNode);
```

The following is the new structure:

```
<?xml version="1.0"?>
<root>
  <child>
    <subchild/>
  </child>
</root>
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 42, “XmlDoc Class,” AddNode, page 1772, Chapter 42, “XmlDoc Class,” InsertNode, page 1791, Chapter 42, “XmlDoc Class,” GetChildNode, page 1781, Chapter 42, “XmlDoc Class,” FindNode, page 1776 and Chapter 42, “XmlDoc Class,” FindNodes, page 1777.

FindNode

Syntax

FindNode(*Path*)

Description

Use the FindNode method to return a reference to an XmlNode.

The path is specified as the list of tag names, to the node that you want to find, each separated by a slash (/).

Parameters

Parameter	Description
Path	Specify the tag names up to and including the name of the node that you want returned, starting with a slash and each separated by a slash (/). This is known as the XPath query language. See http://www.w3.org/TR/xpath

Returns

An XmlNode matching the path if successful. If not successful, the IsNull property on the XmlNode is set to True.

Example

Suppose your XmlDoc has the following structure:

```
<?xml version="1.0"?>
<myroot>
   <postreqresponse>
      <candidate>
         <user>
            <location scenery="great" density="low" blank="eh?"/>
         </user>
      </candidate>
   </postreqresponse>
</myroot>
```

You want to return a reference to the location attribute. Use the following *Path* to do it:

```
&XmlNode = &MyDoc.FindNode("/myroot/postreqresponse/candidate/user/location");
```

See Also

Chapter 42, “XmlDoc Class,” GetChildNode, page 1781 and Chapter 42, “XmlDoc Class,” FindNodes, page 1777.

Copyright © 1988-2007, Oracle. All rights reserved.

FindNodes

Syntax

FindNodes(*Path*)

Description

Use the FindNodes method to return a reference to an array containing all the XmlNodes that match the path.

The path is specified as the list of tag names, to the node that you want to find, each separated by a slash (/).

Parameters

Parameter	Description
Path	Specify the tag names up to and including the name of the node that you want returned, each separated by a slash (/). This is known as the XPath query language. See http://www.w3.org/TR/xpath

Returns

An array of XmlNode objects. If there is no match for the specified path, an array with 0 elements is returned.

Example

Using the following input:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following PeopleCode program finds all of the *Field1* nodes:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.DocumentElement.FindNodes("MsgData/Transaction/Record1⇒
/Field1");

If &field1List.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 42, "XmlDoc Class," GetChildNode, page 1781, Chapter 42, "XmlDoc Class," FindNode, page 1776 and Chapter 8, "Array Class," page 215.

GenXmlString

Syntax

GenXmlString()

Description

Use the GenXmlString method to generate an XML string of the XmlNode.

Parameters

None.

Returns

An XML String.

Example

```
&MyXML = &MyNode.GenXmlString();
```

See Also

Chapter 42, "XmlDoc Class," GenXmlString, page 1778
XmlDoc method

GetAttributeName

Syntax

GetAttributeName(*Index*)

Description

Use the GetAttributeName method to return the specified attribute.

Parameters

Parameter	Description
Index	Specify an integer representing the attribute you want to access.

Returns

A string.

Example

```
Local XmlDoc &inXMLDoc;
Local string &attName;
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&inXMLDoc.DocumentElement.AddAttribute("name", "Joe");
&inXMLDoc.DocumentElement.AddAttribute("address", "1234 West Eastland");

&attName = &inXMLDoc.DocumentElement.GetAttributeName(2);
```

GetAttributeValue

Syntax

GetAttributeValue({*Name* | *Index*})

Description

Use the GetAttributeValue method to return the specific attribute value, given an attribute name. The attribute name is case-sensitive, so you must specify the exact name.

Parameters

Parameter	Description
Name | *Index*	Specify the name or index of the attribute whose value you want to access.

Returns

A string containing the value of the specified attribute.

Example

```
Local XmlDoc &inXMLDoc;
Local string &attName;
Local string &attValue;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&inXMLDoc.DocumentElement.AddAttribute("name", "Joe");
&inXMLDoc.DocumentElement.AddAttribute("address", "1234 West Eastland");

&attName = &inXMLDoc.DocumentElement.GetAttributeName(2);
&attValue = &inXMLDoc.DocumentElement.GetAttributeValue(&attName);
```

GetCDataValue

Syntax

GetCDataValue()

Description

Use the GetCDataValue method to return the *value* of the first CDATA section in an XmlNode.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to the value of the first CDATA section as a string.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &cdataNode;
Local string &theData;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&cdataNode = &dataNode.AddCDataSection("this is a bunch of text");

&theData = &dataNode.GetCDataValue();
```

See Also

Chapter 42, "XmlDoc Class," AddCDataSection, page 1767, Chapter 42, "XmlDoc Class," InsertCDataSection, page 1785 and Chapter 42, "XmlDoc Class," GetCDataValues, page 1780.

GetCDataValues

Syntax

GetCDataValues()

Description

Use the GetCDataValues method to return an array of string containing the *values* of the CDATA section in an XmlNode.

Parameters

None.

Returns

An array of string.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &cdataNode;
Local array of string &theData;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&cdataNode = &dataNode.AddCDataSection("this is a bunch of text");
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&cdataNode = &dataNode.AddCDataSection("more text");
&cdataNode = &dataNode.AddCDataSection("still more text");

&theData = &dataNode.GetCDataValues();
```

See Also

Chapter 42, "XmlDoc Class," AddCDataSection, page 1767, Chapter 42, "XmlDoc Class," InsertCDataSection, page 1785, Chapter 42, "XmlDoc Class," GetCDataValue, page 1779 and Chapter 8, "Array Class," page 215.

GetChildNode

Syntax

GetChildNode(*index*)

Description

Use the GetChildNode method to return the specified child node of an XmlNode.

Parameters

Parameter	Description
Index	Specify an integer representing the child node that you want to access.

Returns

A reference to an XmlNode. If the specified XmlNode isn't found, the IsNull property for the XmlNode is set to True.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data2");

&dataNode = &inXMLDoc.DocumentElement.GetChildNode(2);
```

See Also

Chapter 42, "XmlDoc Class," FindNode, page 1776 and Chapter 42, "XmlDoc Class," FindNodes, page 1777.

GetElement

Syntax

GetElement()

Description

Use the GetElement method to return the first child element node in the list of child nodes.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to an element.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data2");

&dataNode = &inXMLDoc.DocumentElement.GetElement();
```

See Also

Chapter 42, "XmlDoc Class," AddElement, page 1769, Chapter 42, "XmlDoc Class," InsertElement, page 1787, Chapter 42, "XmlDoc Class," GetElements, page 1782 and Chapter 42, "XmlDoc Class," GetElementsByTagName, page 1783.

GetElements

Syntax

GetElements()

Description

Use the GetElements method to return an array of all the XmlNode objects that are elements.

Parameters

None.

Returns

An array of XmlNode of all the element nodes. If there are no element nodes, an array with 0 elements is returned.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local array of XmlNode &dataList;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data2");

&dataList = &inXMLDoc.DocumentElement.GetElements();
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
If &dataList.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

See Also

Chapter 42, "XmlDoc Class," AddElement, page 1769, Chapter 42, "XmlDoc Class," InsertElement, page 1787, Chapter 42, "XmlDoc Class," GetElement, page 1781, Chapter 42, "XmlDoc Class," GetElementsByTagName, page 1783 and Chapter 8, "Array Class," page 215.

GetElementsByTagName

Syntax

GetElementsByTagName(*TagName*)

Description

Use the GetElementsByTagName method to return an array of XmlNode objects that match the specified tag name.

Parameters

Parameter	Description
TagName	Specify the tag name that you want to look for.

Returns

An array of XmlNode. If you specify an invalid tag name, the returned array has 0 elements.

Example

Using the following input:

```
<?xml version="1.0"?>
<PSmessage>
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following PeopleCode program finds all *Field1* nodes:

```
Local array of XmlNode &field1List;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&field1List = &inXMLDoc.DocumentElement.GetElementsByTagName("Field1");

If &field1List.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

See Also

Chapter 42, "XmlDoc Class," AddElement, page 1769, Chapter 42, "XmlDoc Class," InsertElement, page 1787, Chapter 42, "XmlDoc Class," GetElements, page 1782, Chapter 42, "XmlDoc Class," GetElement, page 1781 and Chapter 8, "Array Class," page 215.

GetElementsByTagNameNS

Syntax

GetElementsByTagNameNS(*NamespaceURI*, *TagName*)

Where *NamespaceURI* can have *one* of the following forms:

URL.*URLName*

Or a string URL, such as:

`http://www.peoplesoft.com/`

Description

Use the GetElementsByTagNameNS method to return an array of XmlNode objects that match the specified namespace and qualified name.

Parameters

Parameter	Description
NamespaceURI	Specify the URI that contains the Namespace for the XmlDoc, as a string.
TagName	Specify the name of the element that you want to retrieve.

Returns

An array of XmlNode. If you specify an invalid NamespaceURI, the returned array has 0 elements.

Example

Using the following input:

```
<?xml version="1.0"?>
<Psmessage xmlns="http://www.peoplesoft.com">
    <MsgData>
        <Transaction>
            <Record1 class="R">
                <Field1>one</Field1>
                <Field1>two</Field1>
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
                <Field1>three</Field1>
            </Record1>
        </Transaction>
    </MsgData>
</PSmessage>
```

The following PeopleCode program finds all *Field1* nodes:

```
Local array of XmlNode &field1List;

&field1List = &inXMLDoc.DocumentElement.GetElementsByTagNameNS("http:⇒
//www.peoplesoft.com", "Field1");

If &field1List.Len = 0 Then
   /* do error processing, no nodes returned */
Else
   /* do regular processing */
End-If;
```

InsertCDataSection

Syntax

InsertCDataSection(*Data*, *Position*)

Description

Use the InsertCDataSection method to insert a CDATA section in an XmlNode, at the location specified by *Position*.

Note. You cannot use this method with a SoapDoc object.

Parameters

Parameter	Description
Data	Specify the data for the CDATA section as a string.
Position	Specify where you want to insert the CDATA Section, as a number.

Returns

A reference to the newly created CDATA section.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &cdataNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&cdataNode = &dataNode.AddCDataSection("this is a bunch of text");
&cdataNode = &dataNode.AddCDataSection("still more text");

&cdataNode = &dataNode.InsertCDataSection("more text", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <![CDATA[this is a bunch of text]]>
    <![CDATA[still more text]]>
  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <![CDATA[this is a bunch of text]]>
    <![CDATA[more text]]>
    <![CDATA[still more text]]>
  </data>
</root>
```

See Also

Chapter 42, "XmlDoc Class," AddCDataSection, page 1767, Chapter 42, "XmlDoc Class," GetCDataValue, page 1779 and Chapter 42, "XmlDoc Class," GetCDataValues, page 1780.

InsertComment

Syntax

InsertComment(*Data, Position*)

Description

Use the InsertComment method to insert a comment in an XmlNode at the location specified by *Position*.

Parameters

Parameter	Description
Data	Specify the data for the comment as a string.
Position	Specify where you want to insert the comment, as a number.

Returns

A reference to the newly created comment.

 Copyright © 1988-2007, Oracle. All rights reserved.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &commentNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&commentNode = &dataNode.AddComment("this is a comment");
&commentNode = &dataNode.AddComment("still another comment");

&commentNode = &dataNode.InsertComment("more comments", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <!--this is a comment-->
    <!--still another comment-->
  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <!--this is a comment-->
    <!--more comments-->
    <!--still another comment-->
  </data>
</root>
```

See Also

Chapter 42, "XmlDoc Class," AddComment, page 1768

InsertElement

Syntax

InsertElement(*TagName*, *Position*)

Description

Use the InsertElement method to insert an element in an XmlNode at the location specified by *Position*.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TagName	Specify the tagname for the element as a string.
Position	Specify where you want to insert the element, as a number.

Returns

A reference to the newly created element.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&elementNode = &dataNode.AddElement("first");
&elementNode = &dataNode.AddElement("second");

&commentNode = &dataNode.InsertElement("third", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <first/>
    <second/>
  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
    <first/>
    <third/>
    <second/>
  </data>
</root>
```

See Also

Chapter 42, "XmlDoc Class," AddElement, page 1769

 Copyright © 1988-2007, Oracle. All rights reserved.

InsertElementNS

Syntax

InsertElementNS(*NamespaceURI*, *TagName*, *Position*)

where *NamespaceURI* can have *one* of the following forms:

URL.*URLname*

OR a string URL, such as

`http://www.peoplesoft.com/`

Description

Use InsertElementNS to insert a new element in an XmlNode, at the location specified by *Position*.

Parameters

Parameter	Description
NamespaceURI	Specify the URI that contains the Namespaces for the XmlDoc.
TagName	Specify the name of the element that you want to create, as a string.
Position	Specify where you want to insert the element, as a number.

Returns

A reference to the newly created element.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root xmlns='http:⇒
//www.peoplesoft.com'/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&elementNode = &dataNode.AddElement("first");
&elementNode = &dataNode.AddElement("second");

&elementNode = &dataNode.InsertElementNS("http://www.peoplesoft.com", "third", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root xmlns="http://www.peoplesoft.com">
  <data>
    <first/>
    <second/>
  </data>
</root>
```

Copyright © 1988-2007, Oracle. All rights reserved.

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root xmlns="http://www.peoplesoft.com">
  <data>
    <first/>
    <third/>
    <second/>
  </data>
</root>
```

InsertEntityReference

Syntax

InsertEntityReference(*Name*, *Position*)

Description

Use the InsertEntityReference method to add an entity reference. An entity reference refers to the content of the named entity.

Parameters

Parameter	Description
Name	Specify the name of the entity to which this reference refers to.
Position	Specify where you want to insert the entity reference, as a number.

Returns

A reference to the newly created entity reference.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &dataNode;
Local XmlNode &entityRef;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&dataNode = &inXMLDoc.DocumentElement.AddElement("data");
&entityRef = &dataNode.AddEntityReference("first");
&entityRef = &dataNode.AddEntityReference("second");

&entityRef = &dataNode.InsertEntityReference("third", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
```


Copyright © 1988-2007, Oracle. All rights reserved.

```
<root>
  <data>
&first;&second;  </data>
</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <data>
&first;&third;&second;  </data>
</root>
```

See Also

Chapter 42, "XmlDoc Class," AddEntityReference, page 1771

InsertNode

Syntax

InsertNode(*&NewNode*, {*&RefNode* | *Position*})

Description

Use the InsertNode method to insert a new node. The node is inserted either before the node specified by *&RefNode*, or at the location specified by *Position*.

&NewNode refers to an already instantiated XmlNode object. You must create the node before you can insert it.

&RefNode refers to an already instantiated XmlNode object.

Parameters

Parameter	Description
&NewNode	Specify the name of an already instantiated XmlNode that you want to insert.
&RefNode | *Position*	Specify either the name of an existing node or the location where you want the node to be inserted.

Returns

A reference to the newly inserted node.

Example

The following PeopleCode program inserts a node using a position:

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><third><child/></third><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root><first/><second/></root>");
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&childNode = &firstDoc.DocumentElement.FindNode("/third");

&inXMLDoc.DocumentElement.InsertNode(&childNode, 2);
```

The following PeopleCode program inserts a node using a reference node:

```
Local XmlDoc &inXMLDoc, &firstDoc;
Local XmlNode &childNode, &secondNode;

&firstDoc = CreateXmlDoc("<?xml version='1.0'?><myroot><third><child/></third><⇒
/myroot>");
&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root><first/><second/></root>");
&childNode = &firstDoc.DocumentElement.FindNode("/third");
&secondNode = &inXMLDoc.DocumentElement.FindNode("/second");

&inXMLDoc.DocumentElement.InsertNode(&childNode, &secondNode);
```

The following is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <first/>
  <second/>
</root>
```

The following is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
  <first/>
  <third>
    <child/>
  </third>
  <second/>
</root>
```

See Also

Chapter 42, "XmlDoc Class," AddNode, page 1772

InsertProcessInstruction

Syntax

InsertProcessInstruction(*Target*, *Data*, *Position*)

Description

Use the InsertProcessInstruction method to insert a processing instruction to an XmlNode at the location specified by *Position*.

Note. Youcannot use this method with a SoapDoc object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Target	Specify the application to which the instruction is directed, as a string.
Data	Specify the data to be used with the instruction.
Position	Specify where you want to insert the process instruction, as a number.

Returns

A reference to the newly created processing instruction.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &procInst;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&procInst = &inXMLDoc.DocumentElement.AddProcessInstruction("first", "firstvalue");
&procInst = &inXMLDoc.DocumentElement.AddProcessInstruction("second",⇒
 "secondvalue");

&procInst = &inXMLDoc.DocumentElement.InsertProcessInstruction("third",⇒
 "thirdvalue", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
  <?first firstvalue?>
  <?second secondvalue?>
</root>
```

This is the XML document after the inser:

```
<?xml version="1.0"?>
<root>
  <?first firstvalue?>
  <?third thirdvalue?>
  <?second secondvalue?>
</root>
```

See Also

Chapter 42, "XmlDoc Class," AddProcessInstruction, page 1773

Copyright © 1988-2007, Oracle. All rights reserved.

InsertText

Syntax

InsertText(*Data*, *Position*)

Description

Use the InsertText method to insert a text node. The node is inserted at the location indicated by *Position*.

Note. A text node is *not* the same as a text declaration. The text declaration is added automatically with the CreateXmlDoc function. A text node just contains text within an XmlDoc.

Parameters

Parameter	Description
Data	Specify the data to be used for the text node, as a string.
Position	Specify where you want to insert the text ndoe, as a number.

Returns

A reference to the newly created text node.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &textNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&textNode = &inXMLDoc.DocumentElement.AddText("first text");
&textNode = &inXMLDoc.DocumentElement.AddText("second text");

&textNode = &inXMLDoc.DocumentElement.InsertText("third text", 2);
```

This is the XML document before the insert:

```
<?xml version="1.0"?>
<root>
first textsecond text</root>
```

This is the XML document after the insert:

```
<?xml version="1.0"?>
<root>
first textthird textsecond text</root>
```

See Also

Chapter 42, "XmlDoc Class," AddText, page 1774

 Copyright © 1988-2007, Oracle. All rights reserved.

RemoveAllChildNode

Syntax

RemoveAllChildNode()

Description

Use the RemoveAllChildNode method to remove all child nodes for an XmlNode.

Parameters

None.

Returns

None.

Example

Using the following PeopleCode program:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");
&elementNode = &inXMLDoc.DocumentElement.AddElement("second");
&elementNode = &inXMLDoc.DocumentElement.AddElement("third");

&inXMLDoc.DocumentElement.RemoveAllChildNode();
```

This is the XML document before the removal:

```
<?xml version="1.0"?>
<root>
  <first/>
  <second/>
  <third/>
</root>
```

This is the XML document after the removal:

```
<?xml version="1.0"?>
<root/>
```

See Also

Chapter 42, "XmlDoc Class," AddNode, page 1772, Chapter 42, "XmlDoc Class," InsertNode, page 1791, Chapter 42, "XmlDoc Class," GetChildNode, page 1781, Chapter 42, "XmlDoc Class," FindNode, page 1776, Chapter 42, "XmlDoc Class," FindNodes, page 1777 and Chapter 42, "XmlDoc Class," RemoveChildNode, page 1796.

Copyright © 1988-2007, Oracle. All rights reserved.

RemoveChildNode

Syntax

RemoveChildNode({*Position* | *Node*})

Description

Use the RemoveChildNode method to remove the child node in an XmlNode specified by *Node*, or located at *Position*.

Parameters

Parameter	Description
Position | *Node*	Specify the child node that you want to delete, either using its position number or its name.

Returns

A reference to the removed XmlNode.

Example

The following PeopleCode program uses a position:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode, &removedNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");
&elementNode = &inXMLDoc.DocumentElement.AddElement("second");
&elementNode = &inXMLDoc.DocumentElement.AddElement("third");

&removedNode = &inXMLDoc.DocumentElement.RemoveChildNode(2);
```

The following PeopleCode program uses a reference node:

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode, &removedNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");
&elementNode = &inXMLDoc.DocumentElement.AddElement("second");
&elementNode = &inXMLDoc.DocumentElement.AddElement("third");

&elementNode = &inXMLDoc.DocumentElement.FindNode("/second");
&removedNode = &inXMLDoc.DocumentElement.RemoveChildNode(&elementNode);
```

This is the XML document before the removal:

```
<?xml version="1.0"?>
<root>
  <first/>
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
  <second/>
  <third/>
</root>
```

This is XML document after the removal:

```
<?xml version="1.0"?>
<root>
  <first/>
  <third/>
</root>
```

See Also

Chapter 42, “XmlDoc Class,” AddNode, page 1772, Chapter 42, “XmlDoc Class,” InsertNode, page 1791, Chapter 42, “XmlDoc Class,” GetChildNode, page 1781, Chapter 42, “XmlDoc Class,” FindNode, page 1776, Chapter 42, “XmlDoc Class,” FindNodes, page 1777 and Chapter 42, “XmlDoc Class,” RemoveAllChildNode, page 1795.

XmlNode Properties

The following are the XmlNode properties.

AttributesCount

Description

This property returns the number of attributes for an XmlNode, as a number.

This property is read-only.

Example

```
Local XmlDoc &inXMLDoc;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&inXMLDoc.DocumentElement.AddAttribute("first", "John");
&inXMLDoc.DocumentElement.AddAttribute("last", "Public");
&inXMLDoc.DocumentElement.AddAttribute("address", "1234 West Eastland");

For &i = 1 To &inXMLDoc.DocumentElement.AttributesCount
   /* do some processing of the attributes here */
End-For;
```

ChildNodeCount

Description

This property returns the number of child nodes for an XmlNode, as a number.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

Index

Description

This property returns the location (or position) of a node in the parent's child list, as a number. The root node always returns a 1.

This property is read-only.

IsNull

Description

This property returns a Boolean value, indicating whether the node is a valid node. Use this property to verify the value of the add or insert XmlNode methods.

This property is read-only.

Example

```
Local XmlDoc &inXMLDoc;
Local XmlNode &elementNode;

&inXMLDoc = CreateXmlDoc("<?xml version='1.0'?><root/>");
&elementNode = &inXMLDoc.DocumentElement.AddElement("first");

&elementNode = &inXMLDoc.DocumentElement.FindNode("/second");

If &elementNode.IsNull Then
   /* do some error processing here, will not find "second" */
End-If;
```

LocalName

Description

This property returns the local part of a namespace as a string. The local part is the second part of the namespace entry. If you want access to the first part of a namespace, use the Prefix property.

This property is read-only.

See Also

Chapter 42, "XmlDoc Class," Prefix, page 1800

NamespaceURI

Description

This property returns a reference to the namespace URI as a string.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

NextSibling

Description

This property returns the next child node in an XmlNode, as an XmlNode.

Use the IsNull property to verify that the XmlNode that is returned is valid.

This property is read-only.

See Also

Chapter 42, "XmlDoc Class," IsNull, page 1798

NodeName

Description

This property returns the name of the XmlNode as a string. You can use this property to rename an XmlNode.

This property is read-write.

NodePath

Description

This property returns the path of the XmlNode, from the root.

This property is read-only.

NodeType

Description

This property returns the type of the XmlNode. You can use either a numeric value or a constant. The values are:

Numeric Value	Constant Value	Description
1	%ElementNode	Element node
2	%AttributeNode	Attribute node
3	%TextNode	Text node
4	%CDataSectionNode	CData section node
5	%EntityReferenceNode	Entity reference node
6	%EntityNode	Entity node
7	%ProcessingInstructionNode	Processing instruction node
8	%CommentNode	Comment node
9	%DocumentNode	Document node
10	%DocumentTypeNode	Document type node
11	%NotationNode	Notation node
12	%XmlDeclNode	XML DECL node

Copyright © 1988-2007, Oracle. All rights reserved.

NodeValue

Description

This property returns the value of the XmlNode, as a string.

This property is read-write.

ParentNode

Description

This property returns a reference to the parent node for the XmlNode, as an XmlNode object.

Use the IsNull property to verify that the XmlNode that is returned is valid.

This property is read-only.

See Also

Chapter 42, “XmlDoc Class,” IsNull, page 1798

Prefix

Description

This property returns the prefix part of a namespace as a string. The prefix is the first part of the namespace entry. If you want access to the second part of a namespace, use the LocalName property.

This property is read-only.

See Also

Chapter 42, “XmlDoc Class,” LocalName, page 1798

PreviousSibling

Description

This property returns a reference to the previous node in the XmlNode, as an XmlNode object.

Use the IsNull property to verify that the XmlNode that is returned is valid.

This property is read-only.

See Also

Chapter 42, “XmlDoc Class,” IsNull, page 1798

 Copyright © 1988-2007, Oracle. All rights reserved.

CHAPTER 43

XML Publisher Classes

This chapter provides an overview of the PeopleSoft XML Publisher classes, and discusses the following topics:

- XML Publisher terms
- Life-cycles of the XML Publisher classes
- XML Publisher classes hierarchies
- Error handling
- Data types of XML Publisher objects
- Scope of XML Publisher objects
- How to import XML Publisher classes
- How to create XML Publisher objects
- XML Publisher classes language reference
- XML Publisher classes examples

See Also

Enterprise PeopleTools 8.49 PeopleBook: XML Publisher for PeopleSoft Enterprise, "Getting Started with XML Publisher"

Understanding the PeopleSoft XML Publisher and the XML Publisher Classes

The PeopleSoft XML Publisher enables you to separate the data, layout, and translation layers of a report from each other. This can improve flexibility, as well as reduce maintenance. You need to create report definitions, define templates, and so on, using the XML publisher. The XML Publisher classes enable you to access the runtime portions of the XML publishing process programmatically, that is, after the templates and reports have been created.

The XML Publisher classes are divided into the following categories:

- Report manager definition classes
- Report manager search classes
- XML generation classes
- XML Publisher engine classes

Copyright © 1988-2007, Oracle. All rights reserved.

Report Manager Definition Classes

You need to create a report definition for use by the XML Publisher and associates it with a data source using the XML Publisher. You also need to create templates and associate them with a report. Then at runtime, the definitional information is called by the page viewer or processing program to and fed to the XML Publisher core engine to produce the formatted output. You can specify the runtime parameters either using the XML Publisher or the XML Publisher report manager definition classes.

See Chapter 43, "XML Publisher Classes," Report Manager Definition Classes Constructors, page 1810.

Report Manager Search Classes

After you've published a report, it is stored in the report repository. Using the report manager search classes, you can search for a report based on defined search keys, or even add additional search keys for searching.

See Chapter 43, "XML Publisher Classes," Report Manager Search Classes Constructors, page 1848.

XML Generation Classes

Use the XML generation classes to generate XML sample data and XML schema for rowset and PeopleSoft query data.

See Chapter 43, "XML Publisher Classes," XML Generation Class Constructors, page 1858.

XML Publisher Engine Classes

Use the XML Publisher engine classes to directly access the Oracle XML Publiser APIs, as well as to specify page numbers and watermarks on merged PDF reports.

See Chapter 43, "XML Publisher Classes," XML Publisher Engine Class Constructors, page 1864.

XML Publisher Terms

The following is a list of general XML publisher terms and their definitions.

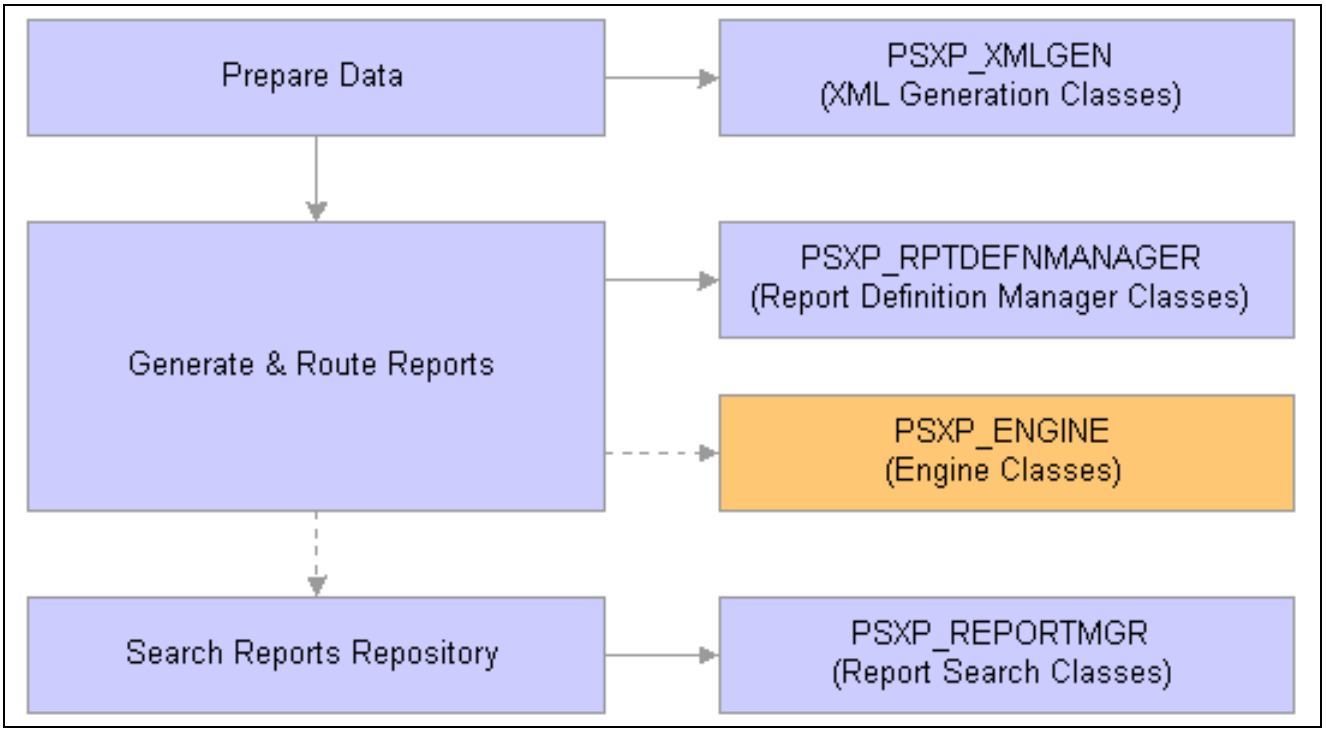
EFT	Electronic Funds Transfer, a format used by the banking industry.
eText	Electronic text, specifically, files in ASCII text format.
FO	Formatting objects, an XML markup language for document formatting.
HTML	Hyper Text Markup Language, a markup language designed for the creation of web pages.
PDF	Portable Document Format, a file format developed by Adobe Systems for representing documents in an independent format.
RTF	Rich Text Format, a document file format used for cross-platform document interchange.
XML	Extendable Markup Language, a general purpose markup language for many types of data.
SMS	Short messaging service, a short alphanumeric (160 characters) that is generally sent to a mobile device, such as a mobile phone.

 Copyright © 1988-2007, Oracle. All rights reserved.

Life-Cycles of the XML Publisher Classes

The following is the general flow of the XML Publisher classes, how and when they're used in the publishing process.

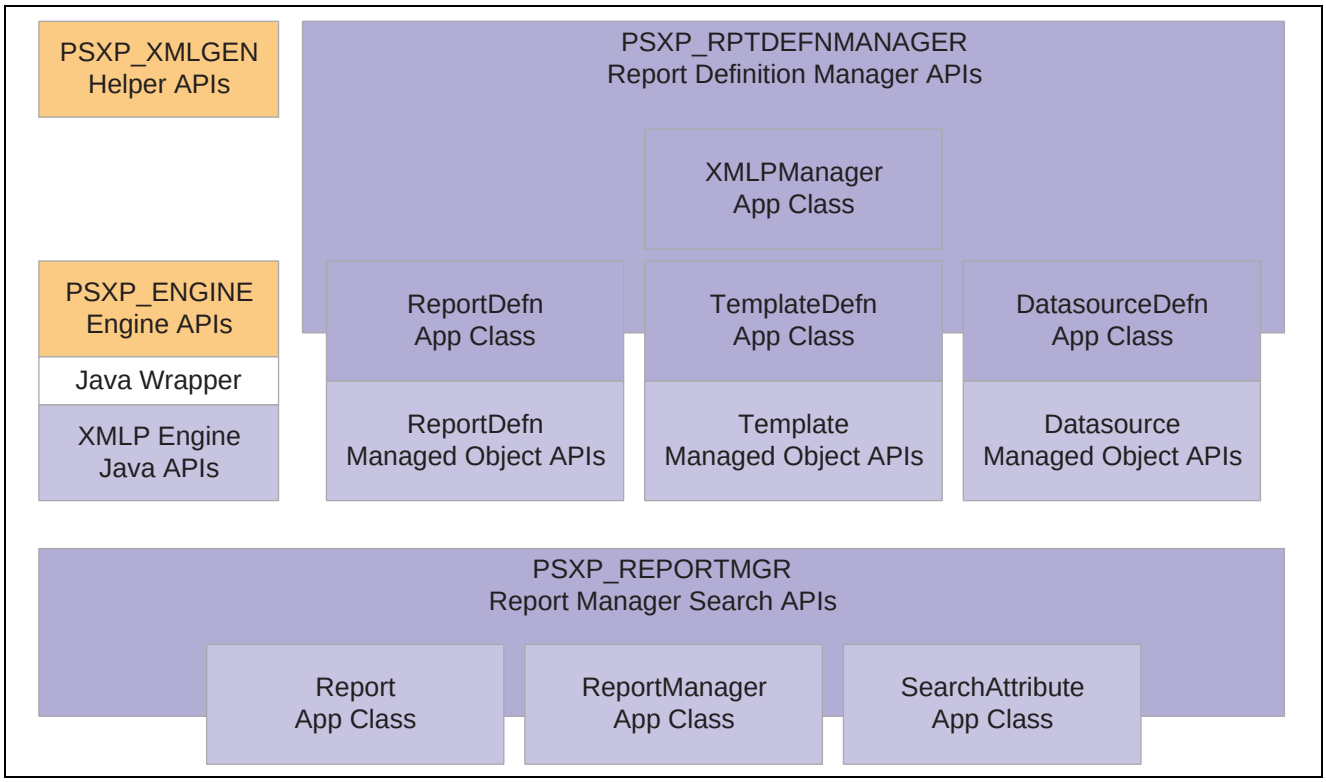
For most reports, you do not need to use the XML Publisher engine classes. In addition, you may or may not use the report manager search classes after a report is published.



XML Publisher classes life-cycle flow

The following is a representation of how the different classes relate to each other.

Copyright © 1988-2007, Oracle. All rights reserved.



XML Publisher classes relationships

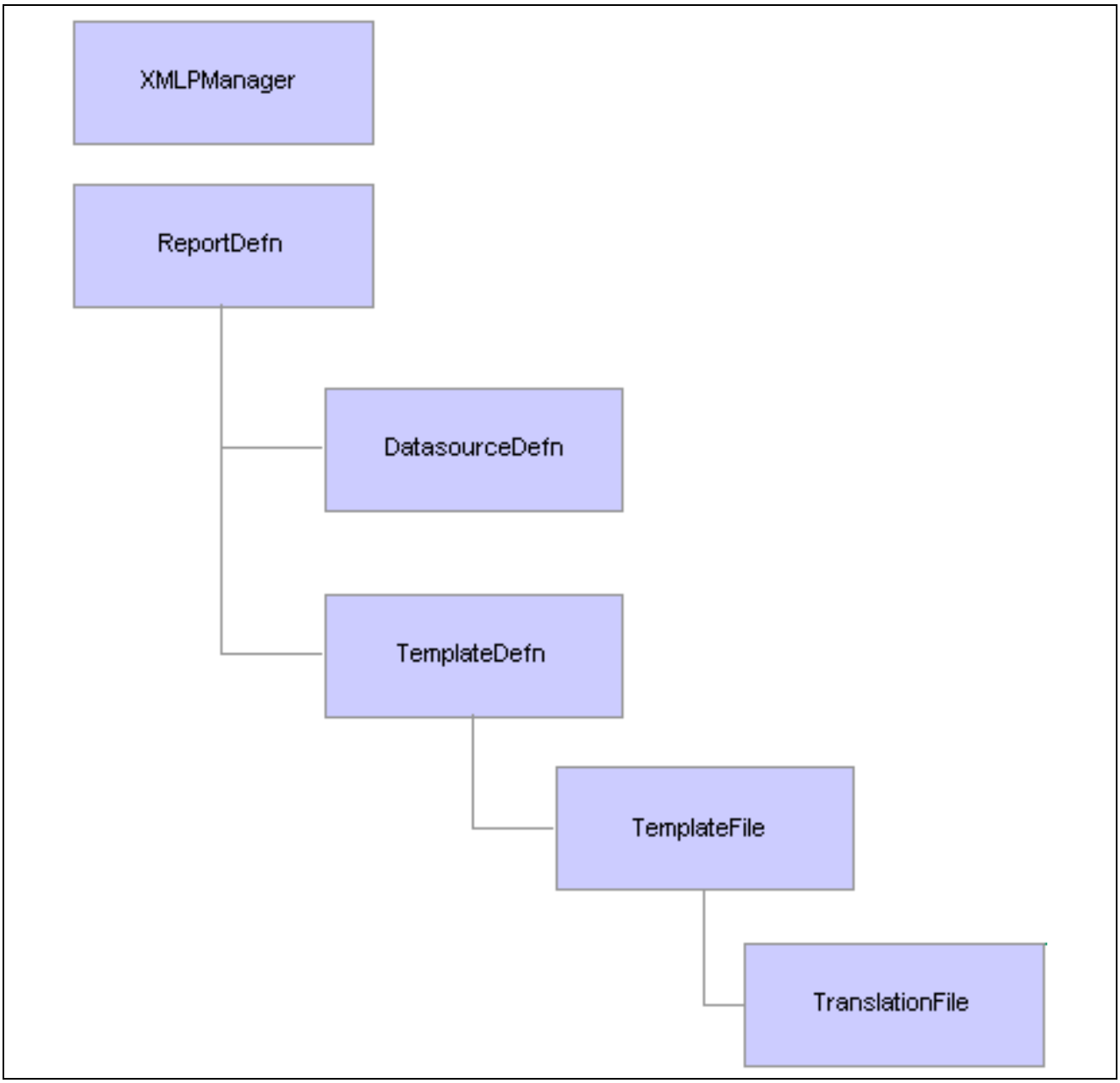
XML Publisher Classes Hierarchies

Only the report manager definition and XML Publisher engine classes have a hierarchy. The other categories of classes do not have a hierarchy, that is, you can instantiate all the objects in that class from the top level object.

Report Manager Definition Classes Hierarchy

There are many different classes used with the report manager definition classes. The following flowchart shows the different classes, and how they are interrelated.

Copyright © 1988-2007, Oracle. All rights reserved.

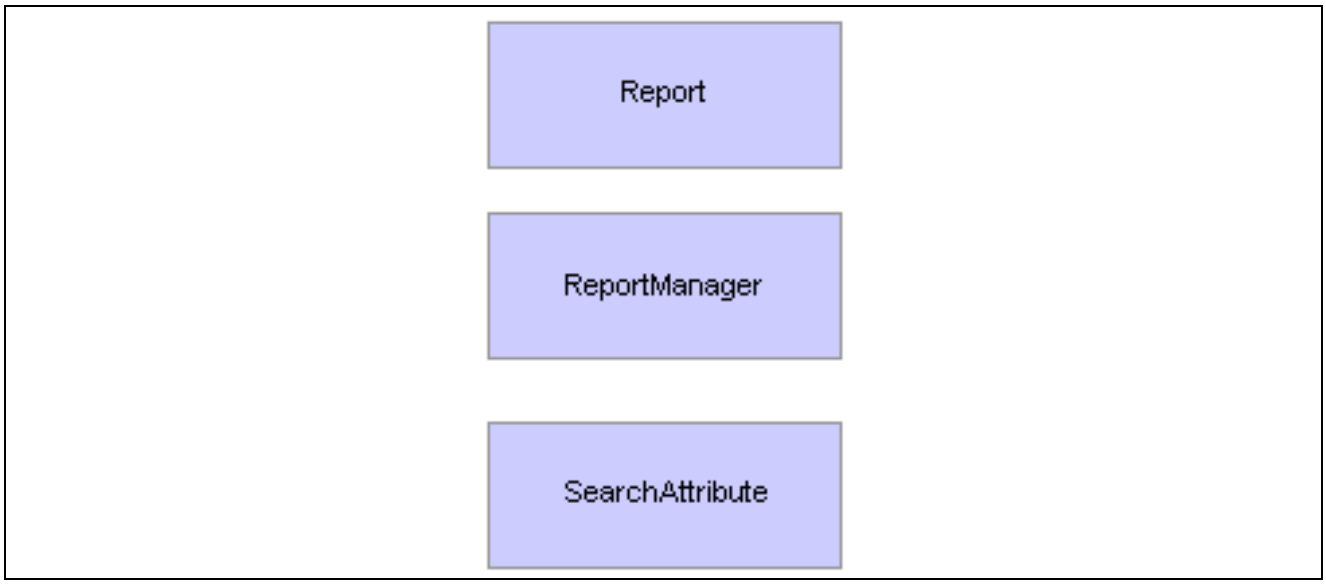


XML Publisher classes hierarchy

XML Publisher Report Search Classes Structure

The XML Publisher report search classes are composed of a few classes. All of the classes can be instantiated from a top level object. The following is the structure of the XML Publisher report search classes:

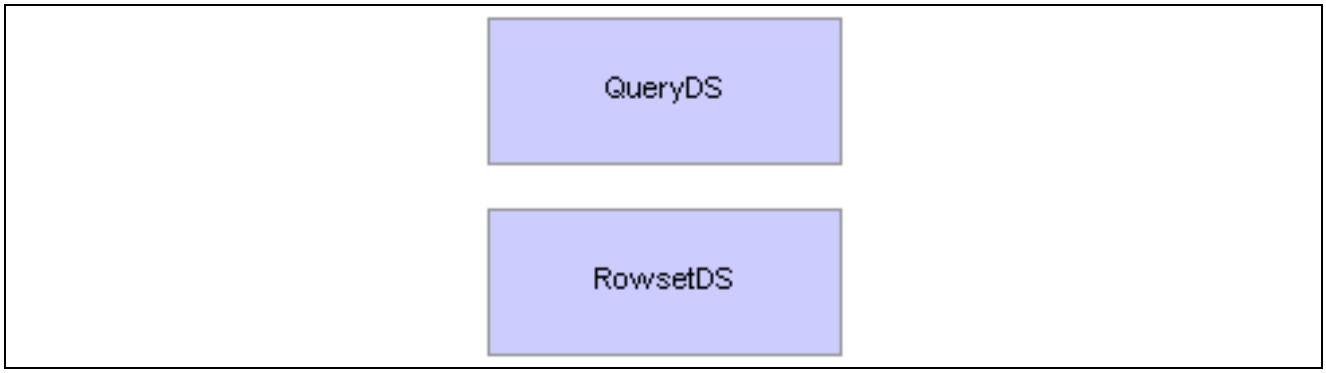
Copyright © 1988-2007, Oracle. All rights reserved.



XML Publisher report search classes structure

XML Publisher Generation Classes Structure

The XML Publisher generation classes are composed of a few classes. All of the classes can be instantiated from a top level object. The following is the structure of the XML Publisher generation classes:

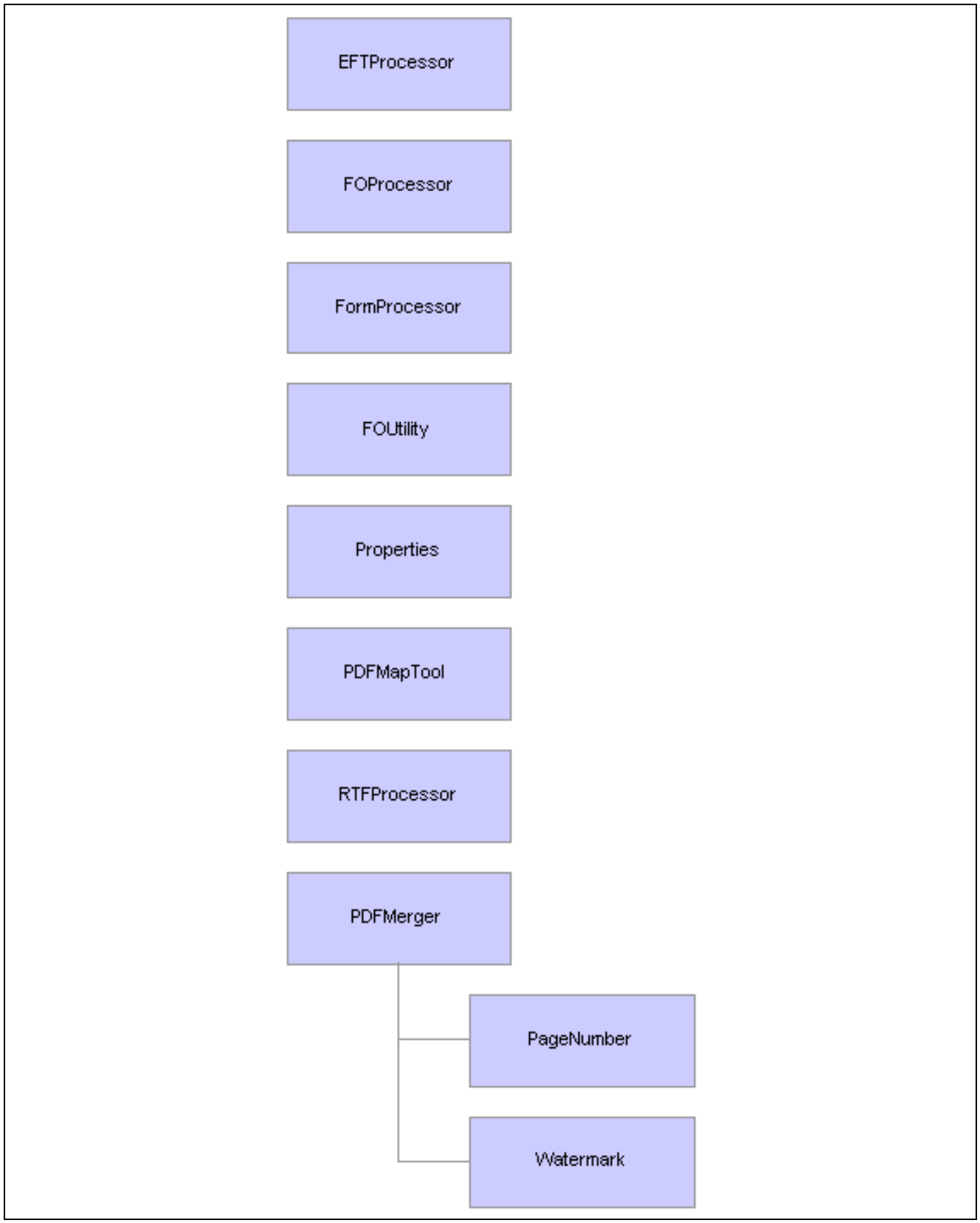


XML Publisher generation classes structure

XML Publisher Engine Classes Hierarchy

The XML Publisher engine classes are composed of many different classes. Two classes, can't be instantiated from the top-level object, but instead, are instantiated from the PDFMerger class, the PageNumber and Watermark class. The following flowchart shows the different classes, and how they are interrelated.

Copyright © 1988-2007, Oracle. All rights reserved.



XML Publisher engine classes hierarchy

Copyright © 1988-2007, Oracle. All rights reserved.

Error Handling

The different categories of classes handle errors differently.

The report manager definition and report manager search classes methods and properties should be wrapped in a try-catch statement to handle any errors.

The general classes generally return a populated object that can be checked for a null value to determine any errors.

The XML Publisher engine class methods generally contain an error parameter as part of their signature that you can check for any errors.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Built-in Functions," Try

Data Types of XML Publisher Objects

Every XML Publisher class is its own data type, that is, report definitions are declared as data type ReportDefn, template definitions are declared as type TemplateDefn, and so on.

Scope of XML Publisher Objects

These objects can only be instantiated from PeopleCode.

An XML Publisher object can be declared of type Local, Component or Global.

How to Import the XML Publisher Classes

The XML Publisher classes are not built-in classes, like rowset, field, record, and so on. They are application classes. Before you can use these classes in your PeopleCode program, you must import them to your program.

An import statement names either all the classes in a package or one particular application class. For importing the XML Publisher classes, PeopleSoft recommends that you import all the classes in the application package.

Class Category	Application Package Name
Report manager definition classes	PSXP_RPTDEFNMANAGER
Report search manager classes	PSXP_REPORTMGR

 Copyright © 1988-2007, Oracle. All rights reserved.

Class Category	Application Package Name
XML generation classes	PSXP_XMLGEN
XML Publisher engine classes	PSXP_ENGINE

You should only import the classes that you need to for your application.

Your import statements should look like the following:

```
import PSXP_RPTDEFNMANAGER:*;
import PSXP_REPORTMGR:*;
import PSXP_XMLGEN:*;
import PSXP_ENGINE:*;
```

Using the asterisks after the package name makes all the application classes directly contained in the named package available. Application classes contained in subpackages of the named package are not made available.

See Also

Chapter 6, “Application Classes,” page 153

How to Create XML Publisher Objects

After you’ve imported the appropriate XML Publisher classes, you instantiate an object of one of those classes using the constructor for the class and the Create function.

The following example creates a new instance of a ReportDefn class, as the variable &rptDefn:

```
&rptDefn = Create ReportDefn(&rptDefnId);
```

Search Operator Values

Methods in the report manager definition and report search manager classes enable you to search for existing definitions. Many of the parameters for these methods should be specified in pairs, that is, a value specified with a search operator. All the search operator parameters use the same values.

The following lists the search operator values that can be used with the XML Publisher classes:

Value	Description
%PSXP_SrchBegins	The name to search for begins with the specified value.
%PSXP_SrchContains	The name to search for contains the specified value.
%PSXP_SrchEquals	The name to search for equals the specified value.
%PSXP_SrchNotEquals	The name to search for does not equal the specified value.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
%PSXP_SrchLessThan	The name to search for is less than the specified value.
%PSXP_SrchLessEquals	The name to search for is less than or equal to the specified value.
%PSXP_SrchGreaterThan	The name to search for is greater than the specified value.
%PSXP_SrchGreaterEquals	The name to search for is greater than or equal to the specified value.
%PSXP_SrchBetween	The name to search for is between two values. Separate the values by a comma. Do not use quotation marks. For example, `ACCT1,ACCT9`.
%PSXP_SrchIn	The name to search for is in the specified list. Separate the values with commas. Do not use quotation marks. For example, `ACCT1, ACCT2, ACCT3`.

Report Manager Definition Classes Constructors

Before you can use the report manager definition classes, you must import the application package that contains them, as follows:

```
import PSXP_RPTDEFNMANAGER:*;
```

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the report manager definition classes, in alphabetical order.

DataSourceDefn

Syntax

DataSourceDefn(*DataSourceID*, *DataSourceType*, *Public*)

Description

Use the DataSourceDefn constructor to instantiate a DataSourceDefn object. You must then use the Get method to initialize the object.

Parameters

Parameter	Description
DataSourceID	Specify a unique name for the data source, as a string.
DataSourceType	Specify the type for the data source. Valid values are:

 Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
QRY	A PeopleSoft query.
RST	A PeopleSoft rowset.
XMD	An XmlDoc object.
XML	An XML file.

Parameter	Description
Public	If the data source type is a PeopleSoft query, specifying *Public* as true means that the data source is a public query. Specifying false for this parameter means the data source is a private query. For other data source types, you must specify true for this parameter. You receive an error if you specify false.

Returns

A reference to a DataSourceDefn object.

See Also

Chapter 43, "XML Publisher Classes," DataSourceDefn Class Methods, page 1813

ReportDefn

Syntax

ReportDefn(*ReportId*)

Description

Use the ReportDefn constructor to instantiate a ReportDefn object. You must then use the Create or Get method to initialize the object.

Parameters

Parameter	Description
ReportId	Specify a unique ID to be associated with the report definition.

Returns

A reference to a ReportDefn object.

See Also

Chapter 43, "XML Publisher Classes," ReportDefn Class Methods, page 1817

TemplateDefn

Syntax

TemplateDefn(*TemplateID*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the TemplateDefn constructor to instantiate a TemplateDefn object. You must then use the Get method to initialize the object.

You generally don't use the constructor to instantiate a TemplateDefn object. Instead, use the GetTemplateDefn method.

Parameters

Parameter	Description
TemplateId	Specify a unique identifier for the template, as a string.

Returns

A reference to a TemplateDefn object.

See Also

Chapter 43, "XML Publisher Classes," TemplateDefn Class Methods, page 1834
Chapter 43, "XML Publisher Classes," GetTemplate, page 1821 and Chapter 43, "XML Publisher Classes," GetDefaultTemplate, page 1819.

TemplateFile

Syntax

```
TemplateFile(&TemplateDefn, EffDate)
```

Description

Use the TemplateFile constructor to instantiate a TemplateFile object.

You generally don't use the constructor to instantiate a TemplateFile object. Instead, use the GetTemplateFile method.

Parameters

Parameter	Description
&TemplateDefn	Specify an already instantiated and populated template definition object that is associated with the template file.
EffDate	Specify the effective date of the template file that you want to access.

Returns

A reference to a TemplateFile object.

See Also

Chapter 43, "XML Publisher Classes," GetTemplateFile, page 1835 and Chapter 43, "XML Publisher Classes," GetActiveTemplateFile, page 1834.

Chapter 43, "XML Publisher Classes," TemplateFile Class Methods, page 1838

 Copyright © 1988-2007, Oracle. All rights reserved.

TranslationFile

Syntax

```
TranslationFile(&TemplateDefn, EffectiveDate, LanguageCode)
```

Description

Use the TranslationFile constructor to instantiate a TranslationFile object.

You generally don't use the constructor to instantiate a TranslationFile object. Instead, use the GetTranslationFile method to access an existing translation file.

Parameters

Parameter	Description
&TemplateDefn	Specify an already instantiated and populated template definition object, that is associated with the translation file.
EffectiveDate	Specify the effective date for the translation file, as a date.
LanguageCode	Specify the three-letter language code for the translation file, such as `ENG, FRA, ESP` and so on.

Returns

A reference to a TranslationFile object.

See Also

Chapter 43, "XML Publisher Classes," GetTranslationFile, page 1839

Chapter 43, "XML Publisher Classes," TranslationFile Class Method, page 1841

DataSourceDefn Class Methods

The following are the DataSourceDefn class methods, in alphabetical order.

Get

Syntax

```
Get()
```

Description

Use the Get method to initialize the object with the existing underlying data source definition.

Parameters

None.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

GetData

Syntax

GetData(*&PromptRecord*)

Description

Use the GetData method to return the existing data file.

This method should only be used when the type of the data source is a PeopleSoft query.

If the query requires prompts, you must specify an already instantiated and populated record object that contains the values for the query prompts. Otherwise, specify a null value (" ").

Parameters

Parameter	Description
&PromptRecord	Specify an already instantiated and populated record object that contains the prompts for the query if necessary, otherwise specify a null value.

Returns

A string containing the path to the data.

See Also

Chapter 43, "XML Publisher Classes," GetSampleData, page 1814

GetSampleData

Syntax

GetSampleData()

Description

Use the GetSampleData method to return a path to the file containing the sample data.

The file is populated when the data source is first loaded into memory. The file is then created on the application server system. The path that is returned is an absolute path name.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.

Parameters

None.

Returns

A string containing the path name to the file containing the sample data.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 43, “XML Publisher Classes,” GetData, page 1814

GetSchema

Syntax

```
GetSchema()
```

Description

Use the GetSchema method to return a path to the existing schema file. This method is only available for data source definitions that have PeopleSoft query defined as the type.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.

Parameters

None.

Returns

A string containing the path to the existing schema file.

See Also

Chapter 43, “XML Publisher Classes,” GetData, page 1814

DataSourceDefn Class Properties

The following are the DataSourceDefn class properties, in alphabetical order.

ActiveFlag

Description

This property returns the status of the data source definition. This property returns a boolean value: true if the data source definition is active, false otherwise.

This property is read-only.

Description

Description

The Description property returns the description for the data source, as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

IsPublic

Description

This property is only meaningful for data source definitions that are defined as PeopleSoft queries.

This property returns a boolean value: true if the data source is a public query, false if the data source is a private query.

For all other data source types, this property always returns a value of true.

This property is read-only.

LastUpdatedBy

Description

This property returns the user Id of the user who last updated the data source definition, as a string.

This property is read-only.

LastUpdateDTTM

Description

This property returns the date time of the last time the data source definition was updated, as a date time value.

This property is read-only.

Name

Description

This property specifies the name of the data source definition, as a string.

This property is read-only.

ObjectOwnerID

Description

This property returns the user Id of the user who currently owns the data source definition.

This property is read-only.

RegisteredBy

Description

This property returns the user Id of the user who originally registered the data source.

This property is read-only.

 Copyright © 1988-2007, Oracle. All rights reserved.

RegisteredDTTM

Description

This property returns the date time of the time when the data source was originally registered, as a date time value.

This property is read-only.

Type

Description

This property returns the type of the data source, as a string. Valid values are:

Value	Description
QRY	A PeopleSoft query.
RST	A PeopleSoft rowset.
XMD	An XmlDoc object.
XML	An XML file.

This property is read-only.

ReportDefn Class Methods

The following are the ReportDefn class methods, in alphabetical order.

DisplayOutput

Syntax

```
DisplayOutput()
```

Description

Use the DisplayOutput method to display the report generated by the ProcessReport method in a separate browser window.

You must successfully call the ProcessReport method prior to calling this method.

This method displays a single report. Therefore, the report definition must not be set for bursting.

Parameters

None.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 43, "XML Publisher Classes," ProcessReport, page 1823

Get

Syntax

```
Get()
```

Description

Use the Get method to return a reference to the existing ReportDefn object.

Parameters

None.

Returns

A reference to the newly instantiated and populated ReportDefn object.

GetActiveTemplatesByDistributionChannel

Syntax

```
GetActiveTemplatesByDistributionChannel(DistributionChannel, AsOfDate)
```

Description

Use the GetActiveTemplatesByDistributionChannel method to return an array of TemplateDefn objects for the specified distribution channel and as of date.

Parameters

Parameter	Description
DistributionChannel	Specify the distribution channel name as a string.
AsOfDate	Specify the as of date for the template definitions that you want to access, as a date.

Returns

An array of TemplateDefn objects.

See Also

Chapter 43, "XML Publisher Classes," TemplateDefn Class Methods, page 1834

GetDatasource

Syntax

```
GetDatasource()
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetDatasource method to return a reference to the data source definition object registered with the report definition.

Parameters

None.

Returns

A reference to a DatasourceDefn object.

See Also

Chapter 43, “XML Publisher Classes,” DataSourceDefn Class Methods, page 1813

GetDefaultTemplate

Syntax

```
GetDefaultTemplate()
```

Description

Use the GetDefaultTemplate method to return a reference to the default template definition registered with the report definition.

Parameters

None.

Returns

A reference to a TemplateDefn object.

See Also

Chapter 43, “XML Publisher Classes,” TemplateDefn Class Methods, page 1834

GetOutDestFormatString

Syntax

```
GetOutDestFormatString(OutDestFormat)
```

Description

Use the GetOutDestFormatString method to convert the output format returned from system variable %OutDestFormat to an XML Publisher output format string that can be passed to the output type parameter of the ProcessReport method. This method is useful when processing reports generated by application engine programs that are run process scheduler.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
OutDestFormat	Specify the output destination format as a number. If you specify something other than a valid value, the method throws an exception. Valid values are:

Value	Description
2	PDF format
5	HTM format
8	XLS format
12	RTF format

Returns

A string if successful. Valid values are:

- PDF
- HTM
- XLS
- RTF

See Also

Chapter 43, "XML Publisher Classes," ProcessReport, page 1823

GetPSQueryPromptRecord

Syntax

```
GetPSQueryPromptRecord()
```

Description

Use the GetPSQueryPromptRecord method to return the runtime prompts of a query as a record object. This method should only be used when the data source type associated with the report definition is a PeopleSoft query. This method returns a null when the data source type isn't a PeopleSoft Query.

Parameters

None.

Returns

A record object populated with the query prompts if the data source definition is a PeopleSoft query, null otherwise.

See Also

Chapter 43, "XML Publisher Classes," SetPSQueryPromptRecord, page 1825

 Copyright © 1988-2007, Oracle. All rights reserved.

GetTemplate

Syntax

```
GetTemplate(TemplateID)
```

Description

Use the GetTemplate method to access the specified template definition.

The template must have already been associated with the report definition.

Parameters

Parameter	Description
TemplateID	Specify the name of the template that you want to access, as a string.

Returns

A reference to a TemplateDefn object.

See Also

Chapter 43, “XML Publisher Classes,” TemplateDefn Class Methods, page 1834

GetTemplateList

Syntax

```
GetTemplateList()
```

Description

Use the GetTemplateList method to return a list of templates registered with the report definition.

Parameters

None.

Returns

An array of TemplateDefn objects.

See Also

Chapter 43, “XML Publisher Classes,” TemplateDefn Class Methods, page 1834

Chapter 8, “Array Class,” page 215

PrintOutput

Syntax

```
PrintOutput(DestPrinter)
```

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the PrintOutput method to print the report definition object executing the method. The ProcessReport method must be called successfully before you use this method.

Parameters

Parameter	Description
DestPrinter	Specify the printer you want to use for printing the report, as a string.

Returns

None.

ProcessEtextReport

Syntax

ProcessEtextReport(*TemplateID, LanguageCd, AsOfDate*)

Description

Use the ProcessEtextReport when you want to process a report that has an etext template and the report output is expected to be a string read by PeopleCode instead of a file.

For any other type of report processing, use the ProcessReport method.

Note. ProcessEtextReport does not support following: bursting, dynamic template selection, and report definitions with template type other than etext.

Before you generate the report, if you are using a data source other than a PeopleSoft query, you must specify that data source using one of the set runtime data methods, such as SetRuntimeDataRowset or SetRuntimeDataXMLFile.

Parameters

Parameter	Description
TemplateId	Specify the template to be used for processing this report, as a string. You can specify a null value for this parameter, that is, (""). When you specify a null value, the default template associated with the report definition is used. **Note.** The template must be an etext template.
LanguageCD	Specify the three-letter language code for the report definition, such as `ENG, FRA, ESP` and so on. This parameter controls both the translation and the language specific output details such as font selection and right-to-left page orientation for Arabic/Hebrew. You can specify a null value for this parameter, that is, (""). If you specify a null value, the current language of the session is used.
AsOfDate	Specify the as of date for the report, as a date.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A string.

Example

```
/* XML Publisher sample report using XML Doc data source */

Local PSXP_RPTDEFNMANAGER:ReportDefn &oRptDefn;

/* create report definition */

&oRptDefn = create PSXP_RPTDEFNMANAGER:ReportDefn("MYREPORT");

&oRptDefn.Get();

Local XmlDoc &inXMLDoc;

Local string &strout;

&inXMLDoc = CreateXmlDoc("");

&ret = &inXMLDoc.ParseXmlFromURL("D:\temp\etext\TestEft.xml");

/*attach XMLDoc Object to Report Definition;*/

&oRptDefn.SetRuntimeDataXMLDoc(&inXMLDoc);

/* process etext report */

&strout = &oRptDefn.ProcessETextReport("", MYREPORT_AET.LANGUAGE_CD, %Date);

WriteToLog(%ApplicationLogFence_Level1, "string output from processEtextReport API⇒
 " | &strout);
```

See Also

Chapter 43, “XML Publisher Classes,” ProcessReport, page 1823

ProcessReport

Syntax

ProcessReport(*TemplateId*, *LanguageCD*, *AsOfDate*, *OutputFormat*)

Description

Use the ProcessReport method to generate a report and store the information.

Copyright © 1988-2007, Oracle. All rights reserved.

Before you generate the report, if you are using a data source other than a PeopleSoft query, you must specify that data source using one of the set runtime data methods, such as SetRuntimeDataRowset or SetRuntimeDataXMLFile.

After you use the ProcessReport method, you can use the Publish method to post the report, the DisplayOutput method to display the report in a browser window, or the PrintOutput method to print the report.

Use the ProcessEtextReport instead when you want to process a report that has an etext template and the report output is expected to be a string read by PeopleCode instead of a file.

Parameters

Parameter	Description
TemplateId	Specify the template to be used for processing this report, as a string. You can specify a null value for this parameter, that is, (""). When you specify a null value, the default template associated with the report definition is used.
LanguageCD	Specify the three-letter language code for the report definition, such as `ENG, FRA, ESP` and so on. This parameter controls both the translation and the language specific output details such as font selection and right-to-left page orientation for Arabic/Hebrew. You can specify a null value for this parameter, that is, (""). If you specify a null value, the current language of the session is used.
AsOfDate	Specify the as of date for the report, as a date.
OutputFormat	Specify the output format, as a string. See Chapter 43, "XML Publisher Classes," GetOutDestFormatString, page 1819. You can specify a null value for this parameter, that is, (""). When you specify a null value, the default output format associated with the report definition is used.

Returns

None.

See Also

Chapter 43, "XML Publisher Classes," DisplayOutput, page 1817 , Chapter 43, "XML Publisher Classes," PrintOutput, page 1821 , Chapter 43, "XML Publisher Classes," SetPSQueryPromptRecord, page 1825 , Chapter 43, "XML Publisher Classes," SetRuntimeDataRowset, page 1826 , Chapter 43, "XML Publisher Classes," SetRuntimeDataXMLDoc, page 1826 , Chapter 43, "XML Publisher Classes," SetRuntimeDataXMLFile, page 1827 and Chapter 43, "XML Publisher Classes," SetRuntimeProperties, page 1828.

Chapter 43, "XML Publisher Classes," ProcessEtextReport, page 1822

Publish

Syntax

```
Publish(ServerName, ReportPath, FolderName, ProcessInstanceId)
```

Description

Use the Publish method to publish the current report definition

 Copyright © 1988-2007, Oracle. All rights reserved.

You must have successfully completed a call to the ProcessReport method before you can use this method, either directly or by using one of the XML Publisher XML Publisher engine classes.

Parameters

Parameter	Description
ServerName	Specify the server to use for publishing the report, as a string. You can specify a null value for this parameter, that is, ("").
ReportPath	Specify an absolute path to the report, as a string. You can specify a null value for this parameter, that is, (""). This parameter should only be used for publishing reports that are not generated by directly using the ProcessReport method. Forward and backward slashes should be coded according to the operating system of the application server that contains the report.
FolderName	Specify the name of the report folder that the report should be posted to. You can specify a null value for this parameter, that is, (""). If you do not provide a value for this parameter, the default folder is used to post reports.
ProcessInstanceId	Specify the process instance of the calling process, such as the calling application engine program. If you specify a 0, a new process instance is generated to track the report posting process.

Returns

None.

See Also

Chapter 43, "XML Publisher Classes," ProcessReport, page 1823

SetPSQueryPromptRecord

Syntax

SetPSQueryPromptRecord(*&Record*)

Description

Use the SetPSQueryPromptRecord method to specify an already instantiated record object that contains the prompt values for the query to be used to populate the report.

This method can only be used with reports that have a PeopleSoft query defined as the data source.

You must use this method before using the ProcessReport method.

Parameters

Parameter	Description
&Record	Specify an already instantiated and populated record object that contains the values for the query to be used to populate the report.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

See Also

Chapter 43, "XML Publisher Classes," ProcessReport, page 1823 and Chapter 43, "XML Publisher Classes," GetPSQueryPromptRecord, page 1820.

Chapter 30, "Query Classes," page 1233

SetRuntimeDataRowset

Syntax

SetRuntimeDataRowset(*&Rowset*)

Description

Use the SetRuntimeDataRowset method to specify an existing rowset object as the data source for the report.

Report definitions that use a data source other than a PeopleSoft query must set that data source before generating the report using the ProcessReport method.

Parameters

Parameter	Description
&Rowset	Specify an already instantiated and populated rowset object to be used as the data source for the report.

Returns

None.

See Also

Chapter 43, "XML Publisher Classes," GetDatasource, page 1818 , Chapter 43, "XML Publisher Classes," ProcessReport, page 1823 , Chapter 43, "XML Publisher Classes," SetRuntimeDataXMLDoc, page 1826 and Chapter 43, "XML Publisher Classes," SetRuntimeDataXMLFile, page 1827.

Chapter 33, "Rowset Class," page 1421

SetRuntimeDataXMLDoc

Syntax

SetRuntimeDataXMLDoc(*&XmlDoc*)

Description

Use the SetRuntimeDataXMLDoc method to specify an existing XmlDoc object as the data source for the report.

Report definitions that use a data source other than a PeopleSoft query must set that data source before generating the report using the ProcessReport method.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&XmlDoc	Specify an already instantiated and populated XmlDoc object to be used as the data source for the report.

Returns

None.

See Also

Chapter 43, "XML Publisher Classes," GetDatasource, page 1818 , Chapter 43, "XML Publisher Classes," ProcessReport, page 1823 , Chapter 43, "XML Publisher Classes," SetRuntimeDataRowset, page 1826 and Chapter 43, "XML Publisher Classes," SetRuntimeDataXMLFile, page 1827.

Chapter 42, "XmlDoc Class," page 1745

SetRuntimeDataXMLFile

Syntax

```
SetRuntimeDataXMLFile(FilePath)
```

Description

Use the SetRuntimeDataXMLFile method to specify an existing file as the data source for the report.

Report definitions that use a data source other than a PeopleSoft query must set that data source before generating the report using the ProcessReport method.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server use a path with back slashes.

Parameters

Parameter	Description
FilePath	Specify an absolute path to the file that you want to use. You must include the file name and file extension in the parameter. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server use a path with back slashes.

Returns

None.

See Also

Chapter 43, "XML Publisher Classes," GetDatasource, page 1818 , Chapter 43, "XML Publisher Classes," ProcessReport, page 1823 , Chapter 43, "XML Publisher Classes," SetRuntimeDataRowset, page 1826 and Chapter 43, "XML Publisher Classes," SetRuntimeDataXMLDoc, page 1826.

Chapter 16, "File Class," page 523

Copyright © 1988-2007, Oracle. All rights reserved.

SetRuntimeProperties

Syntax

SetRuntimeProperties(*&NameArray, &ValueArray*)

Description

Use the SetRuntimeProperties method to set additional runtime values and properties required for generating this report.

Parameters

Parameter	Description
&NameArray	Specify an already instantiated and populated array of string containing the names of the additional properties or variables that you need for this report.
&ValueArray	Specify an already instantiated and populated array of string containing the values of the additional properties or variables that you need for this report. The values should be in the same order as the names listed in *&NameArray*.

Returns

None.

Example

The following example sets extra parameters for securing a document.

```
&asPropName = CreateArrayRept("", 0);
&asPropValue = CreateArrayRept("", 0);
&asPropName.Push("pdf-security");
&asPropValue.Push("true");
&asPropName.Push("pdf-open-password");
&asPropValue.Push("user1");
&asPropName.Push("pdf-permissions-password");
&asPropValue.Push("owner1");
&oRptDefn.SetRuntimeProperties(&asPropName, &asPropValue);
```

See Also

Chapter 43, “XML Publisher Classes,” ProcessReport, page 1823

ReportDefn Class Properties

The following are the ReportDefn class properties, in alphabetical order.

 Copyright © 1988-2007, Oracle. All rights reserved.

AllowRecipientEntry

Description

Use this property to specify whether the requester can add viewers to the process scheduler distribution page when running the report. This property takes a boolean value, true if requestor can add viewers, false otherwise.

This property is read-write.

AllowViewerEntry

Description

Use the AllowViewerEntry property to specify whether the requestor can add email instructions to the process scheduler distribution page. This property takes a boolean value; true if the requestor can add the instructions, false otherwise.

This property is read-write.

ArchiveAfter

Description

Use this property to specify the number of days that the report is retained in the report repository.

This property is read-write.

BurstFieldName

Description

Use the BurstFieldName to specify the name of the field used for bursting the report when it is generated.

This property is required if you are going to burst a report. If this property has no value, the report will *not* be burst when it is generated.

If you are bursting a report, you also may also want to set the following properties:

- SecurityFieldName
- SecurityIDType
- SecurityJoinTable
- TemplateControlFieldName
- UseBurstValueAsOutputValueName

This property is read-write.

Considerations Using Bursting with File Layouts

In order for bursting to work with a file layout, you must place the burst field at the highest repeating level in the file structure. You must also do the following:

1. Open the file in write mode.
2. Write the following code to the file:

Copyright © 1988-2007, Oracle. All rights reserved.

```
&file.writeline("<?xml version='1.0'?>");
```

3. Close the file and open it again in append mode while processing the file layout. This ensures that:
   - The start tags (<start> </start>) are not inserted at the beginning and end of the file.
   - The first line in the file is "<?xml version='1.0'?>".

See Also

Chapter 43, "XML Publisher Classes," SecurityFieldName, page 1832 , Chapter 43, "XML Publisher Classes," SecurityIDType, page 1832 , Chapter 43, "XML Publisher Classes," SecurityJoinTable, page 1833 and Chapter 43, "XML Publisher Classes," UseBurstValueAsOutputFileName, page 1834.

CategoryId

Description

Use the CategoryId property to specify the report category ID for the report definition.

This property is read-write.

Description

Description

Use the Description property to specify a description for the report definition.

This property is read-write.

DestinationPrinter

Description

Use the DestinationPrinter property to specify a printer for the generated report.

FolderName

Description

Use the FolderName property to specify a folder name for the generated report to be posted to.

This property is read-write.

ID

Description

Use the ID property to specify the ID for the report.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

IsReadOnly

Description

This property returns a boolean value: true if the report definition is read-only to the current end user, false otherwise.

This property is read-only.

LastUpdateDTTM

Description

This property returns the last time that this report definition was updated, as a datetime.

This property is read-only.

LastUpdateOprID

Description

This property returns the last user Id of the user who updated this report definition.

This property is read-only.

ObjectOwnerID

Description

This property returns the user Id of the owner of the data source associated with the report definition.

This property is read-only.

OutDestination

Description

This property returns the output destination of the report definition, as a string.

This property is read-write.

OutDestinationType

Description

This property returns the type of device specified for the output of the report definition, as a string.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

OutputEditable

Description

This property specifies whether or not the internal Adobe flag of a PDF report output file has the setting turned on to allow editing.

Note. This property should only be specified when the template type is RTF or XSL, and when generating PDF output.

This property is read-write.

RegisteredBy

Description

This property returns the name of user who originally created the report definition, as a string.

This property is read-only.

RegisteredDTTM

Description

This property returns the date time when the report definition was first created, as a date time value.

This property is read-only.

SecurityFieldName

Description

Use this property to specify the field name from the Security Join Table that stores Viewer ID. This is used with bursting.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," BurstFieldName, page 1829 , Chapter 43, "XML Publisher Classes," SecurityIDType, page 1832 and Chapter 43, "XML Publisher Classes," SecurityJoinTable, page 1833.

SecurityIDType

Description

Use this property to specify the type of ID that the Security Viewer ID is; either a user Id or a permission list. This is used with bursting. Valid values are:

Value	Description
2	User Id
3	Permission list role

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," BurstFieldName, page 1829 , Chapter 43, "XML Publisher Classes," SecurityFieldName, page 1832 and Chapter 43, "XML Publisher Classes," SecurityJoinTable, page 1833.

SecurityJoinTable

Description

Use this property to specify the name of a record that stores either a user ID or permission list assigned to a data value found in the XML data for this report definition. This prompt list is filtered for records that meet this criteria. This is used for bursting reports.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," BurstFieldName, page 1829 , Chapter 43, "XML Publisher Classes," SecurityFieldName, page 1832 and Chapter 43, "XML Publisher Classes," SecurityIDType, page 1832.

Status

Description

This property returns the status of the report definition. This property returns a string. Valid values are:

Value	Description
A	The report definition is active and can be run.
I	The report definition is inactive. It cannot be run and it isn't translatable.
P	The report definition is in progress. This is the default value. The report definition cannot be run, but it is translatable.

This property is read-only.

TemplateControlFieldName

Description

Use this property to specify the name of the template control field, as a string. This field is the schema tag value from first child level of the report definition.

This property is used with bursting.

This property is read-only.

See Also

Chapter 43, "XML Publisher Classes," BurstFieldName, page 1829

Copyright © 1988-2007, Oracle. All rights reserved.

UseBurstValueAsOutputFileName

Description

Use the UseBurstValueAsOutputFileName property to specify different names for all the files that result from bursting. This property takes a boolean value: true, the report output files are named *BurstValue.Extention.* False is the default value.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," BurstFieldName, page 1829

TemplateDefn Class Methods

The following are the TemplateDefn class methods, in alphabetical order.

Get

Syntax

```
Get()
```

Description

Use the Get method to populate the current TemplateDefn object with an existing template definition.

Parameters

None.

Returns

None.

GetActiveTemplateFile

Syntax

```
GetActiveTemplateFile(AsOfDate)
```

Description

Use the GetActiveTemplateFile method to return a reference to the active template file specified by the as of date. If you want to access inactive template files as well, use the GetTemplateFile method instead.

Parameters

Parameter	Description
AsOfDate	Specify the as of date that you want to use for the active template file, as a date.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A reference to a TemplateFile object.

See Also

Chapter 43, “XML Publisher Classes,” GetTemplate, page 1821

Chapter 43, “XML Publisher Classes,” TemplateFile Class Methods, page 1838

GetTemplateFile

Syntax

```
GetTemplateFile()
```

Description

Use the GetTemplateFile method to return a reference to the associated template file object.

Parameters

None.

Returns

A TemplateFile object.

See Also

Chapter 43, “XML Publisher Classes,” GetActiveTemplatesByDistributionChannel, page 1818 and Chapter 43, “XML Publisher Classes,” GetDefaultTemplate, page 1819.

Chapter 43, “XML Publisher Classes,” TemplateFile Class Methods, page 1838

GetTemplateFileList

Syntax

```
GetTemplateFileList()
```

Description

Use the GetTemplateFileList method to return a list of all the template files associated with the template definition.

Parameters

None.

Returns

An array of TemplateFile objects.

See Also

Chapter 43, “XML Publisher Classes,” TemplateFile Class Methods, page 1838

Copyright © 1988-2007, Oracle. All rights reserved.

TemplateDefn Class Properties

The following are the TemplateDefn properties, in alphabetical order.

Description

Description

This property returns the description of the template definition, as a string.

This property is read-write.

DistributionChannel

Description

This property returns the name of the distribution channel associated with the template definition, as a string. Valid values are:

Value	Description
0	Web posting
1	EFT format
2	Printer
3	Email
4	Fax
5	SMS

This property is read-write.

ID

Description

This property returns the Id of the template definition, as a string.

This property is read-only.

IsReadOnly

Description

This property returns a boolean value; true if the template definition object is read-only to the current user, false otherwise.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

IsSubTemplate

Description

This property returns a boolean value: true if the template definition is a sub-template, false otherwise.

This property is read-only.

LanguageCode

Description

This property returns the language code associated with the template definition, as a string.

This property is read-only.

LastUpdateDTTM

Description

This property returns the date time of the last time the template definition was updated, as a date time value.

This property is read-only.

LastUpdateOprID

Description

This property returns the last user Id of the user who updated this template definition.

This property is read-only.

ObjectOwnerID

Description

This property returns the user Id of the owner of the data source associated with the template definition.

This property is read-only.

RegisteredBy

Description

This property returns the user Id of the user who originally registered the template definition.

This property is read-only.

RegisteredDTTM

Description

This property returns the date time of the time when the template definition was originally registered, as a date time value.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

ReportCategoryID

Description

This property returns the report category Id associated with the template definition.

This property is applicable only if the template definition is a sub-template.

This property is read-only.

Type

Description

This property returns the type of template definition, as a string. Valid values are:

- ETX (Etext)
- PDF
- RTF
- XLS

This property is read-only.

TemplateFile Class Methods

The following are the TemplateFile class methods, in alphabetical order.

GetFile

Syntax

```
GetFile()
```

Description

Use the GetFile method to return a string that contains the absolute path to the file.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server use a path with back slashes.

Parameters

None.

Returns

A string.

 Copyright © 1988-2007, Oracle. All rights reserved.

GetMapFile

Syntax

GetMapFile()

Description

Use the GetMapFile method to return the template map file. This method returns a string containing the absolute path to the file.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server use a path with back slashes.

Note. Only use this method when the template type is PDF. Map files are only used with PDF template processing.

Parameters

None.

Returns

A string containing the path name to the file.

GetTranslationFile

Syntax

GetTranslationFile(*LanguageCD*)

Description

Use the GetTranslationFile method to return a reference to the specified translation file.

Parameters

Parameter	Description
LanguageCD	Specify the three-letter language code for the report definition, such as ENG, FRA, ESP and so on. This parameter controls both the translation and the language specific output details such as font selection and right-to-left page orientation for Arabic/Hebrew. You can specify a null value for this parameter, that is, (""). If you specify a null value, the current language of the session is used.

Returns

A reference to a TranslationFile object.

See Also

Chapter 43, "XML Publisher Classes," TranslationFile Class Method, page 1841

Copyright © 1988-2007, Oracle. All rights reserved.

GetTranslationFileList

Syntax

```
GetTranslationFileList()
```

Description

Use the GetTranslationFileList method to return all the translation files associated with the template file.

Parameters

None.

Returns

An array of TranslationFile objects.

See Also

Chapter 43, "XML Publisher Classes," TranslationFile Class Method, page 1841

TemplateFile Class Properties

The following describes the TemplateFile class properties, in alphabetical order.

EffectiveDate

Description

This property returns the effective date associated with the template file, as a date value.

This property is read-only.

FileName

Description

This property returns the name of the template file. The file must already exist on the application server. If you want to create the file and then return a path to it, use the GetFile method instead.

This property is read-only.

See Also

Chapter 43, "XML Publisher Classes," GetFile, page 1838

MapFileName

Description

This property returns just the name of the map file associated with the template definition, as a string. This property is only used when the template type is PDF.

 Copyright © 1988-2007, Oracle. All rights reserved.

If you want to create the file and then return a path to it, use the GetMapFile method instead.

This property is read-only.

See Also

Chapter 43, "XML Publisher Classes," GetMapFile, page 1839

Status

Description

This property returns the status of the template file. Valid values are:

Value	Description
A	The template file is active.
I	The template file is inactive.
P	The template file is in progress. This is the default value.

This property is read-only.

TranslationFile Class Method

The following is the TranslationFile class method.

GetFile

Syntax

```
GetFile()
```

Description

Use the GetFile method to generate the file, as well as return a the path to the actual translation file.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server use a path with back slashes.

Parameters

None.

Returns

A string containing the absolute path to the file.

See Also

Chapter 43, "XML Publisher Classes," FileName, page 1840

Copyright © 1988-2007, Oracle. All rights reserved.

TranslationFile Class Properties

The following are the TranslationFile class properties, in alphabetical order.

Description

Description

This property returns the description of a translation file.

This property is read-only.

FileName

Description

This property specifies the file name of a translation file, as a string. This is just the file name, not the path to the file. This is the name that is displayed to the user.

This property is read-only.

LanguageCode

Description

Use this property to specify the language code of the translation file. This is the translated language, not the base language.

This property is read-write.

Status

Description

This property specifies the status of the translation file. Valid values are:

Value	Description
A	The translation file is active.
I	The translation file is inactive.
P	The translation file is in process. This is the default value.

This property is read-only.

XMLPManager Class Methods

The following are the XMPLManager class methods, in alphabetical order.

Copyright © 1988-2007, Oracle. All rights reserved.

GetDatasourceDefnList

Syntax

GetDatasourceDefnList(*DataSourceId*, *DataSourceIdOp*, *DataSourceType*, *Owner*, *Descr*, *DescrOp*, *ObjectOwnerId*, *Active*, *CaseSensitive*)

Description

Use the GetDatasourceDefnList method to return an array of DataSourceDefn objects.

Parameters

Parameter	Description
DataSourceId	Specify a data source ID associated with a data source object that you want to access, as a string.
DataSourceIdOp	Specify the search operator to be used with the value specified in *DataSourceIdOp*, as a number. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
DataSourceType	Specify the data source for a data source object that you want to access. Valid values are:

Value	Description
QRY	PeopleSoft query
RST	Rowset
XMD	XmlDoc object
XML	XML file

Parameter	Description
Owner	Specify whether to search based on owner.

Value	Description
0	Do not search based on owner.
1	Search public query based data source, and other data source types.
2	Search private queries only. **Note.** If the data source type is not a PeopleSoft query, and the owner is 2 (private queries only,) result is always empty.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameter	Description
Descr	Specify the description of a data source that you'd like to access, as a string.
DescrOp	Specify the search operator to be used with the value specified in *Descr*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
ObjectOwnerId	Specify an object owner ID for a data source that you'd like to access, as a string. (Application owner?)
Active	Specify the status of the data source to be searched, whether active or inactive. Valid values are:

Value	Description
I	Search only inactive data sources.
A	Search only active data sources.

Parameter	Description
CaseSensitive	Specify whether the description specified by *Descr* is case sensitive. This parameter takes a Boolean value: True, the description is case sensitive, false otherwise. The default value is false.

Returns

An array of DataSourceDefn objects.

See Also

Chapter 43, "XML Publisher Classes," DataSourceDefn Class Methods, page 1813

Chapter 8, "Array Class," page 215

GetReportDefnList

Syntax

```
GetReportDefnList(ReportID, ReportIdOp, Descr, DescrOp , TemplateId, TemplateIdOp,
TemplateDescr, TemplateDescrOp, DatasourceType, DataSourceId, DataSourceIdOp,
DataSourceOwner, TemplateType, ReportCategoryId, ReportStatus, CaseSensitive)
```

Description

Use the GetReportDefnList method to return a list of all the report definitions in the system.

This method doesn't return any report definition based on a PeopleSoft query that the current user does not have access to, or any report that the current user doesn't have access to based on the report category.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
ReportID	Specify the report ID of a report that you'd like to search for, as a string.
ReportIDOp	Specify the search operator to be used with the value specified in *ReportID*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
Descr	Specify the description of a report that you'd like to access, as a string.
DescrOp	Specify the search operator to be used with the value specified in *Descr*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
TemplateId	Specify the template ID associated with a report that you want to access, as a string.
TemplateIdOp	Specify the search operator to be used with the value specified in *TemplateId*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
TemplateDescr	Specify the description of the template associated with a report you want to access, as a string.
TemplateDescrOp	Specify the search operator to be used with the value specified in *TemplateDescrOp*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
DataSourceType	Specify the data source for a report that you want to access. Valid values are:

Value	Description
QRY	A PeopleSoft query.
RST	A PeopleSoft rowset.
XMD	An XmlDoc object.
XML	An XML file.

Parameter	Description
DataSourceId	Specify a data source ID associated with a report you want to access.
DataSourceIdOp	Specify the search operator to be used with the value specified in *DataSourceId*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
DataSourceOwner	Specify whether to search based on owner.

Value	Description
0	Do not search based on owner
1	Search public queries only.

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
2	Search private queries only. If *DataSourceType* <> QRY and Owner = 2, result is always an empty array.

Parameter	Description
TemplateType	Specify a template type associated with a report that you want to access, as a string. Valid values are:

Value	Description
RTF	Specifies a template type of rich text format.
PDF	Specifies a template type of published document format.
HTM	Specifies a template type hypertext markup language.
ETX	Specifies a template type of text.

Parameter	Description
ReportCategoryId	Specify a report category ID associated with a report that you want to access.
ReportStatus	Specify a report status associated with a report that you want to access. Valid values are: • A — Active • I — Inactive • P — In progress
CaseSensitive	Specify whether the description specified by *Descr* or *TemplateDescr* is case sensitive. This parameter takes a Boolean value: True, the description is case sensitive, false otherwise. The default value is false.

Returns

An array of ReportDefn objects.

See Also

Chapter 43, "XML Publisher Classes," ReportDefn Class Methods, page 1817

Chapter 8, "Array Class," page 215

GetTemplateDefnList

Syntax

```
GetTemplateDefnList(TemplateId, TemplateIdOp, Descr, DescrOp, TemplateType,
DistChannel, ReportCategoryId, IsSubtemplate, CaseSensitive)
```

Description

Use the GetTemplateDefnList method to return a list of template definition objects.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
TemplateId	Specify the template ID associated with a template definition that you want to access, as a string.
TemplateIdOp	Specify the search operator to be used with the value specified in *TemplateId*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
Descr	Specify the description of a template definition that you'd like to access, as a string.
DescrOp	Specify the search operator to be used with the value specified in *Descr*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
TemplateType	Specify a template type associated with a template definition that you want to access, as a string. Valid values are:

Value	Description
RTF	Specifies a template type of rich text format.
PDF	Specifies a template type of published document format.
HTM	Specifies a template type hypertext markup language.
ETX	Specifies a template type of text.

Parameter	Description
DistChannel	Specify the name of a distribution channel associated with a template definition, as a string. Valid values are:

Value	Description
0	Web posting
1	EFT format
2	Printer
3	Email
4	Fax
5	SMS

Parameter	Description
ReportCategoryId	Specify a report category ID associated with a template definition, as a string.
IsSubTemplate	Specify whether the template is a sub-template. This parameter takes a numeric value:

Copyright © 1988-2007, Oracle. All rights reserved.

Value	Description
0	The template is not a sub-template (false).
1	The template is a sub-template (true).

Parameter	Description
CaseSensitive	Specify whether the description specified by *Descr* is case sensitive. This parameter takes a Boolean value: True, the description is case sensitive, false otherwise. The default value is false.

Returns

An array of TemplateDefn objects.

See Also

Chapter 43, "XML Publisher Classes," TemplateDefn Class Methods, page 1834

Report Manager Search Classes Constructors

Before you can use the report manager search classes, you must import the application package that contains them, as follows:

```
import PSXP_REPORTMGR:*;
```

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the report manager search classes, in alphabetical order.

Report

Syntax

Report(*RptId*, *Prcsinstance*, *Contentid*, *Dbname*, *RptName*, *RptDescr*, *RptURL*, *RptCreateDttm*, *RptExpireDt*, *FldrName*)

Description

Use the report constructor to instantiate and populate a report object with the data of an already published report.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RptId	Specify the report Id of the report you want to access, as a string.
Prcsinstance	Specify the process instance of the report you want to access, as a number.
Contentid	Specify the content Id number of the report you want to access, as a number.
Dbname	Specify the data base name, as a string.
RptName	Specify the name of the report you want to access, as a string.
RptDescr	Specify the description of the report you want to access, as a string.
RptURL	Specify the URL to the report details page, as a string.
RptCreateDttm	Specify the date and time of the report you want to access, as a date time value.
RptExpireDt	Specify the date when the report will be archived, and no longer considered active, as a date.
FldName	Specify the name of the folder the report was published to, as a string. You must specify an absolute path name, using forward and back slashes as required by the operating system of the application server or process scheduler.

Returns

A reference to a report object.

See Also

Chapter 43, “XML Publisher Classes,” Report Class Properties, page 1850

ReportManager

Syntax

```
ReportManager()
```

Description

Use the ReportManager constructor to instantiate an instance of the ReportManager class.

Parameters

None.

Returns

A ReportManager object.

See Also

Chapter 43, “XML Publisher Classes,” ReportManager Class Methods, page 1853

Copyright © 1988-2007, Oracle. All rights reserved.

SearchAttribute

Syntax

SearchAttribute(*AttrName*, *AttrValue*, *CompareOperator*)

Description

Use the SearchAttribute constructor to instantiate an instance of a SearchAttribute object.

Parameters

Parameter	Description
AttrName	Specify the attribute name, as a string.
AttrValue	Specify the attribute value, as a string.
CompareOperator	Specify a comparison operator. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.

Returns

A SearchAttribute object.

See Also

Chapter 43, "XML Publisher Classes," SearchAttribute Class Properties, page 1857

Report Class

The report class is one of the report manager search classes. Use the report class to access individual reports that have already been published to the report repository.

Report Class Properties

The following are the report class properties, in alphabetical order.

contentId

Description

This property returns the content Id of the report, as a number.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

CreatedDateTime

Description

This property returns the date time when the report was created, as a date time.

This property is read-only.

DatabaseName

Description

This property returns the data base name associated with the report, as a string.

This property is read-only.

Description

Description

This property returns the description of the report definition, as a string.

This property is read-only.

ExpireDate

Description

This property returns the date when the report will be archived, as a date value.

This property is read-only.

FileURL

Description

This property returns a URL to the actual report file in the report repository.

Use the ReportURL property to access the report detail page instead.

This property is read-only.

See Also

Chapter 43, “XML Publisher Classes,” ReportURL, page 1852

FolderName

Description

This property returns the full path name of the folder where the report is stored, as a string.

This property is read-only.

Copyright © 1988-2007, Oracle. All rights reserved.

ProcessInstanceID

Description

This property returns the process instance Id associated with the report when it was posted, as a number.

This property is read-only.

See Also

Chapter 43, "XML Publisher Classes," ReportInstanceID, page 1852

ReportInstanceID

Description

This property returns the report instance Id, that was populated when the report was run, as a number.

This property is read-only.

See Also

Chapter 43, "XML Publisher Classes," ProcessInstanceID, page 1852

ReportName

Description

This property returns the name of the report, as a string.

This property is read-only.

ReportURL

Description

This property returns a URL as a string, that points to the report detail page.

Use the FileURL to access a URL to the actual file containing the report in the report repository.

This property is read-only.

See Also

Chapter 43, "XML Publisher Classes," FileURL, page 1851

ReportManager Class

The ReportManager class is one of the report manager search classes. Use the ReportManager class to access set the search criteria for reports that have already been published to the report repository.

 Copyright © 1988-2007, Oracle. All rights reserved.

ReportManager Class Methods

The following are the ReportManager class methods, in alphabetical order.

AddSearchFieldCriteria

Syntax

AddSearchFieldCriteria(*&SearchAttribute*)

Description

Use the AddSearchFieldCriteria method to add an already instantiated and populated SearchAttribute object to the search.

Parameters

Parameter	Description
&SearchAttribute	Specify an already instantiated and populated SearchAttribute object.

Returns

A boolean, true if the criteria was added successfully, false otherwise.

Example

```
Local PSXP_REPORTMGR:SearchAttribute &oSearch;
Local integer &compOpSrch;

&CompOpSrch = oRptMgr.PSXP_SrchBegins;

& oSearch = create PSXP_REPORTMGR:SearchAttribute (&sAttrname, &sAttrVal, &compOp⇒
Srch);

&bResult = &oRptMgr.addSearchFieldCriteria(&oSearch);
```

See Also

Chapter 43, “XML Publisher Classes,” SearchAttribute Class Properties, page 1857

GetReportList

Syntax

GetReportList()

Description

Use the GetReportList method to return a list of the report definition objects that satisfy all the criteria you specified with the other methods associated with the ReportManager object.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

An array of ReportDefn objects.

See Also

Chapter 43, "XML Publisher Classes," ReportDefn Class Properties, page 1828

SetBurstFieldCriteria

Syntax

```
SetBurstFieldCriteria(BurstFld, BurstOp, BurstValue)
```

Description

Use the SetBurstFieldCriteria method to specify the burst criteria to be used with searching report definitions.

Parameters

Parameter	Description
BurstField	Specify the name of the burst field used to generate the report, as a string.
BusrtOp	Specify the search operator to be used *BurstField*, as a string. See Chapter 43, "XML Publisher Classes," Search Operator Values, page 1809.
BurstValue	Specify the pattern string to be matched with burst field value using *BurstOp*.

Returns

A boolean value, true if the method completes successfully, false otherwise.

See Also

Chapter 43, "XML Publisher Classes," BurstFieldName, page 1829

SetCaseSensitive

Syntax

```
SetCaseSensitive(IsCaseSensitive)
```

Description

Use the SetCaseSensitive method to specify the case sensitivity flag applicable to search field and bursting criteria for searching reports.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
IsCaseSensitive	Specify a boolean value: true to make the search case sensitive, false to make it case insensitive.

Returns

None.

SetDateCriteria

Syntax

SetDateCriteria(*createdDate, createdLastVal, createdUnit*)

Description

Use the SetDateCriteria method to specify the creation date criteria for searching reports.

Parameters

Parameter	Description
createdDate	Specify a date used to search for reports generated on and after this date. You should specify a null value (" ") for this parameter if using *createdLastVal* and *createdUnit*.
createdLastVal	Specify a value to be used n conjunction with *createdUnit*, as a number. If you specify a value, it is used and any value given for *createdDate*is ignored. If you are not using this search criteria, you should specify a null value (" ") for this parameter.
createdUnit	Specify a value to be used for reports generated during last *n* number of periodic units. This value is used with *createdLasVal*. For example, if *createdLastVal* is set to 10, and *createdUnit* is set to 1, the reports generated in the last ten days are returned. Valid values are:

Value	Description
1	Days
2	Hours
3	Minutes

Returns

None.

SetFolderCriteria

Syntax

SetFolderCriteria(*FolderName*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the SetFolderCriteria method to specify the folder criteria for searching reports.

Parameters

Parameter	Description
FolderName	Specify the name of the folder the report is posted in, as a string.

Returns

A boolean value, true if the search criteria is added successfully, false otherwise.

SetProcessInstanceCriteria

Syntax

SetProcessInstanceCriteria(*FromPID, ToPID*)

Description

Use the SetProcessInstanceCriteria method to specify the process instance range criteria for searching reports.

Parameters

Parameter	Description
FromPID	Specify a process instance Id to be used as the starting range from which to search, as a number. This value must be less than or equal to *ToPID*. You can specify a null value for this parameter, that is, (" ").
ToPID	Specify a process instance Id to be used as the ending range from which to search, as a number. This value must be greater than or equal to *FromPID*. You can specify a null value for this parameter, that is, (" ").

Returns

A boolean value: true if the search criteria is added successfully, false otherwise.

SetReportIDCriteria

Syntax

SetReportIDCriteria(*ReportId*)

Description

Use the SetReportIDCriteria method to specify a report Id for searching reports.

Parameters

Parameter	Description
ReportId	Specify a report Id to be used for searching for report definitions.

Copyright © 1988-2007, Oracle. All rights reserved.

Returns

A boolean value: true if the search criteria was set successfully, false otherwise.

SetUserIdCriteria

Syntax

SetUserIdCriteria(*UserId*)

Description

Use the SetUserIdCriteria method to specify a user Id to be used for searching reports intended to be accessed by that recipient.

Parameters

Parameter	Description
UserId	Specify the user Id to be matched with the intended recipient of the report.

Returns

A boolean: true if the search criteria is set successfully, false otherwise.

SearchAttribute Class

The SearchAttribute class is one of the report manager search classes. Use the SearchAttribute class to discovery name-value pairs for the search criteria for reports that have already been published to the report repository. In addition, you can also specify a comparison operator to be used with the name-value pairs.

SearchAttribute Class Properties

The following are the properties for the SearchAttribute class, in alphabetical order.

attrName

Description

This property returns the attribute name used with search.

This property is read-only.

attrValue

Description

This property returns the attribute value used with search???

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-only.

compareOp

Description

This property returns the comparison operator used with search.

This property is read-only.

XML Generation Class Constructors

Before you can use the XML generation classes, you must import the application package that contains them, as follows:

```
import PSXP_XMLGEN:*;
```

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the report manager definition classes, in alphabetical order.

QueryDS

Syntax

QueryDS()

Description

Use the QueryDS constructor to instantiate a QueryDS object.

Parameters

None.

Returns

A reference to a QueryDS object.

See Also

Chapter 43, “XML Publisher Classes,” QueryDS Class Methods, page 1859

RowsetDS

Syntax

RowsetDS()

Description

Use the RowsetDS constructor to instantiate a RowsetDS object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to a RowsetDS object.

See Also

Chapter 43, "XML Publisher Classes," RowsetDS Class Methods, page 1861

QueryDS Class

Use the QueryDS class to generates query schema in XSD based on the query object passed in.

QueryDS Class Methods

The following are the QueryDS class methods, in alphabetical order.

GetXMLData

Syntax

GetXMLData(*&Query*, *&Rowset*, *XSDLocation*)

Description

Use the GetXMLData method to create query data in XML format, based on the specified query and rowset objects.

Parameters

Parameter	Description
&Query	Specify an already open query object to be used as the basis for the XML data.
&Rowset	Specify an already instantiated and populated rowset object to be used as the basis for the XML data.
XSDLocation	Specify the XSD location for the XML schema validation. You can also specify a null value for this parameter, that is, (" ").

Returns

A string containing the generated XML data.

Example

```
Local PTXMLP_XMLGEN:QueryDS
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&oDS;&oDS = create PTXMLP_XMLGEN:QueryDS();

&oQuery = %Session.GetQuery();

&oQuery.Open(&sQueryName, False, False);
&recPromptRec = &oQuery.PromptRecord;
&bResult = Prompt(&sQueryName | " Prompts", "", &recPromptRec);
&oRowSet = &oQuery.RunToRowset(&recPromptRec, 0);
&sResult = &ds.getXMLData(&oQuery, &oRowSet, &strSchemaFile);
```

See Also

Chapter 30, "Query Classes," RunToRowset, page 1272

GetXMLSampleData

Syntax

GetXMLSampleData(*&Query*, *&Rowset*, *Rows*, *XSDLocation*)

Description

Use the GetXMLSampleData method to create sample query data in XML from the query and rowset objects specified.

If you specify a populated rowset object, the method generates the XML from it. If you specify Null (" ") or if the rowset object is empty, the method generates random sample data.

Parameters

Parameter	Description
&Query	Specify an already open query object to be used as the basis for the XML data.
&Rowset	Specify an already instantiated and populated rowset object to be used as the basis for the XML data. You can also specify a null value for this parameter, that is, (" ").
Rows	Specify the number of rows of XML data to be generated, as a number.
XSDLocation	Specify the XSD location for the XML schema validation. You can also specify a null value for this parameter, that is, (" ").

Returns

A string containing the generated XML data.

Example

```
Local PTXMLP_XMLGEN:QueryDS &oDS;

&oDS = create PTXMLP_XMLGEN:QueryDS();
&oQuery = %Session.GetQuery();
&oQuery.Open(&sQueryName, False, False);
&sResult = &oDS.getXMLSampleData(&oQuery, "", &nNumRows, &sSchemaFile);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 30, "Query Classes," page 1233

GetXSDSchema

Syntax

```
GetXSDSchema(&Query)
```

Description

Use the GetXSDSchema method to create a query schema in XSD format from the specified query object.

Parameters

Parameter	Description
&Query	Specify an already open query object to be used as the basis for the XSD.

Returns

A string containing the generated query schema in an XSD string.

RowsetDS Class

Use the RowsetDS class to generate XSD schema and XML data from a specified rowset object.

RowsetDS Class Methods

The following are the RowsetDS class methods, in alphabetical order.

AddXMLData

Syntax

```
AddXMLData(&ParentNode, &Rowset)
```

Description

Use the AddXMLData method to add the XML data of a rowset to the specified parent node. This method can be used to combine data of multiple rowsets in a single XML.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&ParentNode	Specify an already instantiated and populated XMLNode object. This is the parent node, where the XML data is to be added. Usually it is the DocumentElement of the combined XML.
&Rowset	Specify an already instantiated and populate rowset object containing the data to be added.

Returns

None.

Example

```
Local XmlDoc &xmlDoc;
Local XmlNode &xmlDocNode;
Local PSXP_XMLGEN:RowSetDS &oDS;

&oDS = create PSXP_XMLGEN:RowSetDS();

&xmlDoc = CreateXmlDoc("<?xml version='1.0'?><" | &sXmlDataName | "    xmlns:xsi="⇒
 | """ | "http://www.w3.org/2001/XMLSchema-instance" | """ | "/>");
&xmlDocNode = &xmlDoc.DocumentElement;

/* assume you have populated &rsResult1, &rsResult2 */

&dsRS.addXMLData(&xmlDocNode, &rsResult1);
&dsRS.addXMLData(&xmlDocNode, &rsResult2);
Return &xmlDoc.GenXmlString();
```

AddXSDSchema

Syntax

AddXSDSchema(*&ParentNode*, *&Rowset*)

Description

Use the AddXSDSchema method to add the XSD schema of a rowset to the specified parent node. This method can be used to combine schema of multiple rowsets in a single XSD.

Parameters

Parameter	Description
&ParentNode	Specify an already instantiated and populated XMLNode object. This is the parent node, where the XSD data is to be added. Usually it is the DocumentElement of the combined XSD.
&Rowset	Specify an already instantiated and populate rowset object containing the data to be added.

 Copyright © 1988-2007, Oracle. All rights reserved.

Returns

None.

Example

```
Local XmlDoc &xmlDoc;
Local XmlNode &xmlDocNode;
Local PSXP_XMLGEN:RowSetDS &oDS;

&oDS = create PSXP_XMLGEN:RowSetDS();

&xmlDoc = CreateXmlDoc("<?xml version='1.0'?><xsd:schema xmlns:xsd=" |    """" |⇒
 "http://www.w3.org/2001/XMLSchema" | """" | "/>");

&xmlDocNode = &xmlDoc.DocumentElement;
/* assume you have populated &rsResult1, &rsResult2 */

&dsRS.addXSDSchema(&xmlDocNode, &rsResult1);
&dsRS.addXSDSchema(&xmlDocNode, &rsResult2);
Return &xmlDoc.GenXmlString();
```

GetXMLData

Syntax

GetXMLData(*&Rowset*, *XSDLocation*)

Description

Use the GetXMLData method to create query data in XML format, based on the specified rowset object.

Parameters

Parameter	Description
&Rowset	Specify an already instantiated and populated rowset object to be used as the basis for the XML data.
XSDLocation	Specify the XSD location for the XML schema validation.

Returns

A string containing the generated XML data.

See Also

Chapter 30, "Query Classes," RunToRowset, page 1272

GetXSDSchema

Syntax

GetXSDSchema(*&Rowset*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GetXSDSchema method to create a schema in XSD format based on the specified rowset object.

Parameters

Parameter	Description
&Rowset	Specify an already instantiated and populated rowset object to be used as the basis for the XSD.

Returns

A string containing the generated query schema in an XSD string.

XML Publisher Engine Class Constructors

Before you can use the XML Publisher engine classes, you must import the application package that contains them, as follows:

```
import PSXP_ENGINE:*;
```

You must use the constructor for each class to instantiate an instance of that class. The following are the constructors for the XML Publisher engine classes, in alphabetical order.

EFTProcessor

Syntax

EFTProcessor()

Description

Use the EFTProcessor constructor to instantiate an EFTProcessor object.

Parameters

None.

Returns

An EFTProcessor object.

See Also

Chapter 43, “XML Publisher Classes,” EFTProcessor Class, page 1868

FOProcessor

Syntax

FOProcessor()

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the FOProcessor constructor to instantiate an FOProcessor object.

Parameters

None.

Returns

An FOProcessor object.

See Also

Chapter 43, "XML Publisher Classes," FOProcessor Class, page 1871

FormProcessor

Syntax

```
FormProcessor()
```

Description

Use the FormProcessor constructor to instantiate an FormProcessor object.

Parameters

None.

Returns

A FormProcessor object.

See Also

Chapter 43, "XML Publisher Classes," FormProcessor Class, page 1877

FOUtility

Syntax

```
FOUtility()
```

Description

Use the FOUtility constructor to instantiate a FOUtility object.

Parameters

None.

Returns

A FOUtility object.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 43, "XML Publisher Classes," FOUtility Class, page 1880

PageNumber

Syntax

```
PageNumber()
```

Description

Use the PageNumber constructor to instantiate a PageNumber object.

Parameters

None.

Returns

A PageNumber object.

See Also

Chapter 43, "XML Publisher Classes," PageNumber Class, page 1882

PDFMapTool

Syntax

```
PDFMapTool()
```

Description

Use the PDFMapTool constructor to instantiate a PDFMapTool object.

Parameters

None.

Returns

A reference to a PDFMapTool object.

See Also

Chapter 43, "XML Publisher Classes," PDFMapTool Class, page 1885

PDFMerger

Syntax

```
PDFMerger()
```

Description

Use the PDFMerger constructor to instantiate a PDFMerger object.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

None.

Returns

A reference to a PDFMerger object.

See Also

Chapter 43, “XML Publisher Classes,” PDFMerger Class, page 1888

Properties

Syntax

```
Properties()
```

Description

Use the Properties constructor to instantiate a Properties object.

Parameters

None.

Returns

A Properties object.

See Also

Chapter 43, “XML Publisher Classes,” Properties Class, page 1890

RTFProcessor

Syntax

```
RTFProcessor()
```

Description

Use the RTFProcessor constructor to instantiate an RTFProcessor object.

Parameters

None.

Returns

An RTFProcessor object.

See Also

Chapter 43, “XML Publisher Classes,” RTFProcessor Class, page 1892

Copyright © 1988-2007, Oracle. All rights reserved.

Watermark

Syntax

Watermark()

Description

Use the Watermark constructor to instantiate a Watermark object.

Parameters

None.

Returns

A Watermark object.

See Also

Chapter 43, "XML Publisher Classes," Watermark Class, page 1895

EFTProcessor Class

Use the EFTProcessor class to access the Oracle EFTProcessor API.

EFTProcessor Class Methods

The following are the EFTProcessor class methods, in alphabetical order.

GenerateOutput

Syntax

GenerateOutput(*EFTFile*, *XMLDataFile*, *OutputFile*, *Error*)

Description

Use the GenerateOutput method to create an output file from the specified EFT source file and XML data file.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
EFTFile	Specify the full path name of the EFTFile to be used. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XMLDataFile	Specify the full path name of the XML data file to be used. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFile	Specify the full path name of a text output file to be used to contain the output data. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completes successfully, false otherwise.

Example

```
Local PSXP_ENGINE:EFTProcessor &oEFT;
&oEFT = create PSXP_ENGINE:EFTProcessor();

&sErr = "";

&bResult = &oEFT.generateOutput(&sXSLFile, &XMLDatafile, &sOutputFile, &sErr);
```

See Also

Chapter 43, "XML Publisher Classes," GenerateXSL, page 1869

GenerateXSL

Syntax

GenerateXSL(*RTFTemplateFile*, *XSLOutputFile*, *Error*)

Description

Use the GenerateXSL method to create an XSL file from the specified RTF template file. *RTFTemplateName* must actually be an EFT source file.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RFTTemplateFile	Specify the full path name of the EFT source file. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XSLOutputFile	Specify the full path name of a file to contain the XSL output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

Example

```
Local PSXP_ENGINE:EFTProcessor &oEFT;
&oEFT = create PSXP_ENGINE:EFTProcessor();

&sErr = "";

&bResult = &oEFT.generateXSL(&sEFTFile, &sXSLOutputFile, &sErr);
```

See Also

Chapter 43, “XML Publisher Classes,” GenerateOutput, page 1868

EFTProcessor Class Properties

The following are the EFTProcessor class properties, in alphabetical order.

ConfFile

Description

Use the ConfFile property to specify a file that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify an already instantiated and populated Properties object that contains the configuration properties, use the ConfProp property instead.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 43, "XML Publisher Classes," ConfProp, page 1871

ConfProp

Description

Use the ConfProp property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify a file that contains the configuration properties, use the ConfFile property instead.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," ConfFile, page 1870

FOProcessor Class

Use the FOProcessor class to access the Oracle FOProcessor API.

FOProcessor Class Methods

The following are the FOProcessor class methods, in alphabetical order.

GenerateMergedOutput

Syntax

GenerateMergedOutput(*&XSLTemplateFileArray, &XMLDataFileArray, &XliffFileArray, OutputFileName, OutputFormat, Error*)

Description

Use the GenerateMergedOutput method to create formatted output from multiple XSL template files, XML data files and optionally, Xliff files.

The output is merged in the order of the files in the arrays, that is, the first part of the merged output comes from the first template with the first data file, as well as the first Xliff file (if appropriate), the second part from the second template and data file, and so on.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&XSLTemplateFileArray	Specify an already instantiated and populated array of string that contains the names of the XSL template files to be used.
&XMLDataFileArray	Specify an already instantiated and populated array of string that contains the names of the XML data files to be used.
&XliffFileArray	Specify an already instantiated and populated array of string that contains the names of the Xliff files to be used. If you do not want to use Xliff files, specify a null (" ") for this parameter.
OutputFileName	Specify the full path name of a file to be used for the merged output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFormat	Specify the format for the output. Valid values are:

Numeric Value	Constant Value	Description
1	&FORMAT_RTF	Specify RTF format.
2	&FORMAT_PDF	Specify PDF format.
3	&FORMAT_HTML	Specify HTML format
4	&FORMAT_EXCEL	Specify Excel format.

Parameter	Description
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

See Also

Chapter 43, "XML Publisher Classes," GenerateMergedOutputFO, page 1872 and Chapter 43, "XML Publisher Classes," GenerateOutput, page 1873.

Chapter 8, "Array Class," page 215

GenerateMergedOutputFO

Syntax

```
GenerateMergedOutputFO(&FOFIleArray, OutputFile, OutputFormat, Error)
```

Description

Use the GenerateMergedOutputFO method to create output from multiple formatted objects.

 Copyright © 1988-2007, Oracle. All rights reserved.

The output is merged in the order of the files in the arrays, that is, the first part of the merged output comes from the first template with the first data file, as well as the first Xliff file (if appropriate), the second part from the second template and data file, and so on.

Parameters

Parameter	Description
&FOFileArray	Specify an already instantiated and populated array of string containing the file names of the formatted objects you want to use.
OutputFile	Specify the full path name of the file to contain the output from the method. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFormat	Specify the format for the output. The valid values are:

Numeric Value	Constant Value	Description
1	&FORMAT_RTF	Specify RTF format.
2	&FORMAT_PDF	Specify PDF format.
3	&FORMAT_HTML	Specify HTML format
4	&FORMAT_EXCEL	Specify Excel format.

Parameter	Description
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

See Also

Chapter 43, “XML Publisher Classes,” GenerateMergedOutput, page 1871

Chapter 8, “Array Class,” page 215

GenerateOutput

Syntax

```
GenerateOutput(XSLTemplateFile, XMLDataFile, XLiffFile, OutputFile, OutputFormat, Error)
```

Description

Use the GenerateOutput method to create output from an XSL template file, an XML data file and, optionally, an Xliff file.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
XSLTemplateFile	Specify the full path name of the XSL template file that you want to use for creating the output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XMLDataFile	Specify the full path name of an XML file that contains the data you want to use for creating the output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XliffFile	Specify the name of an Xliff file to be used. If you do not want to use an Xliff file, specify a null (“ ”) for this parameter. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFile	Specify the full path name of a file to contain the output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFormat	Specify the format for the output. Valid values are:

Numeric Value	Constant Value	Description
1	&FORMAT_RTF	Specify RTF format.
2	&FORMAT_PDF	Specify PDF format.
3	&FORMAT_HTML	Specify HTML format
4	&FORMAT_EXCEL	Specify Excel format.

Parameter	Description
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

See Also

Chapter 43, “XML Publisher Classes,” GenerateOutputFO, page 1874

GenerateOutputFO

Syntax

```
GenerateOutputFO(FOFileName, OutputFile, OutputFormat, Error)
```

 Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the GenerateOutputFO method to create output from formatted objects.

Parameters

Parameter	Description
FOFileName	Specify the full path name of a file that contains the formatted object you want to use for output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFile	Specify the full path name of the file that you want to contain the output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFormat	Specify the format for the output. Valid values are:

Numeric Value	Constant Value	Description
1	&FORMAT_RTF	Specify RTF format.
2	&FORMAT_PDF	Specify PDF format.
3	&FORMAT_HTML	Specify HTML format
4	&FORMAT_EXCEL	Specify Excel format.

Parameter	Description
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

See Also

Chapter 43, "XML Publisher Classes," GenerateOutput, page 1873

FOProcessor Class Properties

The following are the FOProcessor class properties, in alphabetical order.

ConfFile

Description

Use the ConfFile property to specify a file that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

Copyright © 1988-2007, Oracle. All rights reserved.

If you want to specify an already instantiated and populated Properties object that contains the configuration properties, use the ConfProp property instead.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," ConfProp, page 1876

ConfProp

Description

Use the ConfProp property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify a file that contains the configuration properties, use the ConfFile property instead.

Example

```
Local PTXMLP_ENGINE:FOProcessor &oFO;
Local PTXMLP_ENGINE:Properties &oProp;

&oFO = create PTXMLP_ENGINE:FOProcessor();
&oProp = create PTXMLP_ENGINE:Properties();

&bResult = &oProp.setProperty("pdf-security", "true");
&bResult = &oProp.setProperty("pdf-permissions-password", "abc");
&bResult = &oProp.setProperty("pdf-encription-level", "0");

&oFO.ConfProp = &oProp;
```

See Also

Chapter 43, "XML Publisher Classes," ConfFile, page 1875

Locale

Description

Use the Locale property to specify the locale for the processing. If no value is specified for this property, the default system locale will be used.

The locale is specified using a two character ISO language code and a two character ISO country code (LC-CC). When you don't need to reflect country specific formatting, the locale code is made up of just the language code.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

FormProcessor Class

Use the FormProcessor class to access the Oracle FormProcessor API.

FormProcessor Class Methods

The following are the FormProcessor class methods, in alphabetical order.

ExtractFieldNames

Syntax

ExtractFieldNames(*PDFTemplateFileName*, *IncludeReservedNames*, *FieldNameArray*, *Error*)

Description

Use the ExtractFieldNames method to extract the field names from the specified PDF template.

Parameters

Parameter	Description
PDFTemplateFileName	Specify the full path name of the PDF template file that you want to extract the field names from. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
IncludeReservedNames	Specify whether to include reserved names in the list of field names. This parameter takes a boolean value: true to include the reserved names, false otherwise.
FieldNameArray	Specify an already instantiated array of string to contain the names of the fields from the PDF template. The array doesn't have to be created with a length.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

See Also

Chapter 43, "XML Publisher Classes," ExtractFieldValues, page 1877

ExtractFieldValues

Syntax

ExtractFieldValues(*PDFTemplateFileName*, *IncludeReservedNames*, *FieldValueArray*, *Error*)

Copyright © 1988-2007, Oracle. All rights reserved.

Description

Use the ExtractFieldValues method to extract the field values from the specified PDF template.

Parameters

Parameter	Description
PDFTemplateFileName	Specify the full path name of the PDF template file that you want to extract the field values from. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
IncludeReservedNames	Specify whether to include reserved names in the list of field values. This parameter takes a boolean value: true to include the reserved names, false otherwise.
FieldValueArray	Specify an already instantiated array of string to contain the values of the fields from the PDF template. The array doesn't have to be created with a length.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

See Also

Chapter 43, "XML Publisher Classes," ExtractFieldNames, page 1877

FillForm

Syntax

```
FillForm(PDFTemplateFile, XMLDataFile, PDFOutputFile, Error)
```

Description

Use the FillForm method to fill a PDF form by merging XML data into a PDF template.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
PDFTemplateFile	Specify the full path name of the file that contains PDF template that you want to fill out. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XMLDataFile	Specify the full path name of the file that contains the XML data to be used for filling out the PDF template. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
PDFOutputFile	Specify the full path name of the file where the output should be stored. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean value: true if the method completed successfully, false otherwise. If false, the *Error* parameter is populated with the returned error message.

FormProcessor Class Properties

The following are the FOProcessor class properties, in alphabetical order.

ConfFile

Description

Use the ConfFile property to specify a file that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify an already instantiated and populated Properties object that contains the configuration properties, use the ConfProp property instead.

This property is read-write.

See Also

Chapter 43, “XML Publisher Classes,” ConfProp, page 1880

Copyright © 1988-2007, Oracle. All rights reserved.

ConfProp

Description

Use the ConfProp property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify a file that contains the configuration properties, use the ConfFile property instead.

This property is read-write.

Example

```
Local PTXMLP_ENGINE:FOProcessor &oFO;
Local PTXMLP_ENGINE:Properties &oProp;

&oFO = create PTXMLP_ENGINE:FOProcessor();
&oProp = create PTXMLP_ENGINE:Properties();

&bResult = &oProp.setProperty("pdf-security", "true");
&bResult = &oProp.setProperty("pdf-permissions-password", "abc");
&bResult = &oProp.setProperty("pdf-encription-level", "0");

&oFO.ConfProp = &oProp;
```

See Also

Chapter 43, “XML Publisher Classes,” ConfFile, page 1879

Locale

Description

Use the Locale property to specify the locale for the processing. If no value is specified for this property, the default system locale will be used.

The locale is specified using a two character ISO language code and a two character ISO country code (LC-CC). When you don’t need to reflect country specific formatting, the locale code is made up of just the language code.

This property is read-write.

FOUtility Class

Use the FOUtility class to access the Oracle FOUtility API.

 Copyright © 1988-2007, Oracle. All rights reserved.

FOUtility Class Methods

The following are the FOUtility class methods, in alphabetical order.

GenerateFO

Syntax

GenerateFO(*XSLFile*, *XMLDataFile*, *OutputFOFile*, *Error*)

Description

Use the GenerateFO method to generate a formatted object file from the specified XSL file and XML data.

Parameters

Parameter	Description
XSLFile	Specify the full path name of a file that contains the XSL you want to use for the formatted objects. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XMLDataFile	Specify the full path name of an XML file that contains the data you want to use for the formatted objects. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
OutputFOFile	Specify the full path name of a file to be used for the formatted object output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

MergeFOs

Syntax

MergeFOs(*&FOFileArray*, *OutputFOFile*, *Error*)

Description

Use the MergeFOs method to merge formatted objects into an output file.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
&FOFileArray	Specify an already instantiated and populated array of formatted object file names.
OutputFOFile	Specify the full path name for the formatted object output file. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PTXMLP_ENGINE:FOUtility &oFOU;

&oFOU = create PTXMLP_ENGINE:FOUtility();
&sErr = "";
&asFOFiles= CreateArray(&sFOFile1);
&asFOFiles.Push(&sFOFile2);

&bResult = &oFOU.mergeFOs(&asFOFiles, &sOutputFOFile, &sErr);
```

FOUtility Class Property

The following is the FOUtility class property Prop.

Prop

Description

Use the Prop property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

This property is read-write.

PageNumber Class

Use the PageNumber class to set the page number for a merged PDF file. Objects in this class are used with the PDFMerger class property PageNumber.

 Copyright © 1988-2007, Oracle. All rights reserved.

PageNumber Class Properties

The following are the PageNumber class properties, in alphabetical order.

BackgroundFile

Description

Use the BackgroundFile property to specify a file to be used for the page number, as a string.

You must specify an absolute file name.

This property is read-write.

Considerations Using the BackgroundFile Property

You should either use the BackgroundFile property, or you should use the other properties associated with this class, like position or font. You cannot use both.

The file you specify must be a PDF file. The page size of the background PDF must be the same as the target PDF document, otherwise the page numbering doesn't work properly. All page numbering starts on the first page of the target document.

FontName

Description

Use the FontName property to specify the name of the font to be used for the page number, as a string. Valid values are:

- Courier
- Courier-Bold
- Courier-BoldOblique
- Helvetica (this is the default value)
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Oblique
- Symbol
- Times-Bold
- Time-BoldItalic
- Time-Italic
- Time-Roman
- ZapfDingbats

If you specify an invalid font name, the default (Helvetica) is used.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 43, “XML Publisher Classes,” FontSize, page 1884

FontSize

Description

Use the FontSize property to specify the size of the page number, as a number.

The default value is 8.

This property is read-write.

See Also

Chapter 43, “XML Publisher Classes,” FontName, page 1883

PositionX

Description

Use the PositionX property to specify the X axis position of the text page number in the merged document.

This property is read-write.

See Also

Chapter 43, “XML Publisher Classes,” PositionY, page 1884

PositionY

Description

Use the PositionY property to specify the Yaxis position of the text page number in the merged document.

This property is read-write.

See Also

Chapter 43, “XML Publisher Classes,” PositionX, page 1884

StartFromPageNum

Description

Use the StartFromPageNum property to specify the page index from which you’d like to start the page numbering.

For example, if you have a PDF document which has two cover pages, and you want to start printing page numbers on the document from the third page, specify a three for this property.

This property is read-write.

 Copyright © 1988-2007, Oracle. All rights reserved.

See Also

Chapter 43, “XML Publisher Classes,” StartNum, page 1885

StartNum

Description

Use the StartNum property to specify the page number to use as the first page number in the merged document. If you don’t specify a starting number, the pages are numbered starting from 1.

This property is read-write.

See Also

Chapter 43, “XML Publisher Classes,” StartFromPageNum, page 1884

PDFMapTool Class

Use the PDFMapTool class to access the PeopleSoft map tool.

PDFMapTool Class Methods

The following are the PDFMapTool class methods, in alphabetical order.

EnableMap

Syntax

EnableMap(*XSDFile*, *PDFTemplateFile*, *XMLInputFile*, *PDFOutputFile*)

Description

Use the EnableMap method to create a map-enabled PDF file based on the specified PDF template file.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
XSDFile	Specify the full path of the XSD (XML schema file) as a string. The file is used for field mapping between the PDF fields and the source data fields. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
PDFTemplateFile	Specify the full path of the PDF template file to be used, as a string. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
XMLInputFile	Specify the full path of the XML file containing sample data, as a string. The data is used for previewing. You can specify a null value for this parameter, that is, (""). You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
PDFOutputFile	Specify the full path of the map-enabled PDF file to be generated by this method. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.

Returns

None.

extractMap

Syntax

extractMap(*PDFFile, OutMapFile*)

Description

Use the extractMap method to extract a map file from a map-enabled PDF file.

Parameters

Parameter	Description
PDFFile	Specify the full path of the file containing the map-enabled PDF that you want to extract the map from, as a string. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
OutMapFile	Specify the full path of the map file generated by this method. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.

Returns

None.

Copyright © 1988-2007, Oracle. All rights reserved.

processForm

Syntax

processForm(*PDFTemplateFile, XMLDataFile, MapFile, OutputPDFFile*)

Description

Use the processForm method to generate a output from the PDF template and map-enabled file. This is not the same as ProcessReport, which performs full report generation.

Parameters

Parameter	Description
PDFTemplateFile	Specify the full path of the PDF template file to be used in processing the form, as a string. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
XMLDataFile	Specify the full path of the XML data file to be used for generating the form, as a string. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
MapFile	Specify the full path of the map file to be used when generating the form, as a string. When a value isn't provided for this parameter, the field names in the PDF form and the tag names in XML document is used to fill the data. You can specify a null value for this parameter, that is, (""). You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.
OutputPDFFile	Specify the full path to the output PDF file generated by this method. You must specify the complete path to the file, using the appropriate back or forward slashes as required by the operating system of the application server.

Returns

None.

PDFMapTool Class Property

The following is a description of the PDFMapTool class property Locale.

Locale

Description

Use the Locale property to specify the locale for the processing. If no value is specified for this property, the default system locale will be used.

The locale is specified using a two character ISO language code and a two character ISO country code (LC-CC). When you don't need to reflect country specific formatting, the locale code is made up of just the language code.

Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

PDFMerger Class

Use the PDFMerger class to access the Oracle PDFMerger API.

PDFMerger Class Method

The following is the PDFMerger class method MergePDFs.

MergePDFs

Syntax

MergePDFs(*PDFFileArray*, *PDFOutputFile*, *Error*)

Description

Use the MergePDFs method to merge the specified PDF files into a single output file.

The order of the files specified in the array are the order in which the files are merged.

Parameters

Parameter	Description
PDFFileArray	Specify an already instantiated and populated array of string containing the names of the PDF files that you want to merge together.
PDFOutputFile	Specify the full path name of the file you want populated with the merged PDF file. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PTXMLP_ENGINE:PDFMerger &oMerger;

&sErr = "";
&asNames = CreateArray(&sPdfFile1);
&asNames.Push(&sPdfFile2);
```

 Copyright © 1988-2007, Oracle. All rights reserved.

```
&bResult = &oMerger.mergePDFs(&asNames, &sOutputPdfFile, &sErr);
```

PDFMerger Class Properties

The following are the PDFMerger class properties, in alphabetical order.

ConfProp

Description

Use the ConfProp property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify a file that contains the configuration properties, use the ConfFile property instead.

Locale

Description

Use the Locale property to specify the locale for the processing. If no value is specified for this property, the default system locale will be used.

The locale is specified using a two character ISO language code and a two character ISO country code (LC-CC). When you don't need to reflect country specific formatting, the locale code is made up of just the language code.

This property is read-write.

PageNumber

Description

Use this property to specify an already instantiated and populated PageNumber object to be used with the merged document.

This property is read-write.

Example

```
Local PTXMLP_ENGINE:PDFMerger &oMerger;
Local PTXMLP_ENGINE:PageNumber &oPageNumber;

&oMerger = create PTXMLP_ENGINE:PDFMerger();
&oPageNumber = create PTXMLP_ENGINE:PageNumber();

&oPageNumber.FontName = "Symbol";
&oPageNumber.FontSize = 16;
&oPageNumber.PositionX = 300;
&oPageNumber.PositionY = 20;
&oMerger.oPageNumber = &oPageNumber;
```

Copyright © 1988-2007, Oracle. All rights reserved.

```
&sErr = "";
&asNames = CreateArray(&sPdfFile1);
&asNames.Push(&sPdfFile2);
&bResult = &oMerger.mergePDFs(&asNames, &sOutputPdfFile, &sErr);
```

See Also

Chapter 43, "XML Publisher Classes," PageNumber Class, page 1882

Watermark

Description

Use this property to specify an already instantiated and populated Watermark object to be used with the merged object.

This property is read-write.

Example

```
Local PTXMLP_ENGINE:PDFMerger &oMerger;
Local PTXMLP_ENGINE:Watermark &oWatermark;

&oMerger = create PTXMLP_ENGINE:PDFMerger();
&oWatermark = create PTXMLP_ENGINE:Watermark();

&oWatermark.Text = "DEMO";
&oWatermark.TextStartPosX = 200;
&oWatermark.TextStartPosY = 400;
&oMerger.oWatermark = &oWatermark;

&sErr = "";
&asNames = CreateArray(&sPdfFile1);
&bResult = &oMerger.mergePDFs(&asNames, &sOutputPdfFile, &sErr);
```

See Also

Chapter 43, "XML Publisher Classes," Watermark Class, page 1895

Properties Class

Use the Properties class to specify the properties for setting up the processor configuration. This class is used with the following classes:

- EFTProcessor
- FOProcessor
- FormProcessor
- FOUtility

 Copyright © 1988-2007, Oracle. All rights reserved.

- PDFMerger
- RTFProcessor

Properties Class Methods

The following are the Properties class methods, in alphabetical order.

GetProperty

Syntax

GetProperty(*Name*, *OutValue*)

Description

Use the GetProperty method to return the value of the specified property.

Parameters

Parameter	Description
Name	Specify the name of the property that you want to access the value.
OutValue	Specify a string to contain the value of the property you want to access.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PTXMLP_ENGINE:Properties &oProp;

&oProp = create PTXMLP_ENGINE:Properties();
%sValue = "";
&bResult = &oProp.getProperty("pdf-security", %sValue);
```

SetProperty

Syntax

SetProperty(*Name*, *Value*)

Description

Use the SetProperty method to specify the value of the specified property.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
Name	Specify the name of the property that you want to set.
Value	Specify the value of the property that you want to set.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PTXMLP_ENGINE:Properties &oProp;

&oProp = create PTXMLP_ENGINE:Properties();
&bResult = &oProp.setProperty("pdf-security", "true");
```

RTFProcessor Class

Use the RTFProcessor class to access the Oracle RTF processor API.

RTFProcessor Class Methods

The following are the RTFProcessor methods, in alphabetical order.

GenerateXSL

Syntax

GenerateXSL(*RTFTemplateFile*, *XSLOutputFile*, *Error*)

Description

Use the GenerateXSL method to generate an XSL file from the specified RTF template.

 Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RTFTemplateFile	Specify the full path name of an RTF template file, as a string. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XSLOutputFile	Specify the full path name of a file to contain the XSL output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PTXMLP_ENGINE:RTFProcessor &oRTF;

&oRTF = create PTXMLP_ENGINE:RTFProcessor();
&sErr = "";
&bResult = &oRTF.generateXSL(&sRTFFile, &sXSLFile, &sErr);
```

See Also

Chapter 43, "XML Publisher Classes," GenerateXSLXliff, page 1893

GenerateXSLXliff

Syntax

GenerateXSLXliff(*RTFTemplateName*, *XSLOutputFile*, *XliffOutputFile*, *Error*)

Description

Use the GenerateXSLXliff method to generate the XSL file and the Xliff from the specified RTF template.

Copyright © 1988-2007, Oracle. All rights reserved.

Parameters

Parameter	Description
RTFTemplateFile	Specify the full path name of an RTF template file, as a string. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XSLOutputFile	Specify the full path name of a file to contain the XSL output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
XliffOutputFile	Specify the full path name of a file to contain the Xliff output. Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.
Error	If any errors occur during processing, this parameter is populated with the text of the error message after processing.

Returns

A boolean: true if the method completed successfully, false otherwise. If this method returns false, the *Error* parameter is populated with the text of the error message that occurred.

Example

```
Local PTXMLP_ENGINE:RTFProcessor &oRTF;

&oRTF = create PTXMLP_ENGINE:RTFProcessor();
&sErr = "";
&bResult = &oRTF.generateXSLXliff(&sRTFFile, &sXSLFile, &sXliffFile, &sErr);
```

See Also

Chapter 43, “XML Publisher Classes,” GenerateXSL, page 1892

RTFProcessor Class Properties

The following are the RTFProcessor class properties, in alphabetical order.

ConfFile

Description

Use the ConfFile property to specify a file that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify an already instantiated and populated Properties object that contains the configuration properties, use the ConfProp property instead.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

See Also

Chapter 43, "XML Publisher Classes," ConfProp, page 1895

ConfProp

Description

Use the ConfProp property to specify an already instantiated and populated Properties object that contains the configuration to be used for processing the document. This property enables you to accommodate different system configurations.

If you want to specify a file that contains the configuration properties, use the ConfFile property instead.

See Also

Chapter 43, "XML Publisher Classes," ConfFile, page 1894

Locale

Description

‘

Use the Locale property to specify the locale for the processing. If no value is specified for this property, the default system locale will be used.

The locale is specified using a two character ISO language code and a two character ISO country code (LC-CC). When you don't need to reflect country specific formatting, the locale code is made up of just the language code.

This property is read-write.

Watermark Class

Use the watermark class to specify a watermark for a merged PDF file. Objects in this class are used with the PDFMerger class property Watermark.

You can generate a watermark either using text or an image. You can either use the text properties or the image properties. You cannot use both.

The following are the image watermark class properties:

- ImageFile
- ImageLowerLeftX
- ImageLowerLeftY
- ImageUpperRightX
- ImageUpperRightY

Copyright © 1988-2007, Oracle. All rights reserved.

The following are the text watermark class properties:

- Text
- TextAngle
- TextFontName
- TextFontSize
- TextStartPosX
- TextStartPosY

Watermark Class Properties

The following are the properties for the Watermark class, in alphabetical order.

ImageFile

Description

Use the ImageFile property to specify the full path name of an image file to be used for the watermark.

Forward or back slashes are used in the path according to the operating system of the application server or process scheduler server. That is, on Unix servers, the directory separator is a forward slash, while a Windows server returns a path with back slashes.

This property is read-write.

ImageFileLowerLeftX

Description

Use the ImageFileLowerLeftX property to specify the lower left, X-axis position for the watermark, as a number.

This property is read-write.

ImageFileLowerLeftY

Description

Use the ImageFileLowerLeftY property to specify the lower left, Y-axis position for the watermark, as a number.

This property is read-write.

ImageFileUpperRightX

Description

Use the ImageFileUpperRightX property to specify the upper right, X-axis position for the watermark, as a number.

 Copyright © 1988-2007, Oracle. All rights reserved.

This property is read-write.

ImageFileUpperRightY

Description

Use the ImageFileUpperRightY property to specify the upper right, Y-axis position for the watermark, as a number.

This property is read-write.

PageIndex

Description

Use the PageIndex property to specify the page where the watermark begins. For example, you might not want a watermark to display on a cover page. The default value is 0, which sets the watermark on all pages (page index starts at 1.)

This property is read-write.

Text

Description

Use the Text property to specify the text for the watermark, as a string.

This property is read-write.

TextAngle

Description

Use the TextAngle property to specify the angle of the watermark text across the page. You must specify a number between 1 and 360.

This property is read-write.

TextFontName

Description

Use the TextFontName property to specify the name of the font to be used for the text for the watermark.

This property is read-write.

TextFontSize

Description

Use the TextFontSize property to specify the size of the text font. The default value is 8.

This property is read-write.

Copyright © 1988-2007, Oracle. All rights reserved.

TextStartPosX

Description

Use the TextStartPosX property to specify the starting position, X-axis location for the text to be used as the watermark.

This property is read-write.

TextStartPosY

Description

Use the TextStartPosY property to specify the starting position, Y-axis location for the text to be used as the watermark.

This property is read-write.

XML Publisher Classes Example

The following example lists a complete example of the code first, then goes through the program line by line.

Generate and Publish a Report

In the following example, a report is created from an existing report definition. It is populated with an already instantiated and populated XmlDoc object, then published.

The following is the complete code sample: the steps explain each line.

```
import PSXP_RPTDEFNMANAGER:*;

Local XMLDoc &xmlDoc;
Local string &rptDefnId = "Financial";
Local string &LanguageCode = "ENG";
Local string &effdt = "2004-4-1";
Local string &outputfmt = "HTM";

Local String &reportPath = GetEnv("PS_SERVDIR") | "\files\reports";
Local String &folderName = "General";
Local string &serverName = "PSNT";
Local ReportDefn &rptDefn;

&rptDefn = Create ReportDefn(&rptDefnId);
&rptDefn.Get();
&rptDefn.SetRuntimeDataXMLDoc(&xmlDoc);
&rptDefn.ProcessReport("", &languageCd, &effdt, &outputfmt);

&rptDefn.Publish(&serverName, &reportPath, &folderName);
```

To generate and publish a report:

 Copyright © 1988-2007, Oracle. All rights reserved.

1. Import the appropriate application class.

   Because this program generates and publishs a report, you need to import the report manager definition class.

   ```
   import PSXP_RPTDEFNMANAGER:*;
   ```

2. Initialize variables.

   The variable declaration strings not only specify values for the variables, but give them type and scope as well. This can be very useful when debugging.

   ```
   Local XMLDoc &xmlDoc;
   Local string &rptDefnId = "Financial";
   Local string &LanguageCode = %Language_User;
   Local string &effdt = "2004-4-1";
   Local string &outputfmt = "HTM";

   Local String &reportPath = GetEnv("PS_SERVDIR") | "\files\reports";
   Local String &folderName = "General";
   Local string &serverName = "PSNT";
   Local ReportDefn &rptDefn;
   ```

3. Instantiate the report definition object and initialize it.

   After you instantiate a report definition object, you must initialize it and populate it using the Get method.

   ```
   &rptDefn = Create ReportDefn(&rptDefnId);
   &rptDefn.Get();
   ```

4. Specify the data for the report.

   This report uses an XmlDoc for the data source, so you must specify the runtime data source for the report before you process it.

   ```
   &rptDefn.SetRuntimeDataXMLDoc(&xmlDoc);
   ```

5. Process the report.

   You must process the report, generate a version of it for the report repository, before you can distribute the report.

   ```
   &rptDefn.ProcessReport("", &languageCd, &effdt, &outputfmt);
   ```

6. Publish the report.

   After you've generated the report, you may want to publish it to another location.

   ```
   &rptDefn.Publish(&serverName, &reportPath, &folderName);
   ```

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

APPENDIX A

Quick Reference for PeopleCode Classes

This appendix discusses:

- PeopleCode classes
- Session class methods and properties
- Mapping of functions to methods
- Mapping of old names to new names
- PeopleCode syntax quick reference

PeopleCode Classes

This section provides an overview of typographical conventions and visual cues and lists the PeopleCode classes in alphabetical order, along with the functions, methods and properties associated with that class.

Typographical Conventions and Visual Cues

This table describes the typographical conventions and visual cues that are used in this quick reference:

Typographical Convention or Visual Cue	Description
D	Identifies the default method for a class.
RO	Identifies a read-only property. Properties that aren't identified as read-only are read-write.

In addition, the usual PeopleCode typographical conventions are used to distinguish between different elements of the PeopleCode language, such as **keyword** to indicate keywords or syntax that must be entered exactly as shown, *variable* for arguments, parameters, and so on.

See Also

Enterprise PeopleTools 8.49 PeopleBook: PeopleCode Language Reference, "PeopleCode Language Reference Preface," PeopleCode Typographical Conventions

AESection

This section lists the functions, methods, and properties, as well as returns (if applicable), for the AESection class.

Copyright © 1988-2007, Oracle. All rights reserved.

Function

Function	Returns
GetAESection(*applid*, *ae_section* [, *effdt*])	AESection object

Methods

Method	Returns
AddStep(*ae_step_name* [,*NewStepName*])	None
Close()	None
Open(*ae_applid*, *ae_section*, [*effdt*])	AESection object
Save()	None
SetSQL(*action_type_string*, *string*)	None
SetTemplate(*ae_applid*, *ae_section*)	None

Property

Property	Returns
IsOpen	Boolean [RO]

Analytic Calculation Engine Classes

This section lists the functions, methods, and properties, as well as returns (if applicable), for the Analytic Calculation Engine Classes.

Functions

Function	Returns
CreateAnalyticInstance(*AnalyticType*, *AIID*, *Descr*, *AppClassName*, *MethodName*, *&RecordRef*, *Force*)	AnalyticInstance object
GetAnalyticInstance(*AIID*)	AnalyticInstance object

Copyright © 1988-2007, Oracle. All rights reserved.

AnalyticInstance Methods

Method	Returns
CheckAsyncStatus()	Number
CheckStatus()	Constant
Copy(*NewAIID*, *ForceDelete*, *&Record*)	None
Delete(*ForceUnload*, *&RecordRef*)	None
GetAnalyticModel((**Model.** *ModelName*)	AnalyticModel object
GetTraceLevel()	Integer
Load((**Message.** *MessageName*), *Sync*, *IdleTimeOut*)	String
RunAsync()	None
RunSync()	None
SetTraceLevel(*TraceLevel*)	Number
Terminate()	None.
Unload()	None

AnalyticInstance Properties

Property	Returns
AnalyticType	String[RO]
ID	Integer[RO]
Messages	Multi-dimensional array of any[RO]

AnalyticModel Methods

Method	Returns
AddMember(*DimName*, *MemberName*)	None
AttachTree(*DimName*, *TreeName*, *SetId*, *UserKeyValue*, *EffDt*, *NodeName*, *OverrideRecord*, *DetailStartLvl*, *TreeDiscardLvl*)	None
CalculateCube(*CubeName*, *Sync*)	None
DetachTree(*DimName*)	None
GetCubeCollection(*CubeCollName*)	CubeCollection object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetCellProperties(*CubeName*, *&Node*)	Array of string
GetMembers(*DimName*, *DimFilter*)	Two-dimensional array
GetTree(*DimName*)	Array of string
Recalculate(*Sync*)	String
RenameMember(*DimName*, *OrigMemberName*, *NewMemberName*)	None

AnalyticModel Property

Property	Returns
Messages	Multi-dimensional array of any[RO]

CubeCollection Methods

Method	Returns
CollapseNode(*DimName*, *&Node*)	None
DrillIntoNode(*DimName*, *&Node*)	None
DrillOutOfNode(*DimName*, *&Node*)	None
ExpandNode(*DimName*, *&Node*, *ExpandAll*)	None
GetData(*&DataRowset*, *StartRow*, *EndRow*[, *NetChanges*])	None
GetDimFilter(*DimName*)	String
GetDimSort(*DimName*)	Array
GetLayout(*&SlicerArray*, *&RowAxisArray*, *&ColumnAxisArray*)	None
GetRowCount()	Integer
GetSlice(*&SliceRecord*)	None
SetData(*&DataRowset)*	None
SetDimensionOrder(*&FieldNames*)	None
SetDimFilter(*DimName*, *DimFilter*)	None
SetDimSort(*DimName*, *IsAscending*[, *Key1*, *IsAscending2*, *Key2*, *IsAscending3*, *Key3*])	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
SetLayout(*&SlicerArray*, *&RowAxisArray*, *&ColumnAxisArray*	None
SetSlice(*&SliceRecord*)	None
ShowHierarchy(*DimName*, *Show*)	None
UnsetDimFilter(*DimName*)	None
UnsetDimSort(*DimName*)	None

CubeCollection Property

Property	Returns
Messages	Multi-dimensional array of any[RO]

Analytic Calculation Engine Metadata Classes

This section lists the methods and properties, as well as returns (if applicable), for the Analytic Calculation Engine metadata classes.

AnalyticModelDefn Methods

Method	Returns
AddCube(*CubeName*)	CubeDefn object
AddCubeCollection(*CubeCollectionName*)	CubeCollectionDefn object
AddDimension(*DimName*)	DimensionDefn object
AddExplicitDimensionSet (*ExplicitDimSetName*)	ExplicitDimensionSet object
AddOrganizer(*OrganizerName*)	OrganizerDefn object
AddUserFunction(*UserFunctionName*)	UserFunction object
CopyCube(*CubeName*, *NewCubeName*)	CubeDefn object
CopyCubeCollection(*CubeCollectionName*, *NewCubeCollectionName*)	CubeCollectionDefn object
CopyDimension(*DimName*, *NewDimName*)	DimensionDefn object
CopyExplicitDimensionSet(*OldExplicitDimSetName*, *NewExplicitDimSetName*)	ExplicitDimensionSet object
CopyTo(*NewModelName*)	AnalyticModelDefn object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
CopyUserFunction(*UserFunctionName*, *NewUserFunctionName*)	UserFunctionDefn object
Create()	None
Delete()	None
DeleteCube(*CubeName*, *ForceDelete*)	None
DeleteCubeCollection(*CubeCollectionName*, *ForceDelete*)	None
DeleteOrganizer(*OrganizerName*)	None
DeleteDimension(*DimensionName*)	None
DeleteExplicitDimensionSet(*ExplicitDimensionSetName*)	None
DeleteUserFunction(*UserFunctionName*, *ForceDelete*)	None
Get()	None
GetCube(*CubeName*)	CubeDefn object
GetCubeCollection(*CubeCollection*)	CubeCollectionDefn object
GetCubeCollectionNames()	Array of string
GetCubeNames()	Array of string
GetDimension(*DimName*)	DimensionDefn object
GetDimenstionNames()	Array of string
GetExplicitDimensionSet	ExplicitDimensionSet object
GetExplicitDimensionSetNames	Array of strings
GetOrganizer(*OrganizerName*)	OrganizerDefn object
GetOrganizerNames()	Array of string
GetUserFunction(*UserFunctionNames*)	UserFunctionDefn object
GetUserFunctionNames()	Array of string
Rename(*NewModelName*)	None
RenameCube(*CubeName*, *NewCubeName*, *ForceRename*)	None
RenameCubeCollection(*CubeCollectionName*, *NewCubeCollectionName*, *ForceRename*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
RenameDimension(*DimensionName*, *NewDimName*, *ForceRename*)	None
RenameExplicitDimensionSet(*ExplicitDimSetName*, *NewExplicitDimSetName*, *ForceRename*)	None
RenameOrganizer(*OrganizerName*, *NewOrganizerName*, *ForceRename*)	None
RenameUserFunction(*UserFunctionName*, *NewUserFunctionName*, *ForceRename*)	None
Save()	None
Validate()	Boolean

AnalyticModelDefn Properties

Property	Returns
CircularFomulaWarn	Boolean
Description	String
IsValid	Boolean[RO]
LongDescription	String
MaxDelta	Integer
MaxIterations	Integer
Messages	Multi-dimensional array of any[RO]
Name	String[RO]
ResolveCircularDeps	Boolean

DimensionDefn Properties

Property	Returns
AggregateSequence	Number[RO]
AggregationUserFunction	String
Comments	String
Name	String[RO]
TotalMemberName	String

Copyright © 1988-2007, Oracle. All rights reserved.

ExplicitDimensionSet Methods

Method	Returns
AttachDimension(*DimName*)	None.
DetachDimension(*DimName*)	None.
GetDimensionName()	Array of string.

ExplicitDimensionSet Properties

Property	Returns
Name	String [RO]
SequenceNumber	Number [RO]

CubeDefn Methods

Method	Returns
AttachDimension(*DimName*)	None
DetachDimension(*DimName*)	None
GetCauses(*CauseType*)	Array of string
GetCircularDeps(*DimName*)	Array of string
GetDimensionAggregate(*DimName*)	String
GetEffects(*EffectType*)	Array of string
GetDimensionNames()	Array of string
GetRule(*DimensionName*)	RuleDefn object
SetDimensionAggregate(*DimName*, *UserFunctionName*)	None
SetRule(*&RuleDefn*)	None
UsesDimension(*DimName*)	Boolean

CubeDefn Properties

Property	Returns
CalcAggregates	Boolean
Comments	String
DimensionCount	Number[RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
FormatType	Varies
IsVirtual	Boolean
Name	String[RO]
Rule	String
ValueDimensionName	String

CubeCollectionDefn Methods

Method	Returns
AttachCube(*CubeName*)	None
DetachCube(*CubeName*)	None
GetAggregateMapping(*PartName*, *IsCube*)	String
GetCubeNames()	Array of string
GetDimensionNames()	Array of string
GetDimSort(*DimensionName*)	Array of string
GetFieldMapping(*PartName*, *IsCube*)	String
GetFilter(*DimensionName*)	String
GetPersistAggregate(*DimensionName*)	String
SetAggregateMapping(*PartName*, *IsCube*, *AggregateFieldName*)	String
SetDimSort(*DimName*, *IsAscending*[,*CubeName1*, *IsAscending2*, *CubeName2*, *IsAscending3*, *CubeName3*])	None
SetFieldMapping(*PartName*, *IsCube*, *FieldName*)	None
SetFilter(*DimensionName*, *FilterName*)	None
SetPersistAggregate(*DimensionName*, *AggregateType*)	None
UsesCube(*CubeName*)	Boolean
UsesDimension(*DimensionName)*	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

CubeCollectionDefn Properties

Property	Returns
AggregateRecName	String
Comments	Number[RO]
CubeCount	Number[RO]
DimensionCount	Number[RO]
Name	String[RO]
RecordName	String

UserFunctionDefn Properties

Property	Returns
Comments	String
Name	String[RO]

UserFunctionDefn Methods

Method	Returns
GetRule( )	RuleDefn object
SetRule(*&Rule*)	None

OrganizerDefn Methods

Method	Returns
AttachPart(*PartName*, *PartType*)	None
DetachPart(*PartName*, *PartType*	None
GetPartNames()	Array of string
UsesPart(*PartName*, *PartType*)	Boolean

OrganizerDefn Properties

Property	Returns
Comments	String
Name	String[RO]

Copyright © 1988-2007, Oracle. All rights reserved.

RuleDefn Methods

Method	Returns
AddRuleExpression(*&Expr*)	None
GenerateRule ( )	None

RuleDefn Property

Property	Returns
RuleString	String[RO]

Assignment Method

Method	Returns
GenerateRule()	String

Assignment Properties

Property	Returns
Expression	Depends on value
Variable	String

Comparison Method

Method	Returns
GenerateRule ()	String

Comparison Properties

Property	Returns
Operand1	RuleExpression object
Operand2	RuleExpression object
Type	String [RO]

Constant Method

Method	Returns
GenerateRule()	String

Copyright © 1988-2007, Oracle. All rights reserved.

Constant Properties

Property	Returns
Type	String [RO]
Value	String [RO]

Cube Methods

Method	Returns
AddIndex(*&MemberReference*)	None
GenerateRule()	String
GetIndexes()	Array of MemberReference objects

Cube Property

Property	Returns
Name	String [RO]

ExpressionBlock Methods

Method	Returns
AddRuleExpression(*&Expr*)	None.
GetRuleExpressions()	Array of rule expression objects.

FunctionCall Methods

Method	Returns
AddArgument(*&RuleExpression*)	None
GenerateRule()	String
GetArguments()	Array of RuleExpression objects

FunctionCall Properties

Property	Returns
Name	String [RO]
Type	String

Copyright © 1988-2007, Oracle. All rights reserved.

MemberReference Method

Method	Returns
GenerateRule()	String

MemberReference Properties

Property	Returns
Dimension	String RO
Member	String RO

Operation Method

Method	Returns
GenerateRule()	String

Operation Properties

Property	Returns
Operand1	RuleExpression object
Operand2	RuleExpression object
Type	String

Variable Method

Method	Returns
GenerateRule()	String

Variable Properties

Property	Returns
Name	String RO
Type	String RO

Analytic Grid Classes

This section lists the functions, methods, and properties, as well as returns (if applicable), for the AnalyticGrid class.

Copyright © 1988-2007, Oracle. All rights reserved.

AnalyticGrid Function

Function	Returns
GetAnalyticGrid(**PAGE**.*pagename*, *gridname*)	AnalyticGrid object

AnalyticGrid Methods

Method	Returns
GetColumn(*ColumnName*)	GridColumn object
GetCubeCollection()	CubeCollection object
LoadData()	None
SetAnalyticInstance(*AIID*)	None
SetLayout(*&SlicerArray*, *&RowAxisArray*, *&ColumnAxisArray*)	None.

AnalyticGrid Properties

Property	Returns
InActive	Boolean
Label	String
ShowGridLines	Boolean
SlicerVisible	Boolean

Analytic Type Classes

This section lists the methods and properties as well as returns for Analytic Type classes.

AnalyticTypeDefn Methods

Method	Returns
AddModel(*ModelName*, *ModelType*	AnalyticTypeModelDefn object
AddRecord(*RecordName*)	AnalyticTypeRecordDefn object
Create()	
Copy(*NewAnalyticTypeName*, *ForceCopy*)	AnalyticTyepDefn object
Delete()	
DeleteModel(*ModelName*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
DeleteRecord(*RecordName*)	None
Get()	None
GetModels()	Array
GetRecordNames()	Array of string
Rename(*NewAnalyticTypeName*)	None
Save()	None

AnalyticTypeDefn Properties

Property	Returns
Comments	String
Description	String
Name	String[RO]
AppClassPath	String
OwnerID	String [RO]

AnalyticTypeModelDefn Properties

Property	Returns
Name	String[RO]
Type	String[RO]

AnalyticTypeRecordDefn Methods

Method	Returns
GetSelectedFields()	Array of string
SetSelectedFields(*SelectedFieldsArray*)	None
UnsetSelectedFields(*FieldName*)	None

AnalyticTypeRecordDefn Properties

Property	Returns
Callback	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Description	String
Name	String[RO]
Readable	Boolean
ReadOnce	Boolean
ScenarioManaged	Boolean
SyncOrder	Integer[RO]
Writeable	Boolean

Application Classes

This section lists the functions, methods, and properties, as well as returns (if applicable), for the Application classes.

Functions

Function	Returns
Class *classname* [{**Extends** | **Implements**}*Classname*] [Method_declarations] [Property_declarations] [**Protected** [Method_declarations] [Instance_declarations] [Constant_declaration]] [**Private** [Method_declarations] [Instance_declarations] [Constant_declaration]] **End-Class**	None
Get *PropertyName* *StatementList* **End-Get**	Depends on assignment of *StatementList*

Copyright © 1988-2007, Oracle. All rights reserved.

Function	Returns
Import *PackageName*:[*PackageName. . .*:]{*Classname* | *}	None
Interface *classname* [{**Extends** | **Implements**}*Classname*] [Method_declarations] [Property_declarations] [**Protected** [Method_declarations] [Instance_declarations] [Constant_declaration]] **End-Interface**	None.
Method *MethodName* *StatementList* **End-Method**	Depends on method
Set *PropertyName* *StatementList* **End-Set**	None

Array

This section lists the functions, methods, and properties, as well as returns (if applicable), for the Array class.

Functions

Function	Returns
CreateArray(*paramlist*)	Array object
CreateArrayAny([*paramlist*])	Array object
CreateArrayRept(*val*, *count*)	Array object
Split(*string*, *separator*)	Array object

Copyright © 1988-2007, Oracle. All rights reserved.

Methods

Method	Returns
Clone()	Array object
Find(*value*)	Index of an element
Get(*index*)	Element of array
Join([*separator* [, *arraystart, arrayend* [, *stringsizehint*]]])	String
Next(*&index*)	Boolean
Pop()	Value of last array element
Push(*paramlist*)	None
Replace(*start*, *length*, [*paramlist*])	None
Reverse()	None
Set(*index*)	None
Shift()	Value of first array element
Sort([*order*])	None
Subarray(*start*, *length*)	Array object
Substitute(*old_val*, *new_val*)	None
Unshift(*paramlist*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Properties

Property	Returns
Dimension	Number [RO]
Len	Number

BPEL

This section lists the constructors, methods, and properties, as well as returns (if applicable) for the BPEL classes AsyncFFSend, BPELUtil, and IBUtil.

Constructors

Constructor	Returns
AsyncFFSend()	AsyncFFSend object
BPELUtil()	BPELUtil object
IBUtil()	IBUtil object

AsyncFFSend Method

Method	Returns
OnRequestSend(*&Msg*)	Message object

BPELUtil Methods

Method	Returns
GetASyncBPELProcessInstanceURL(*TransactionID*)	String
GetBPELProcessBrowserURL(*NodeName*)	String
GetOperationType(*OperationName*)	String
GetSyncBPELProcessURL(*NodeName*, *MessageID*)	String
LaunchASyncBPELProcess(*OperationName*, *&Msg*, *Username*, *Password*)	String
LaunchSyncBPELProcess(*OperationName*, *&Msg*, *Username*, *Password*)	Message object

Copyright © 1988-2007, Oracle. All rights reserved.

IBUtil Methods

Method	Returns
GetBPELConsoleURL(*NodeName*)	String
GetBPELDomain(*NodeName*)	String
GetMessageId(*TransactionID*)	String
GetNode(*MessageID*)	String
GetNumberOfRoutings(*ServiceOperationName*)	Number
GetStatus(*TransactionID*)	String
GetTransactionId(*MessageId*)	String

Business Interlinks and BIDocs

This section lists the functions, methods, and properties, as well as returns (if applicable), for the Business Interlinks and BIDocs classes.

Business Interlink Functions

Function	Returns
GetBIDoc([*XMLString*])	BIDoc object
GetInterlink(**INTERLINK**.*name*)	Business Interlink object

Business Interlink Methods

Method	Returns
AddInputRow(*inputname*, *value*)	Boolean
BulkExecute(**RECORD**.*inputrec* [, RECORD.*outputrec*] [, *user_process_inst* | *user_operid*])	Number
Clear()	None
Execute()	Number
FetchIntoRecord(**RECORD**.*recname,* [, *user_process_inst* | *user_operid*])	Number

 Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
FetchIntoRowset(*&Rowset*)	Number
FetchNextRow(*outputname, value*)	Output row object
GetFieldCount()	Number
GetFieldType(*index*)	Number
GetFieldValue(*index*)	String
InputRowset(*&Rowset*)	Number
MoveFirst()	Boolean
MoveNext()	Boolean

Business Interlink Property

Property	Returns
StopAtError	Boolean

BIDoc Configuration Parameters

Configuration Parameters	Returns
URL	String
All other configuration parameters can be accessed like read-write properties.	Depends on type of Interlink plug-in.

BIDoc Methods

Method	Returns
AddAttribute(*attributename*, *attributevalue*)	Number
AddComment(*comment*)	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
AddProcessingInstruction(*instruction*)	Number
AddText(*text*)	Number
CreateElement(*elementname*)	BIDoc object
GenXMLString()	String
GetAttributeName(*attributenumber*)	String
GetAttributeValue({*attributenumber* | *attributename*})	String
GetNode({*nodenumber* | *nodename*})	BIDoc object

Chart

This section lists the functions, methods, and properties, as well as returns (if applicable), for the chart class.

Note. There is only one chart class. However, the methods and properties have been grouped into logical sections.

Chart Functions

Function	Returns
CreateObject("**Chart**")	Chart object
GetChart(*RecordName.FieldName*)	Chart object
GetGanttChart([*Recordname.FieldName*])	Chart object

Chart Data Methods

Method	Returns
Refresh()	None
SetData({*Record_Name* | *&Rowset*})	None
SetDataHints(*Record_Name.Field_Name*)	None

 Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
SetDataSeries(*Record_Name.Field_Name*)	None
SetDataURLs(*URLString*)	None
SetDataXAxis(*Record_Name.Field_Name*)	None
SetDataYAxis(*Record_Name.Field_Name*)	None

Chart Data Properties

Property	Returns
DataStartRow	Number
DataWidth	Number
ImageMap	Image map [RO]
IsTrueXY	Boolean

Chart Appearance Properties

Property	Returns
GridLines	String
GridLineType	String
Height	Number
IsDrillable	Boolean
IsPlainImage	Boolean
LineType	String
RotationAngle	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Style	String
StyleSheet	String
Type	String
Width	Number

Chart Color Methods

Method	Returns
SetColorArray(*&Array_of_Color*)	None
SetDataColor(*Record_Name.Field_Name*)	None

Chart Axis Methods

Method	Returns
SetXAxisLabels(*&Array_of_Any*)	None
SetYAxisLabels(*&Array_of_Any*)	None

Chart Axis Properties

Property	Returns
XAxisCross	String
XAxisLabelOrient	String
XAxisStyle	String
XAxisTicks	Number
YAxisLabelOrient	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
YAxisMax	Number
YAxisMin	Number
YAxisStyle	String
YAxisTicks	Number

Chart Title Properties

Property	Returns
MainTitle	String
MainTitleOrient	String
MainTitleStyle	String
XAxisTitle	String
XAxisTitleOrient	String
XAxisTitleStyle	String
YAxisTitle	String
YAxisTitleOrient	String
YAxisTitleStyle	String

Chart Legend Method

Method	Returns
SetLegend(*&Array_of_String*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Chart Legend Properties

Property	Returns
HasLegend	Boolean
LegendMaxEntries	Number
LegendPosition	String
LegendStyle	String

Chart Overlay Methods

Method	Returns
SetOLData({*Record_Name* | *&Rowset*})	None
SetOLDataSeries(*Record_Name.Field_Name*)	None
SetOLDataXAxis(*Record_Name.Field_Name*)	None
SetOLDataYAxis(*Record_Name.Field_Name*)	None

Chart Overlay Properties

Property	Returns
OLLineType	None
OLType	None

Scatter Chart Methods

Method	Returns
SetDataAnnotations(*Recordname.FieldName*)	None
SetDataGlyphScale(*Recordname.FieldName*)	None
SetOLDataAnnotations(*Recordname.FieldName*)	None
SetOLDataGlyphScale	None

Copyright © 1988-2007, Oracle. All rights reserved.

Scatter Chart Property

Property	Returns
ShowCrossHair	Boolean

TrueXY Line Chart Properties

Property	Returns
XAxisCrossPoint	Number
XAxisMax	Number
XAxisMin	Number
XAxisPrecision	Number
XAxisScaleResolution	Constant
YAxisCrossPoint	Number
YAxisMax	Number
YAxisMin	Number
YAxisPrecision	Number
YAxisScaleResolution	Constant

Gantt Chart Methods

Method	Returns
SetActualEndDate(*RecordName.FieldName*)	None
SetActualStartDate(*RecordName.FieldName*)	None
SetActualTaskBarColor(*Recordname.FieldName*)	None
SetChartArea(*Percentage*)	None
SetPlannedEndDate(*RecordName.FieldName*)	None
SetPlannedStartDate(*RecordName.FieldName*)	None
SetPlannedTaskBarColor(*RecordName.FieldName*)	None
SetTableXScrollbar(*Scroll_ArrowAmount*)	None
SetTaskAppData(*RecordName.FieldName1*[, *RecordName.FieldName2* [,*RecordName.FieldName3*. . .]])	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
SetTaskAppDataTitles(*&TitleArray*)	None
SetTaskBarURL(*RecordName.FieldName*)	None
SetTaskData({**Record.** *RecordName* | *&Rowset*})	None
SetTaskDependencyChildID(*RecordName.FieldName*)	None
SetTaskDependencyData({*RecordName* | *&Rowset*})	None
SetTaskDependencyType (*RecordName*, *FieldName*)	None
SetTaskDependencyURL(*RecordName.FieldName*)	None
SetTaskDependencyParentID(*RecordName.FieldName*)	None
SetTaskExpanded(*RecordName.FieldName*)	None
SetTaskHints(*RecordName.FieldName*)	None
SetTaskID(*RecordName.FieldName*)	None
SetTaskLabel(*RecordName.FieldName*)	None
SetTaskLevel(*RecordName.FieldName*)	None
SetTaskName(*RecordName.FieldName*)	None
SetTaskProgress(*RecordName.FieldName*)	None
SetTaskMilestone(*RecordName.FieldName*)	None.
SetTaskProgressBarColor(*RecordName.FieldName*)	None
SetWBSNumbering(*RecordName.FieldName*)	None

Gantt Chart Properties

Property	Returns
AxisEndDateTime	Datetime
AxisStartDateTime	Datetime
DataEndDateTime	Datetime
DataStartDateTime	Datetime
PixelsPerRow	Number [RO]
ShowTaskLabels	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
TaskDependencyLineType	Number
TaskTitle	String
TaskMilestoneGlyph	Number
XAxisPosition	Constant
YAxisPosition	Constant

Gantt Chart DateTimeAxis Specific Methods

Method	Returns
SetDayFormat(*Format*)	None
SetHourFormat(*Format*)	None
SetMinuteFormat(*Format*)	None
SetMonthFormat(*Format*)	None
SetSecondFormat(*Format*)	None
SetYearFormat(*Format*)	None

Component Interface

See Appendix A, "Quick Reference for PeopleCode Classes," Component Interface Class Methods and Properties, page 2018.

Exception

This section lists the functions, methods, and properties, as well as returns (if applicable), for the exception class.

Functions

Function	Returns
CreateException(*message_set*, *message_num*, *default_txt [*, *subslist*])	Exception object

Copyright © 1988-2007, Oracle. All rights reserved.

Function	Returns
Throw *Expression*	None
try *Protected StatementList* catch *QualifiedID &Id* *StatementList* end-try	Not applicable

Methods

Method	Returns
GetSubstitution(*Index*)	String
Output()	Depends on context
SetSubstitution(*Index*, *String*)	None
ToString([*AddContext*])	String

Properties

Property	Returns
Context	String [RO]
DefaultText	String
MessageNumber	Number [RO]
MessageSetNumber	Number [RO]
MessageSeverity	String [RO]
SubstitutionCount	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Crypt

This section lists the methods and properties, as well as returns (if applicable) for the Crypt class.

Function

Function	Returns
CreateObject("Crypt")	Crypt object

Methods

Method	Returns
FirstStep()	None
GoToStep(*StepNumber*)	None
LoadLibrary(*LibraryFile*, *LibraryID*)	None
NextStep()	None
Open(*ProfileName*)	None
SetParameter(*Name*,*Value*)	None
UpdateData(*Data*)	None

Properties

Property	Returns
Result	String RO
Verified	Boolean RO

Exception

This section lists the functions, methods, and properties, as well as returns (if applicable), for the exception class.

Functions

Function	Returns
CreateException(*message_set*, *message_num*, *default_txt* [, *subslist*])	Exception object

Copyright © 1988-2007, Oracle. All rights reserved.

Function	Returns
Throw *Expression*	None
try *Protected StatementList* catch *QualifiedID &Id* *StatementList* end-try	None

Methods

Method	Returns
GetSubstitution(*Index*)	String
Output()	String
SetSubstitution(*Index*, *String*)	None
ToString()	String

Properties

Property	Returns
Context	String [RO]
DefaultText	String
MessageNumber	Number [RO]
MessageSetNumber	Number [RO]
MessageSeverity	String [RO]
StackTrace	String [RO]
SubstitutionCount	Number [RO]

Field

This section lists the functions, methods, and properties, as well as returns (if applicable), for the field class.

Copyright © 1988-2007, Oracle. All rights reserved.

Function

Function	Returns
GetField([*recname.fieldname*])	Field object

Methods

Method	Returns
AddDropDownItem(*CodeString*, *DescriptionString*)	None
ClearDropDownList()	None
GetAuxFlag(*FlagNumber*)	Boolean
GetLongLabel(*LabelID*)	String
GetRelated(*recname.fieldname*)	Field object
GetShortLabel(*LabelID*)	String
SearchClear()	None
SetCursorPos({**PAGE**.*pagename* | **%Page**})	None
SetDefault()	None

Properties

Property	Returns
DataAreaCollapsed	Boolean
DecimalPosition	Number
DisplayFormat	String
DisplayOnly	Boolean
DisplayZero	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
DisplayZeroChanged	Boolean
EditError	Boolean
Enabled	Boolean
FieldLength	Number RO
FormatLength	Number RO
FormattedValue	None
IsAltKey	Boolean RO
IsAuditFieldAdd	Boolean RO
IsAuditFieldChg	Boolean RO
IsAuditFieldDel	Boolean RO
IsAutoUpdate	Boolean RO
IsChanged	Boolean RO
IsDateRangeEdit	Boolean RO
IsDescKey	Boolean RO
IsDuplKey	Boolean RO
IsEditTable	Boolean RO
IsEditXlat	Boolean RO
IsFromSearchField	Boolean RO
IsInBuf	Boolean RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
IsKey	Boolean RO
IsListItem	Boolean RO
IsNotUsed	Boolean RO
IsRequired	Boolean RO
IsSearchItem	Boolean RO
IsSystem	Boolean RO
IsThroughSearchField	Boolean RO
IsUseDefaultLabel	Boolean RO
IsYesNo	Boolean RO
Label	String
LabelImage	{**Image**.*imagename* | String}
LongTranslateValue	Any
MessageNumber	Number RO
MessageSetNumber	Number RO
Name	String RO
OriginalValue	Depends on field
ParentRecord	Record Object RO
PromptTableName	String RO
SearchDefault	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
SearchEdit	Boolean
SetComponentChanged	Boolean
ShortTranslateValue	Any
ShowRequiredFieldCue	Boolean
SmartZero	Boolean
SqlText	String
StoredFormat	String RO
Style	String
SystemIDFieldName	String RO
TimeStampFieldName	String RO
Type	String RO
Value	Depends on field
Visible	Boolean

File

This section lists the functions, methods, and properties, as well as returns (if applicable), for the file class.

Functions

Function	Returns
CreateDirectory(*path*, [*, pathtype*])	None
FileExists(*filename* [*, pathtype*])	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Function	Returns
FindFiles(*filespec_pattern* [, *pathtype*])	Array object
GetFile(*filename*, *mode* [, *charset*] [,*pathtype*])	File object

Methods

Method	Returns
Close()	None
CreateRowset()	Rowset object
GetPosition()	Number
GetString([*Strip_Line_Terminator*])	String
Open(*filename*, *mode* [, *charset*] [, *pathtype*])	None
ReadLine(*string*)	Boolean
ReadRowset()	Rowset object
SetFileId(*fileid*, *position*)	
SetFileLayout(FILELAYOUT.*filelayoutname*)	Boolean
SetPosition(*position*)	None
SetRecTerminator(*Terminator*)	None
WriteLine(*string*)	None
WriteRaw(*RawBinary*)	None
WriteRecord(*record*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
WriteRowset(*rowset* [, *Write_Data*])	Boolean
WriteString(*string*)	None

Properties

Property	Returns
CurrentRecord	String [RO]
IgnoreInvalidId	Boolean
IsError	Boolean [RO]
IsNewFileId	Boolean [RO]
IsOpen	Boolean [RO]
Name	String [RO]
UseSpaceForNull	Boolean

Grid and GridColumn

This section lists the functions, methods, and properties, as well as returns (if applicable), for the grid and gridcolumn classses.

Grid Functions

Function	Returns
GetGrid(**PAGE**.*pagename*, *gridname* [, *occursnumber*])	Grid object

Grid Method

Method	Returns
GetColumn(*columname*)[D]	Grid column object

Copyright © 1988-2007, Oracle. All rights reserved.

Grid Properties

Property	Returns
gridcolumnname	Grid column object
Label	String

GridColumn Properties

Property	Returns
Enabled	Boolean
Label	String
Name	String [RO]
Visible	Boolean

Internet Script

This section lists the system variables, methods, and properties, as well as returns (if applicable), for the internet script (iScript) classes.

Internet Script System Variables

System Variable	Returns
%Request	Request object
%Response	Response object

Request Methods

Method	Returns
GetContentBody()	String
GetCookieNames()	Array of string
GetCookieValue(*name*)	String
GetHeader(*name*)	String

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetHeaderNames()	Array of string
GetHelpURL(*HelpContext*)	String
GetParameter(*name*)	String
GetParameterNames()	Array of string
GetParameterValues(*Name*)	Array of string

Request Properties

Property	Returns
AuthTokenDomain	String RO
AuthType	String RO
BrowserPlatform	String RO
BrowserType	String RO
BrowserVersion	String RO
ByPassSignOn	Boolean RO
ContentURI	String RO
ExpireMeta	String RO
FullURI	String RO
HTTPMethod	String RO
LogoutURL	String RO
PathInfo	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Protocol	String RO
QueryString	String RO
RelativeURL	String RO
RemoteAddr	String RO
RemoteHost	String RO
RequestURI	String RO
RemoteUser	String RO
Scheme	String RO
ServerName	String RO
ServerPort	String RO
ServletPath	String RO
Timeout	Integer RO

Response Methods

Method	Returns
Clear()	None
CreateCookie(*name*)	Cookie object
GetCookie(*name*)	Cookie object
GetCookieNames()	Array of string
GetHeader(*name*)	String

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetHeaderNames()	Array of string
GetImageURL(*ImageName*)	String
GetJavaScriptURL(HTML.*Name*)	Srting
GetStyleSheetURL(STYLESHEET.*name*)	String
RedirectURL(*name*)	None
SetContentType(*Type*)	None
SetHeader(*name*, *value*)	None
UseSimpleURL({*True* | *False*})	None
Write(*HTML*)	None
WriteLine(*HTML*)	None

Response Properties

Property	Returns
Charset	String RO
DefaultStyleSheetName	String RO

Cookie Properties

Property	Returns
Domain	String
MaxAge	Number
Name	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Path	String
Secure	Boolean
Value	String

Java

This section lists the functions for the Java class.

Functions

Function	Returns
CopyFromJavaArray(*JavaArray*, *&PeopleCodeArray* [, *&RestrictionArray*])	None.
CopyToJavaArray(*&PeopleCodeArray*, *JavaArray* [, *&RestrictionArray*])	None.
CreateJavaArray(*ElementClassName[], NumberOfElements*)	Java object
CreateJavaObject(*ClassName* [*ConstructorParams*])	Java object
GetJavaClass(*ClassName*)	Java object

Mail Classes

This section lists the functions, methods, and properties, as well as returns (if applicable), for the Mail (MultiChannel Framework) classes.

Functions

Function	Returns
DeQueue(*physical queue ID*, *task type*, *task number*, *agent ID*)	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Function	Returns
EnQueue(*logical queue* , *task type*, *Relative URL*, *Language_Code* [,*subject*][, *agent ID*][, *overflow timeout*][, *escalation timeout*][, *cost*][, *priority*][, *skill level*])	Number
Forward(*from physical queue ID*, *from agent ID*, *task number*, *task type*, *to logical queue ID*[, *to agent ID*])	Number
InitChat(*logical queue ID*, *application data URL*, *customer username*, [*chat_subject*][, *chat_question*][, *wizard_URL*][, *priority*][, *skill_level*] [, *cost*])	Number
MCFBroadcast(*ClusterID*,*QueueID*, *ChannelID*, *AgentState*, *AgentPresence*, *Message*, *MessageSetNumber*, *MessageNumber*, *DefaultMessage*, *SecurityLevel*, *ImportanceLevel*, *SenderId*, *NameValueString*)	None.
NotifyQ(*logical queue ID*, *task type*)	Number

Constructors

Constructor	Returns
MCFBodyPart()	MCFBodyPart object
MCFGetMail()	MCFGetMail object
MCFOutboundEmail()	MCFOutboundEmail object
MCFMailStore()	MCFMailStore object
SMTPSession()	SMTPSession object

MCFBodyPart Methods

Method	Returns
AddHeader(*HeaderName*, *HeaderValue*)	None.
GetHeader(*HeaderName*)	String
GetHeaderCount()	Number
GetHeaderName(*Index*)	String
GetHeaderNames()	String

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetHeaderValues(*Index*)	String
GetUnparsedHeaders()	String
SetAttachmentContent({*FilePath* | *FileURL*}, *FilePathType*, *FileName*, *FileDescr*, *OverrideContentType*, *OverrideCharset*)	None

MCFBodyPart Properties

Properties	Returns
AttachmentURL	String
Charset	String
ContentType	String
Description	String
Disposition	String
Filename	String
FilePath	String
FilePathType	String
IsAttachment	Boolean [RO]
MultiPart	String
Text	String

MCFEmail Properties

Properties	Returns
BCC	String
BounceTo	String
CC	String
From	String
Importance	String
Priority	Number
Recipients	String

Copyright © 1988-2007, Oracle. All rights reserved.

Properties	Returns
RefIDs	String
ReplyIDs	String
ReplyTo	String
Sender	String
Sensitivity	String
Subject	String
Text	String

MCFGetMail Methods

Method	Returns
CreateQuarantineFolder ( )	Boolean
GetCount ( )	Number
GetEmailCount(*user*, *password*, *server*, *node*)	Number
ReadAllEmailHeadersWithAttach(*user*, *password*, *server*, *node*)	Rowset object
ReadEmails (*Count*)	String
ReadEmailsWithAttach(*User*, *password*, *server*, *node*, *count*)	Rowset object
ReadEmailsWithUID (*UID*)	Array of MCFInboundEmail objects
ReadEmailWithAttach(*User*, *password*, *server*, *node*, *UID*)	Rowset object
ReadHeaders ( )	Array of objects
RemoveEmail(*user*, *password*, *server*, *node*, *UID list*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
RemoveEmails(*UID_List*)	String
SetMCFEmail (*user*, *password*, *server*, *node*)	None

MCFGetMail Properties

Properties	Returns
AttachmentRoot	String
ContentTypes	String
ErrorCount	Number RO
IBNode	String
MailServer	String
Password	String
QuarantineCount	Number RO
QuarantineFolder	String
Status	String RO
UserID	String

MCFInboundEmail Methods

Properties	Returns
DumpToFile (*&File*)	None
GetAttachments ( )	Array of objects
GetFrom ( )	Array of string
GetParts ( )	Array of objects
GetSender ( )	Array of string
ReadFromDatabase (*Email_ID*)	Boolean
SaveToDatabase ( )	Number

Copyright © 1988-2007, Oracle. All rights reserved.

MCFInboundEmail Properties

Properties	Returns
AttachList	String
AttachSizes	String
DttmReceived	String
DttmSaved	String
DttmSent	String
IBNode	String
Language	String
MessageID	String
NotifyCC	String
NotifyTo	String
OffsetReceived	Number
OffsetSent	String
Server	String
Size	Number
Status	String RO
UID	String
User	String

MCFMailStore Methods

Method	Returns
AuthorizeEmailAttach(*email ID*, *authentication name*, *authentication type , operation*)	Boolean
DeleteEmail*(email ID*, [*forced delete*])	Boolean
RetrieveEmail(*email ID*)	Rowset object
StoreEmail(*rowset*, *row*)	Number

Copyright © 1988-2007, Oracle. All rights reserved.

MCFMailUtil Methods

Methods	Returns
DecodeText (*TextToDecode*, *&DecodedText*)	Boolean
DecodeWord (*WordToDecode*, *&DecodedWord*)	Boolean
EncodeText (*TextToEncode*, *charset*, *EncodingStyle*, *&EncodedText*)	Boolean
EncodeWord (*WordToEncode*, *charset*, *EncodingStyle*, *&EncodedWord*)	Boolean
IsEmailServerAvailable (*server*, *port*, *user*, *password*)	Boolean
ValidateAddress (*EmailAddress*)	Boolean

MCFMultiPart Methods

Methods	Returns
AddBodyPart (*&bodyPart*)	None
GetBodyPart(*Index*)	MCFBodyPart object
GetContentType ( )	String
GetCount ( )	Number

MCFMultiPart Property

Property	Returns
SubType	String

MCFOutboundEmail Methods

Methods	Returns
AddAttachment({*FilePath* | *FileURL*}, *FilePathType*, *FileName*, *FileDescr*, *OverrideContentType*, *OverrideCharset* [,*UploadPageTitle*])	None
AddHeader(*HeaderName*, *HeaderValue*)	None
GetHeader(*HeaderName*)	Array of string
GetHeaderCount ()	Integer
GetHeaderName(*Index*)	String
GetHeaderNames ( )	String

Copyright © 1988-2007, Oracle. All rights reserved.

Methods	Returns
GetHeaderValues (*Index*)	String
Send ( )	Number
SetSMTPParam(*ParamName*, *ParamValue*)	None

MCFOutboundEmail Properties

Properties	Returns
BackupSMTPPort	Number
BackupSMTPServer	String
BackupSMTPUserName	String
BCC	String
BounceTo	String
CC	String
Charset	String
ContentLanguage	String
ContentType	String
DefaultCharSet	String
Description	String
Disposition	String
ErrorDescription	String RO
ErrorDetails	String
Filename	String
FilePath	String
FilePathType	String
Importance	String
InvalidAddresses	String RO
IsAuthenticationReqd	Boolean
IsOkToSendPartial	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Properties	Returns
IsReturnReceiptReqd	Boolean
MessageNumber	Number RO
MessageSetNumber	Number RO
MultiPart	String
Priority	Number
Recipients	String
RefIDs	Number
ReplyIDs	Number
ReplyTo	String
ResultOfSend	Number RO
Sender	String
Sensitivity	String
SMTPPort	Number
SMTPServer	String
SMTPUserName	String
SMTPUserPassword	String
StatusNotifyOptions	String
StatusNotifyReturn	String
Subject	String
Text	String
TimeToWaitForResult	Number
UsedDefaultConfig	Boolean RO
UsedPrimaryServer	Boolean RO
ValidSentAddresses	String RO
ValidUnsentAddresses	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

SMTPSession Methods

Methods	Returns
Send(*&Email*)	Constant
SendAll(*&Emails*)	Array of numbers
SetSMTPParam(*ParamName*, *ParamValue*)	None

SMTPSession Properties

Properties	Returns
BackupPort	Number
BackupServer	String
BackupUserName	String
BackupUserPassword	String
IsAuthenticationReqd	Boolean
Port	Number
Server	String
UsedDefaultConfig	Boolean RO
UsedPrimaryServer	Boolean RO
UserName	String
UserPassword	String

Message

This section lists the functions, methods, and properties, as well as returns (if applicable), for the message class.

Message Functions

Function	Returns
AddSystemPauseTimes(*StartDay*, *StartTime*, *EndDay*, *EndTime*)	Boolean
CreateMessage(**OPERATION**.*messagename* [,*Message_Type*])	Message object

Copyright © 1988-2007, Oracle. All rights reserved.

Function	Returns
DeleteSystemPauseTimes(*StartDay*, *StartTime*, *EndDay*, *EndTime*)	Boolean
GetNRXmlDoc(*NRID*, *EntityName*)	XmlDoc object
IBPurgeDomainStatus()	Boolean
IBPurgeNodeStatus()	Boolean
NodeDelete(*nodeName*)	Boolean
NodeRename(*oldNodeName*, *newNodeName*)	Boolean
NodeSaveAs(*oldNodeName*, *newNodeName*)	Boolean
PingNode(*MsgNodeName*)	Array of number
PSIGWServiceRequest(*&Req_XmlDoc*, *&Resp_XmlDoc*)	Integer
ReValidateNRXmlDoc(*NRID*, *EntityName*)	Boolean

Message Methods

Method	Returns
Clone()	Message object
CopyPartRowset(*PartIndex*, *&Rowset*)	None.
CopyRowset(*source_rowset* [*, record_list*])	None
CopyRowsetDelta(*source_rowset* [*, record_list*])	None
CopyRowsetDeltaOriginal(*source_rowset*, [*, record_list*])	None
CreateNextSegment()	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
DeleteSegment(*Segment*)	None
ExecuteEdits([*editlevels*])	None
GenXMLPartString(*PartIndex*)	String
GenXMLString()	String
GetContentString([*SegmentIndex*])	String
GetPartAliasName(*PartIndex*)	String
GetPartName(*PartIndex*)	String
GetPartRowset(*PartIndex*)	Rowset object
GetPartVersion(*PartIndex*)	String
GetPartXmlDoc(*PartIndex*)	XmlDoc object
GetRowset([*version*])	Rowset object
GetSegment(*Segment*)	None
GetXmlDoc()	XmlDoc object
LoadXMLPartString(*PartIndex*, *XmlString*)	None
LoadXMLString(*XMLString*)	None
SegmentRestart(*TransactionID*, *Segment_index*, *segmentByDB*)	None
SetEditTable(*%PromptTable*, **RECORD**.*recname*)	None
SetXmlDoc(*&XmlDoc*)	None
UpdateSegment()	None

Copyright © 1988-2007, Oracle. All rights reserved.

Message Properties

Property	Returns
ActionName	String RO
AliasName	String RO
CurrentSegment	Number RO
DoNotPubToNodeName	String
IBInfo	IBInfo object RO
IsDelta	Boolean RO
IsEditError	Boolean RO
IsEmpty	Boolean RO
IsLocal	Boolean RO
IsOperationActive	Boolean RO
IsParts	BooleanRO
IsPartsStructured	BooleanRO
IsRequest	BooleanRO
IsSourceNodeExternal	BooleanRO
IsStructure	Boolean RO
Name	String RO
NRId	String RO
OperationName	StringRO
OperationVersion	StringRO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ParentTransactionId	String[RO]
PartCount	Number[RO]
PubNodeName	String[RO]
QueueName	String[RO]
QueueSeqId	Number[RO]
ResponseStatus	Number [RO]
SegmentCount	Number [RO]
SegmentsByDatabase	Boolean
Size	Number [RO]
TransactionId	String[RO]
Version	String[RO]

IntBroker Methods

Method	Returns
Cancel(*TransactionId*, *QueueName*, *DataType*, *SegmentIndex* | *TransactionIdArray*, *QueueNameArray*, *DataTypeArray*, *SegmentIndexArray*)	Boolean
ConnectorRequest(*&Message*)	Message object
ConnectorRequestUrl(*URL*)	String
DeleteOrphanedSegments(*TransactionId*)	Boolean
GetArchData(*TransactionId*, *SegmentIndex*)	String
GetArchIBInfoData(*TransactionId*, *ParentTransactionId*)	String
GetIBInfoData(*TransactionId*, *DataType*)	String
GetMessage([*TransactionId*, *DataType*])	Message object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetMessageErrors(*TransactionId*)	Array of string
GetMsgSchema(*MsgName*, *MsgVersion*)	String
GetSyncIBInfoData(*TransactionId*, *%LogType* [, *Archive*])	String
GetSyncLogData(*TransactionId*, *%LogType* [, *Archive*])	String
InBoundPublish(*&Message*)	Boolean
IsOperationActive(*OperationName* [,*OperationVersion*])	Boolean
MsgSchemaExists(*MsgName*, *MsgVersion*)	Boolean
Publish(*&Message* [, *&ArrayofNodeNames*] [,*IsEnqueued*])	None.
Resubmit(*TransactionId*, *QueueName*, *DataType*, *SegmentIndex* | *TransactionIdArray*, *QueueNameArray*, *DataTypeArray*, *SegmentIndexArray*)	Boolean
SetErrorMessage(*TransactionId*, *MsgSet*, *MsgNumber*, *Error_Location*, *ParmList*)	Boolean
SetStatus(*&Message*, *Status*)	None
SyncRequest([*&MsgArray*, *&NodeNames*])	Array of messages
Update(*&Message*)	Boolean
UpdateXmlDoc(*&XmlDoc*, *TransactionId*, *DataType*)	Boolean

IBInfo Methods

Method	Returns
AddAttachment(*Path*)	String
ClearAttachments()	None
DeleteAttachment(*Index* | *Content_Id*)	Boolean
GetAttachmentContentID(*Index*)	String
GetAttachmentProperty(*Content_ID*, *Property_Type*)	String
LoadConnectorProp(*ConnectorName*)	Boolean
LoadConnectorPropFromNode(*NodeName*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
LoadConnectorPropFromRouting(*RoutingDefnName*)	Boolean
SetAttachmentProperty(*Content_ID*, *Property_Type*, *Value*)	Boolean

IBInfo Properties

Property	Returns
AppServerDomain	String
IBConnectorInfo	ConnectorInfo object, RO
ConnectorOverride	Boolean
ConversationId	String
Cookies	String
DestinationNode	String, RO
ExternalMessageID	Number RO
ExternalOperationName	String
ExternalUserName	String
ExternalUserPassword	String
FinalDestinationNode	String, RO
HTTPSessionId	String
InReplyToID	String
MessageName	String, RO
MessageQueue	StringRO
MessageType	String, RO
MessageVersion	Number, RO
NodeDN	String, RO
NonRepudiationID	String, RO
NumberOfAttachments	String, RO
OperationType	String, RO
OperationVersion	String, RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
OrigNode	String, RO
OrigProcess	String, RO
OrigTimeStamp	String, RO
OrigUser	String, RO
RequestingNodeName	String, RO
RequestingNodeDescription	String, RO
ResponseAsAttachment	Boolean
SegmentsUnOrder	Boolean
SourceNode	String, RO
SyncServiceTimeout	Number
TransactionID	String, RO
UserIed	String, RO
VisitedNodes	Array of string, RO
WSA_MessageID	String
WSA_ReplyTo	String

ConnectorInfo Collection Methods

Method	Returns
AddConnectorProperties(*Name*, *Value*, *Type*)	Boolean
AddQueryStringArg(*Name*, *Value*)	Boolean
ClearConnectorProperties()	None
ClearQueryStringArgs()	None
DeleteConnectorProperties(*Name*)	Boolean
DeleteQueryStringArg(*Name*)	Boolean
GetConnectorPropertiesName(*Index*)	String
GetConnectorPropertiesType(*Index*)	String
GetConnectorPropertiesValue(*Index*)	String

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetNumberOf ConnectorProperties()	Number
GetNumberOfQueryStringArgs()	Number
GetQueryStringArgName(*Index*)	String
GetQueryStringArgValue(*Index*)	String

ConnectorInfo Collection Properties

Property	Returns
ConnectorClassName	String
ConnectorName	String
Cookies	String
PathInfo	String
RemoteFrameworkURL	String

Mobile

This section lists the system variable, functions, methods, and properties, as well as returns (if applicable), for the mobile classes.

Mobile System Variable

System Variable	Returns
%ThisMobileObject	Mobile object

Mobile Functions

Function	Returns
GenerateMobileTree(*CIObject* [,*CIObject_property*])	None
TransferMobilePage(**MOBILEPAGE**.*PageName*, *Tab*, *CIObject*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Mobile Methods

Method	Returns
Create()	Boolean
Find()	Collection of empty mobile objects
Get()	Boolean
GetName()	String
GetParent()	Mobile object
GetPropertyAttrsByName(*propertyname*)	PropertyAttrs object
GetPropertyByName(*propertyname*)	Depends on property. Will return a collection if *propertyname* is a collection.
GetPropertyInfoByName(*propertyname*)	PropertyInfo object
GetTopParent()	Mobile object
IsNew()	Boolean
IsNewAndNeverSaved()	Boolean
IsSameAs(*&MobileObject*)	Boolean
ReadBlobFromFile(*blobpropertyname*, *filename*	Boolean
Save()	Boolean
SetModified(*IsModified*	Boolean
SetPropertyByName(*propertyname*)	None
ValidateEnum(*propertyname*)	Boolean
ValuesDifferFromLastSync()	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
WasSaved()	Boolean
WriteBlobToFile(*blobpropertyname*, *filename*)	Boolean

Mobile Property

Property	Returns
PropertyInfoCollection	PropertyInfoCollection [RO]

PropertyInfo Collection Method

Method	Returns
Item(*number*)	PropertyInfo object

PropertyInfoCollection Property

Property	Returns
Count	Number

PropertyInfo Properties

Property	Returns
DefaultValue	Depends on the value [RO]
HasDefaultValue	Boolean [RO]
IsCollection	Boolean [RO]
IsRefToParent	Boolean [RO]
IsPeerReference	Boolean [RO]
IsSimpleApplicationProperty	Boolean [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
IsSystemProperty	Boolean RO
Name	String RO
PropertyInfoCollection	PropertyInfo Collection RO

CI Collection Method

Method	Returns
Item(*index*)	Mobile object

CI Collection Property

Property	Returns
Count	Number RO

Data Collection Methods

Method	Returns
DeleteItem(*location*)	Boolean
GetListAttrs(*propertyname*)	ListViewAttrs object
InsertItem(*location*)	Boolean
Item(*Index*)	Mobile object

Data Collection Property

Property	Returns
Count	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

ListViewAttrs Method

Method	Returns
IsSet(*attribute*)	Boolean

ListViewAttrs Properties

Property	Returns
DetailViewLabel	String
Label	String
ListViewLabel	String
Visible	Boolean

PropertyAttrs Method

Method	Returns
IsSet(*attribute*)	Boolean

PropertyAttrs Properties

Property	Returns
DisplayOnly	Boolean
InError	Boolean
Label	String
ShowRequiredCue	Boolean
Visible	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

PeerDefaultAttributes Method

Method	Returns
IsSet(*attribute*)	Boolean

PeerDefaultAttributes Properties

Property	Returns
DisplayOnly	Boolean
Label	String
Visible	Boolean

Notification

This section lists the constructors, methods, and properties, as well as returns (if applicable), for the Notification, NotificationAddress, NotificationTemplate and WorklistEntry classes.

Constructors

Constructors	Returns
Notification(*NotifyFrom*, *dttmCreated*, *language_cd*)	Notifiacation object
NotificationAddress(*Oprid*, *Description*, *Language*, *EmailId*, *Channel*)	NotificationAddress object
NotificationTemplate(*ComponentId*, *Market*, *TemplateId*, *TemplateType*)	NotificationTemplate object
OnAckForMarkedWorkedResp()	OnAckForMarkedWorkedResp object
WLEntryMarkedWorkedResp()	WLEntryMarkedWorkedResp object
WorkList()	Worklist object
WorklistEntry()	WorklistEntry object

Copyright © 1988-2007, Oracle. All rights reserved.

Constructors	Returns
WSWorklistEntry()	WSWorklistEntry object
WSWorklistEntryStatus()	WSWorklistEntryStatus object

Notification Method

Method	Returns
Send()	None

Notification Properties

Property	Returns
ContentType	String
dttmCreated	Datetime
EmailReplyTo	String
FileNames	Array of String
FileTitles	Array of String
language_cd	String
Message	String
NotifyBCC	Array of NotificationAddress objects
NotifyCC	Array of NotificationAddress objects
NotifyFrom	String
NotifyGuid	GUID
NotifyTo	Array of NotificationAddress objects

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
SourceComponent	String
SourceMarket	String
SourceMenu	String
Subject	String
Template	String

NotificationAddress Properties

Property	Returns
Channel	String
Description	String
Emailld	String
Language	String
Oprid	String

NotificationTemplate Methods

Method	Returns
GetAndExpandTemplate(*Language*, *Vars*)	Boolean
SetupCompVarsAndRcpts(*&Rowset_Context*)	String
SetupGenericVars(*&AryValues*)	String

Copyright © 1988-2007, Oracle. All rights reserved.

NotificationTemplate Properties

Property	Returns
ComponentId	String
Instruction	String
Language	String
Market	String
Priority	String
Responses	String
Subject	String
TemplateId	String
TemplateType	String
Text	String

Worklist Method

Method	Returns
Reassign*(FromOperid*, *ToOperid*)	Boolean

WorklistEntry Methods

Method	Returns
Create()	Number
GetResponseStatus()	Number
Reassign(*Operid*)	Boolean
Save()	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
SaveWithCustomData(*&Message*, *&FieldNameArray*, *&FieldValueArray*)	Message object‘
SelectByKey()	Boolean
SelectByMessageId()	Boolean
Update()	Number

WorklistEntry Properties

Property	Returns
actiondttm	Datetime
busactivity	String
buseventname	String
busprocname	String
commentshort	String
do_replicate_flag	String
instanceid	Number
instselectddttm	Datetime
inststatus	String
insttimeoutddttm	Datetime
instworkeddttm	Datetime
IsCreatedViaWebService	Boolean RO
oprid	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
originatorid	String RO
prevoprid	String
requestmessageid	String
ResponseStatus	String RO
syncid	Number
timedout	String
transactionid	Number
url	String
wl_priority	String
wldaystoselect	Number
wldaystowork	Number
worklistname	String

WSWorklistEntry Methods

Method	Returns
OnError(*&Message*)	Depends on implementation
OnNotify(*&Message*)	None

WSWorklistEntry Properties

Property	Returns
InstanceId	String RO
TransactionId	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

Optimization

This section lists the functions, methods, and properties, as well as returns (if applicable), for the Optimization classes.

Functions

Function	Returns
CreateOptEngine(*probinst*, {**%Synch** | **%ASynch**} [, *detailedstatus*] [,*processinstance*])	OptEngine object
CreateOptInterface()	OptInterface object
DeleteOptProbInst(*probinst*[, *detailedstatus*])	Constant
GetOptEngine(*probinst*[, *detailedstatus*])	OptEngine object
GetOptProbInstList(*ProblemType* , *OutputErrorCode* [, *Prefix*][, *detailedstatus*])	Array of string
InsertOptProbInst(*probinst, ProblemType*[, *detailedstatus*] [, *Description*])	Constant
IsValidOptProbInst(*probinst*[, *status*])	Constant

OptEngine Methods

Method	Returns
CheckOptEngineStatus()	Number
FillRowset(*PARAM_NAME, &Rowset* [, *functionstatus*])	Constant
GetDate(*PARAM_NAME*[,*status*])	Date
GetDateArray(*PARAM_NAME*)	Array of Date
GetDateTime(*PARAM_NAME*)	DateTime
GetDateTimeArray(*PARAM_NAME*)	Array of Datetime
GetNumber(*PARAM_NAME*)	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetNumberArray(*PARAM_NAME*)	Array of number
GetString(*PARAM_NAME*)	String
GetStringArray(*PARAM_NAME*)	Array of string
GetTime(*PARAM_NAME*)	Time
GetTimeArray(*PARAM_NAME*)	Array of Time
GetTraceLevel(*component*)	Constant
RunAsynch(*TRANSACTION, PARM_PAIRS*)	Constant
RunSynch(*TRANSACTION, PARM_PAIRS*)	Constant
SetTraceLevel(*Component, Severity*)	Constant
ShutDown()	Constant

OptEngine Properties

Property	Returns
DetailMsgs	Array of String [RO]
DetailStatus	Constant

OptBase Methods

Methods	Returns
GetParmDate(*ParmName*, *&ParmValue*)	Boolean
GetParmDateArray(*ParmName*, *&ParmValue*)	Boolean
GetParmDateTime(*ParmName*, *&ParmValue*)	Boolean
GetParmDateTimeArray(*ParmName*, *&ParmValue*)	Boolean
GetParmNumber(*ParmName*, *&ParmValue*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Methods	Returns
GetParmNumberArray(*ParmName*, *&ParmValue*)	Boolean
GetParmInt‘(*ParmName*, *&ParmValue*)	Boolean
GetParmIntArray(*ParmName*, *&ParmValue*)	Boolean
GetParmString(*ParmName*, *&ParmValue*)	Boolean
GetParmStringArray(*ParmName*, *&ParmValue*)	Boolean
GetParmTime(*ParmName*, *&ParmValue*)	Boolean
GetParmTimeArray(*ParmName*, *&ParmValue*)	Boolean
Init()	Boolean
OptDeleteCallback(*&record*)	Boolean
OptInsertCallback(*&Record*)	Boolean
OptPostUpdateCallback(*&OldRecord*, *&NewRecord*)	Boolean
OptPreUpdateCallback(*&OldRecord*, *&NewRecord*)	Boolean
OptRefreshCallback()	Boolean
SetOutputParmDate(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmDateArray(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmDateTime(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmDateTimeArray(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmNumber(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmNumberArray(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmInt(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmIntArray(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmString(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmStringArray(*ParmName*, *&ParmValue*)	Boolean
SetOutputParmTime(*ParmName*, *&ParmValue*)	Boolean
SetOutputTimeArray(*ParmName*, *&ParmValue*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

OptInterface Methods

Method	Returns
ActivateModel(*ModelID*, *SolverSettingID*	Constant
ActivateObjective(*Model_Name*, *Objective_Name*)	Constant
DeactivateModel(*ModelId*)	Constant
DumpMsgToLog(*LogSeverity*, *Message*)	None
FindRowNum(*&Record*[, *StartRow*[, *EndRow* [, *field_list*]]]	Number
GetSolution(*ModelId*, *varArrayId*, *SkipZero* [, *KeyFieldNames, KeyFieldValues*[, *Sollution*]]	Constant
GetSolutionDetail(*ModelId*, *SolutionType*, *Name*, *&Solution*)	Constant
IsModelActive(*ModelId*)	Boolean
RestoreBounds(*ModelId*[, *varArrayId*])	Constant
SetVariableBoungs(*ModelId*, *varArrayId*,*BoundType*, *LowerBound*, *UpperBound*, *&KeyRecord*[, *ChangeModelBounds*])	Constant
SetVariableType(*ModelId*, *varArrayId*, *varType*)	Constant
Solve(*ModelId*, *SolutionType*[, *objValue*[, *name-value_pairs*]])	Constant

Page

This section lists the functions, methods, and properties, as well as returns (if applicable), for the page class.

Function

Function	Returns
GetPage(**PAGE**.*pagename*)	Page object

Properties

Property	Returns
CopyURLLink	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
CustomizePageLink	Boolean
DisplayOnly	Boolean RO
HelpLink	Boolean
Name	String RO
NewWindowLink	Boolean
Visible	Boolean

PortalRegistry

See Appendix A, "Quick Reference for PeopleCode Classes," PortalRegistry Classes Methods and Properties, page 2030.

PostReport

This section lists the functions, methods, and properties, as well as returns (if applicable), for the PostReport class.

Function

Function	Returns
SetPostReport()	PostReport object

Methods

Method	Returns
AddDistributionOption(*DistIdType*, *DistId*)	Number
Put()	

Copyright © 1988-2007, Oracle. All rights reserved.

Properties

Property	Returns
ExpirationDate	Date
OutDestFormat	String
ProcessInstance	Number
ProcessName	String
ProcessType	String
ReportDesc	String
ReportFolder	String
ReportId	Number
ServerName	String

ProcessRequest

This section lists the functions, methods, and properties, as well as returns (if applicable), for the ProcessRequest class and the related PrcsApi class.

ProcessRequest Functions

Function	Returns
CreateProcessRequest([*ProcessType*, *ProcessName*])	ProcessRequest object
GetNextProcessInstance([*Commit*])	Number
SetupScheduleDefnItem(*ScheduleName, JobName*)	ProcessRequest object

Copyright © 1988-2007, Oracle. All rights reserved.

ProcessRequest Methods

Method	Returns
AddDistributionOption(*DistIdType*, *DistId* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	Number
PrintJobHTMLRpt()	String
PrintJobRqstRpt(*ProcessInstance*, *ItemInstance* [, *PrintJobTree*] [, *PrintDistList*] [, *PrintNotifyList*] [, *PrintSystemMessage*] [, *PrintApplicationMessage*] [, *PrintParamList*])	String
PrintSchdlHTMLRpt([*PrintJobTree*] [, *PrintDistList*] [, *PrintNotifyList*] [, *PrintMessageList*] [, *PrintParamList*])	String
RunJobSetNow()	None
Schedule()	None
SetEmailOption(*EmailSubject*, *EmailText*, *EmailAddress*, *EmailWebReport*, *EmailAttachLog* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None
SetItemFolder(*PortalFolder* [, *JobName*] [, *PrcsItemLevel*] [, *JobSeqNo*])	None
SetOutputOption(*OutputType*, *OutputFormat*, *OutputDest* [, *JobName*][, *PrcsItemLevel*] [, *JobSeqNo*])	None
UpdateRunStatus()	None

ProcessRequest Properties

Property	Returns
EmailAttachLog	Boolean
EmailSubject	String
EmailText	String
EmailWebReport	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
FileName	String
JobName	String
LanguageCd	String
NotifyTextMsgNum	Number
NotifyTextMsgSet	Number
OutDest	String
OutDestFormat	String
OutDestType	String
PortalFolder	String
ProcessInstance	Number
ProcessName	String
ProcessType	String
RunControlID	String
RunDateTime	DateTime
RunLocation	String
RunRecurrence	String
RunStatus	Number
Status	Number [RO]
TimeZone	String

Copyright © 1988-2007, Oracle. All rights reserved.

PrcsApi Constructor

Constructor	Returns
PrcsApi()	PrcsApi object

PrcsApi Methods

Method	Returns
getAllFileNames(*PrcsInstance*)	Array of string
notifyToWindows(*PrcsInstance*, *Message*)	Number

Query

See Appendix A, "Quick Reference for PeopleCode Classes," Query Classes Methods and Properties, page 2056.

Record

This section lists the functions, methods, and properties, as well as returns (if applicable), for the record class.

Functions

Function	Returns
CreateRecord(**RECORD**.*recname*)	Record object
GetRecord([**RECORD**.*recname*])	Record object

Methods

Method	Returns
CompareFields(*RecordObject*)	Boolean
CopyChangedFieldsTo(*RecordObj*)	None
CopyFieldsTo(*RecordObject*)	None
DBPatternMatch(*Value*, *Pattern*, *CaseSensitive*)	Boolean
Delete()	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
ExecuteEdits([*EditLevel*])	None
GetField(*n*/**FIELD**.*fieldname*)[D]	Field Object
Insert()	Boolean
Save([*CopyToOriginal*])	Boolean
SearchClear()	None
SelectByKey([*UseChangedBuffers*])	Boolean
SelectByKeyEffDt(*Date* [, *UseChangedBuffers*])	Boolean
SetDefault()	None
SetEditTable(*%PromptField*, **RECORD**.*recname*)	None
Update([*KeyRecord*])	Boolean

Properties

Property	Returns
FieldCount	Number [RO]
fieldname	Field Object [RO]
IsChanged	Boolean [RO]
IsDeleted	Boolean [RO]
IsEditError	Boolean [RO]
Name	String [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ParentRow	Row Object [RO]
RelLangRecName	String [RO]

Row

This section lists the functions, methods, and properties, as well as returns (if applicable), for the row class.

Function

Function	Returns
GetRow()	Row object

Methods

Method	Returns
CopyTo(*rowobject*)	Row Object
GetNextEffRow()	Row Object
GetPriorEffRow()	Row Object
GetRecord({*n* | **RECORD**.*recname*})[D]	Record Object
GetRowSet({*n* | **SCROLL**.*scrollname*})	Rowset Object
scrollname(*n*)	Row Object

Properties

Property	Returns
ChildCount	Number [RO]
DeleteEnabled	Boolean
IsChanged	Boolean [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
IsDeleted	Boolean RO
IsEditError	Boolean RO
IsNew	Boolean RO
ParentRowSet	Rowset object RO
recname	Record object RO
RecordCount	Number RO
RowNumber	Number RO
Selected	Boolean
Style	String
Visible	Boolean

Rowset

This section lists the functions, methods, and properties, as well as returns (if applicable), for the rowset class.

Functions

Function	Returns
CreateRowset({**RECORD**.*recname* | *&Rowset*} [, {**FIELD**.*fieldname*, **RECORD**.*recname* | *&Rowset*}] . . .)	Rowset object
GetLevel0()	Rowset object
GetRowset([SCROLL.*scrollname*])	Rowset object

Copyright © 1988-2007, Oracle. All rights reserved.

Methods

Method	Returns
ClearDeletesChanges()	None
CopyTo(*&DestRowset* [, **RECORD**.*SourceRecname*, **RECORD**.*DestRecname*]. . .)	None
DeleteRow(*n*)	Boolean
Fill([*wherestring* [, *bindvalue*] . . .])	Number
FillAppend([*wherestring* [, *bindvalue*] . . .])	Number
Flush()	None
FlushRow(*n*)	None
GetCurrEffRow()	Row object
GetNewEffRow()	Row object
GetRow(*n*)[D]	Row object
HideAllRows()	None
InsertRow(*n*)	Boolean
Refresh()	None
Select([*parmlist*], **RECORD**.*selrecord* [, *wherestr*, *bindvars*])	Number
SelectNew([*parmlist*], **RECORD**.*selrecord* [, *wherestr*, *bindvars*])	Number
SetDefault(*recname.fieldname*)	None
ShowAllRows()	None
Sort([*paramlist*,] *sort_fields*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Properties

Property	Returns
ActiveRowCount	Number RO
ChangeOnInit	Boolean
DataAreaCollapsed	Boolean
DBRecordName	String RO
DeleteEnabled	Boolean
EffDt	Date RO
EffSeq	Number RO
InsertEnabled	Boolean
IsEditError	Boolean RO
Level	Number RO
Name	String RO
ParentRow	Row Object RO
ParentRowSet	Rowset Object RO
RowCount	Number RO
SetComponentChanged	Boolean
TopRowNumber	Number RO

RowsetCache

This section lists the functions, methods, and properties, as well as the returns (if applicable) for the RowsetCache class.

Copyright © 1988-2007, Oracle. All rights reserved.

Functions

Function	Returns
CreateRowsetCache(*&Rowset*, [**Rowset.**]*Name*, *Description*)	RowsetCache object
GetRowsetCache(([**Rowset.**]*name*)	RowsetCache object

Methods

Method	Returns
Delete()	Boolean
Get()	Rowset object
Save([[**Rowset.**]*name*] [, *Description*])	Boolean

Properties

Property	Returns
Content	Rowset object
Description	String

Search

See Appendix A, "Quick Reference for PeopleCode Classes," Search Classes Methods and Properties, page 2022.

Session

See Appendix A, "Quick Reference for PeopleCode Classes," Session Classes, page 2010.

SOAPDoc

This section lists the functions, methods, and properties, as well as returns (if applicable), for the SOAPDoc class.

Function

Function	Returns
CreateSOAPDoc()	SOAPDoc object

Copyright © 1988-2007, Oracle. All rights reserved.

Methods

Method	Returns
AddBody()	None
AddEnvelope(*SOAP_Schema*)	None
AddFault(*Fault_Code*, *Fault_String*)	None
AddHeader()	None
AddMethod(*MethodName*, *IsRequest*)	None
AddParm(*ParmName*, *ParmValue*)	None
GetParmName(*Index*)	String
GetParmValue(*Index*)	String
ValidateSOAPDoc([1])	String

Properties

Property	Returns
BodyNode	XmlNode object RO
EnvelopeNode	XmlNode object RO
FaultCode	Number RO
FaultCodeS	String RO
FaultString	String RO
HeaderNode	XmlNode object RO
MethodName	String RO
MethodNode	XmlNode object RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ParmCount	Number [RO]
XmlDoc	XmlDoc object

SQL

This section lists the functions, methods, and properties, as well as returns (if applicable), for the SQL class.

Functions

Function	Returns
CreateSQL([{*sqlstring*| **SQL**.*SqlName*} [, *paramlist*]])	None
DeleteSQL([**SQL**.]*sqlname* [, *dbtype* [, *effdt*]]])	Boolean
FetchSQL([**SQL**.]*sqlname* [, *dbtype* [, *effdt*]])	String
GetSQL(**SQL**.*sqlname* [, *paramlist*])	SQL object
StoreSQL(*sqlstring*, [**SQL**.]*sqlname* [, *dbtype* [, *effdt* [, *ownerId* [, *Description*]]]])	None

Methods

Method	Returns
Close()	Boolean
Execute(*paramlist*)	Boolean
Fetch(*paramlist*)	Boolean
Open(*sql* [, *paramlist*])	None

Copyright © 1988-2007, Oracle. All rights reserved.

Properties

Property	Returns
BulkMode	Boolean
IsOpen	Boolean RO
LTrim	Boolean
ReuseCursor	Boolean
RowsAffected	Number RO
Status	Number RO
TraceName	String
Value	String RO

SyncServer

This section lists the functions, methods, and properties, as well as returns (if applicable), for the SyncServer class.

System Variable

System Variable	Returns
%SyncServer	SyncServer object

Methods

Method	Returns
AddReference(*SyncID*[, *CIDefnName*])	None
AddReferenceType(*CIDefnName*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
CheckForChanges(*SyncDateTime*, *TotalRowCount*[, *HasRefList*])	Boolean
SelectAll()	None

Properties

Property	Returns
ClientPlatform	String
ConflictAlgorithm	String
ConflictStatus	String
ExcludeProperty	String
ExportLocation	String
IsReferenceUnresolved	Boolean, RO
LasySyncDateTime	DateTime, RO
LastSyncRowCount	Number, RO
PropertyValue	String
SessionID	String RO
Synchronizing	String
SyncIDs	Array of number
TypeID	String
validateID	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
validateRowCount	Number RO
validateVersion	String

TransformData

This section lists the functions, methods, and properties, as well as returns (if applicable), for the TransformData class.

System Variable

System Variable	Returns
%TransformData	TransformData object

Properties

Property	Returns
DestMsgName	String RO
DestMsgVersion	String RO
DestMsgVersion	String RO
DestNode	String RO
SourceMsgName	String RO
SourceMsgVersion	String RO
SourceNode	String RO
Status	Number
XmlDoc	XmlDoc object

Tree

See Appendix A, "Quick Reference for PeopleCode Classes," Tree Classes Methods and Properties, page 2074.

 Copyright © 1988-2007, Oracle. All rights reserved.

Universal Queue

This section lists the constructors, methods, and properties, as well as returns (if applicable), for the universal queue classes.

Universal Queue Constructors

Constructor	Returns
Agent(*AgentID*)	Agent object
AgentPhysQueueProps()	Agent physical queue properties object
AgentPhysQueueTasks(*PhysicalQueueID*, *AgentID*)	Agent physical queue tasks object
LogicalQueue(*LogicalQueueID*)	Logical queue object
MCFFactory([*MAX_TASKLIST_ITEMS*])	MCFFactory object
PhysicalQueue(*PhysicalQueueID*)	Physical queue object
Task(*TaskNumber*)	Task object
TaskList(*Status*, *TaskType*, [*PhysicalQueueID*[, *AgentID*]])	Task list object
Util()	Util object

Agent Methods

Method	Returns
Delete()	Number
Refresh()	None
RefreshQTaskList(*PhysQ*)	None

Agent Properties

Properties	Returns
AgentID	String RO
AgentProps	AgentPhysQueueProps object RO
AgentTasks	AgentPhysQueueTask object RO
Buddy	Array of string RO
Language	Array of string RO
Name	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

Properties	Returns
Nickname	String RO
PhysicalQueueID	Array of string RO
TotalPhysicalQueues	Number RO

AgentPhysQueueProps Properties

Property	Returns
AgentID	String RO
PhysicalQueueID	String RO
SkillLevel	String RO
WorkLoad	Number RO

AgentPhysQueueTasks Method

Method	Returns
Refresh(*TaskList*)	None.

AgentPhysQueueTasks Properties

Property	Returns
AcceptedTaskList	Task list object RO
AssignedTaskList	Task list object RO
PhysicalQueueID	String RO

LogicalQueue Properties

Property	Returns
LogicalQueueID	String RO
PhysicalQueueID	Array of PhysicalQueue objects RO

MCFFactory Property

Property	Returns
LogicalQueue	Array of LogicalQueue objects RO

Copyright © 1988-2007, Oracle. All rights reserved.

PhysicalQueue Methods

Method	Returns
Refresh()	None
RefreshTaskList(*TaskList*)	None.

PhysicalQueue Properties

Property	Returns
AcceptedTaskList	Task list object RO
AssignedTaskList	Task list object RO
Agent	Array of agent objects RO
BrowserURL	String RO
EnqueuedTaskList	Task list object RO
EscalatedTaskList	Task list object RO
InternalURL	String RO
IsActive	Boolean RO
LogicalQueueID	String RO
OverflowedTaskList	Task list object RO
PhysicalQueueID	String RO
RENURLID	String RO
TotalAgents	Number RO

Task Methods

Method	Returns
Close(*Comment*)	None
Enqueue(*LogicalQueueID*,*AgentID*, *Timeout*, *ResponseTime*, *Cost*, *Priority*, *MinSkill*)	None.
Refresh()	None
RefreshStatus()	None

Copyright © 1988-2007, Oracle. All rights reserved.

Task Properties

Property	Returns
AgentID	String RO
ApplicationData	String RO
Comments	String
EnqueueTime	Datetime RO
EscalationTime	Datetime RO
Language	String RO
OriginalTime	Datetime RO
OverflowTime	Datetime RO
PhysicalQueueID	String RO
TiedAgentID	String RO

TaskList Method

Method	Returns
Refresh()	None

TaskList Properties

Properties	Returns
AgentID	String RO
PhysicalQueueID	String RO
Task	Array of tasks RO
TaskType	String RO
Total	Number RO

XmlDoc

This section lists the functions, methods, and properties, as well as returns (if applicable), for the XmlDoc and XmlNode classes.

Copyright © 1988-2007, Oracle. All rights reserved.

XmlDoc Functions

Function	Returns
CreateXmlDoc(*XmlString*)	XmlDoc object
GetNRXmlDoc(*NRID*, *NodeName*)	XmlDoc object
RevalidateNRXmlDoc(*NRID*, *EntityName*	Boolean
Transform(*Input, AE_Program_Name*, *Initial_Node_Name*, *Initial_Message__Name*, *Initial_Message_Version*, *Result_Node_Name*, *Result_Message_Name*, *Result_Message_Version*)	XmlDoc object
TransformEx(*XmlString*, *XsltString*)	String

XmlDoc Methods

Method	Returns
CopyRowset(*&InRowset*[, *MessageName*] [, *MessageVersion*])	Boolean
CopyToPSFTMessage(*targetDoc, srcPath, targetPath*)	Number
CopyToRowset(*&InRowset*[, *Message_Name*][, *Message_Version*])	Boolean
CreateDocumentElement(*TagName* [, *NamespaceURI*] [, *&DocType*])	XmlNode object
CreateDocumentType(*Name*, *PublicID*, *SystemID*)	String
GenFormattedXmlString()	String
GenXmlString()	String
GenFormattedXmlString()	String
GetElementsByTagName(*TagName*)	Array of XmlNode objects
LoadIBContent(*Content*[, *RootTagName*])	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
ParseXmlFromURL(*path*)	Boolean
ParseXmlString(*XmlString*)	Boolean

XmlDoc Properties

Property	Returns
DocumentElement	XmlNode object [RO]
IsNull	Boolean [RO]

XmlNode Methods

Method	Returns
AddAttribute(*Name*, *Value*)	None
AddAttributeNS(*NamespaceURI*, *AttributeName*, *Value*)	None
AddCDataSection(*Data*)	XmlNode object
AddComment(*Text*)	XmlNode object
AddElement(*TagName*)	XmlNode object
AddElementNS(*NameSpaceURI*, *TagName*)	XmlNode object
AddEntityReference(*Name*)	XmlNode object
AddNode(*XmlNode*)	XmlNode object
AddProcessInstruction(*Target*, *Data*)	XmlNode object
AddText(*Data*)	XmlNode object
CopyNode(*Node*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
FindNode(*Path*)	XmlNode object
FindNodes(*Path*)	Array of XmlNode objects
GenXmlString()	String
GetAttributeName(*index*)	String
GetAttributeValue({*Name* | *Index*})	String
GetCDataValue()	String
GetCDataValues()	Array of string
GetChildNode(*index*)	XmlNode object
GetElement()	XmlNode object
GetElements()	Array of XmlNode objects
GetElementsByTagName(*TagName*)	Array of XmlNode objects
GetElementsByTagNameNS(*NamespaceURI*, *TagName*)	Array of XmlNode objects
InsertCDataSection(*Data*, *Position*)	XmlNode object
InsertComment(*Data*, *Position*)	XmlNode object
InsertElement(*TagName*, *Position*)	XmlNode object
InsertElementNS(*NameSpaceURI*, *TagName*, *Position*)	XmlNode object
InsertEntityReference(*Name*, *Position*)	XmlNode object
InsertNode(*&NewNode*, {*&RefNode* | *Position*})	XmlNode object
InsertProcessInstruction(*Target*, *Data*, *Position*)	XmlNode object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
InsertText(*Data*, *Position*)	XmlNode object
RemoveAllChildNode()	None
RemoveChildNode({*Position* | *Node*})	XmlNode object

XmlNode Properties

Property	Returns
AttributesCount	Number [RO]
ChildNodeCount	Number [RO]
Index	Number [RO]
IsNull	Boolean [RO]
LocalName	String [RO]
NamespaceURI	String [RO]
NextSibling	XmlNode object [RO]
NodeName	String
NodePath	String [RO]
NodeType	String [RO]
NodeValue	String
ParentNode	XmlNode object [RO]
Prefix	String [RO]
PreviousSibling	XmlNode object [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

XML Publisher

This section lists the constructors, methods, and properties, as well as returns (if applicable), for the XML Publisher classes.

•

- Report manager search classes
- XML generation classes
- XML Publisher engine classes

Report Definition Manager Classes Constructors

Constructor	Returns
DataSourceDefn(*DataSourceID*, *DataSourceType*, *Public*)	DataSourceDefn object
ReportDefn(*ReportId*)	ReportDefn object
TemplateDefn(*TemplateID*)	TemplateDefn object
TemplateFile(*&TemplateDefn*, *EffDate*)	TemplateFile object
TranslationFile(*&TemplateDefn*, *EffectiveDate*, *LanguageCode*)	TranslationFile object

DataSourceDefn Class Methods

Method	Returns
Get()	None
GetData(*&PromptRecord*)	String
GetSampleData()	String
GetSchema	String

DataSourceDefn Class Properties

Property	Returns
ActiveFlag	Boolean^{RO}
Description	String^{RO}
IsPublic	Boolean^{RO}
LastUpdatedBy	String^{RO}
LastUpdateDTTM	DateTime^{RO}
Name	String^{RO}

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ObjectOwnerId	String[RO]
RegisteredBy	String[RO]
RegisteredDTTM	DateTime[RO]
Type	String[RO]

ReportDefn Class Methods

Method	Returns
DisplayOutput()	None
Get()	ReportDefn object
GetActiveTemplatesByDistributionChannel (*DistributionChannel*, *AsOfDate*)	Array of TemplateDefn objects
GetDataSource()	DataSourceDefn object
GetDefaultTemplate()	TemplateDefn object
GetOutDestFormatString(*OutDestFormat*)	String
GetPSQueryPromptRecord()	Record object
GetTemplate(*TemplateID*)	TemplateDefn object
GetTemplateList()	Array of TemplateDefn objects
PrintOutput(*DestPrinter*)	None
ProcessEtextReport(*TemplateID*, *LanguageCd*, *AsOfDate*)	String
ProcessReport(*TemplateId*, *LanguageCD*, *AsOfDate*, *OutputFormat*)	None
Publish(*ServerName*, *ReportPath*, *FolderName*, *ProcessInstanceId*)	None
SetPSQueryPromptRecord(*&Record*)	None
SetRuntimeDataRowset(*&Rowset*)	None
SetRuntimeDataXMLDoc(*&XmlDoc*)	None
SetRuntimeDataXMLFile(*FilePath*)	None
SetRuntimeProperties(*&NameArray*, *&ValueArray*)	None

Copyright © 1988-2007, Oracle. All rights reserved.

ReportDefn Class Properties

Property	Returns
AllowRecipientEntry	Boolean
AllowViewerEntry	Boolean
ArchiveAfter	Number
BurstFieldName	String
CategoryId	String
Description	String
DestinationPrinter	String
FolderName	String
ID	String
IsReadOnly	Boolean[RO]
LastUpdateDTTM	DateTime[RO]
LastUpdatedOprId	String[RO]
ObjectOwnerId	String[RO]
OutDestination	String
OutDestinationType	String
OutputEditable	String
RegisteredBy	String[RO]
RegisteredDTTM	DateTime[RO]
SecurityFieldName	String
SecurityIdType	String
SecurityJoinTable	String
Status	String[RO]
TemplateControlFieldName	String[RO]
UseBurstValueAsOutputFileName	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

TemplateDefn Class Methods

Method	Returns
Get()	None
GetActiveTemplateFile(*AsOfDate*)	TemplateFile object
GetTemplateFile()	TemplateFile object
GetTemplateFileList()	Array of TemplateFile objects

TemplateDefn Class Properties

Property	Returns
Description	String
DistributionChannel	String
ID	String[RO]
IsReadOnly	Boolean[RO]
IsSubTemplate	Boolean[RO]
LanguageCode	String[RO]
LastUpdateDTTM	DateTime[RO]
LastUpdatedOprId	String[RO]
ObjectOwnerId	String[RO]
RegisteredBy	String[RO]
RegisteredDTTM	DateTime[RO]
ReportCategoryId	String[RO]
Type	String[RO]

TemplateFile Class Methods

Method	Returns
GetFile()	String
GetMapFile()	String
GetTranslationFile(*LanguageCD*)	TranslationFile object
GetTranslationFileList()	Array of TranslationFile objects

Copyright © 1988-2007, Oracle. All rights reserved.

TemplateFile Class Properties

Property	Returns
EffectiveDate	DateTime[RO]
FileName	String[RO]
MapFileName	String[RO]
Status	String[RO]

TranslationFile Class Method

Method	Returns
GetFile()	String

TranslationFile Class Properties

Property	Returns
Description	String[RO]
FileName	String[RO]
LanguageCode	String[RO]
Status	String[RO]

XMLPManager Class Methods

Method	Returns
GetDatasourceDefnList(*DataSourceId*, *DataSourceIdOp*, *DataSourceType*, *Owner*, *Descr*, *DescrOp*, *ObjectOwnerId*, *Active*, *CaseSensitive*)	Array of DataSourceDefn objects
GetReportDefnList(*ReportID*, *ReportIdOp*, *Descr*, *DescrOp* , *TemplateId*, *TemplateIdOp*, *TemplateDescr*, *TemplateDescrOp*, *DatasourceType*, *DataSourceId*, *DataSourceIdOp*, *DataSourceOwner*, *TemplateType*, *ReportCategoryId*, *ReportStatus*, *CaseSensitive*)	Array of ReportDefn objects
GetTemplateDefnList(*TemplateId*, *TemplateIdOp*, *Descr*, *DescrOp*, *TemplateType*, *DistChannel*, *ReportCategoryId*, *IsSubtemplate*, *CaseSensitive*)	Array of TemplateDefn objects

Copyright © 1988-2007, Oracle. All rights reserved.

Report Manager Search Classes Constructors

Constructor	Returns
Report(*RptId*, *Prcsinstance*, *Contentid*, *Dbname*, *RptName*, *RptDescr*, *RptURL*, *RptCreateDttm*, *RptExpireDt*, *FldrName*)	Report object
ReportManager()	ReportManager object
SearchAttribute(*AttrName*, *AttrValue*, *CompareOperator*)	SearchAttribute object

Report Class Properties

Property	Returns
contentId	Number[RO]
CreatedDateTime	DateTime[RO]
DatabaseName	String[RO]
Description	String[RO]
ExpireDate	Date[RO]
FileURL	String[RO]
FolderName	String[RO]
ProcessInstanceID	String[RO]
ReportInstanceID	String[RO]
ReportName	String[RO]
ReportURL	String[RO]

ReportManager Class Methods

Method	Returns
AddSearchFieldCriteria(*&SearchAttribute*)	Boolean
GetReportList()	Array of ReportDefn objects
SetBurstFieldCriteria(*BurstFld*, *BurstOp*, *BurstValue*)	Boolean
SetCaseSensitive(*IsCaseSensitive*)	None
SetDateCriteria(*createdDate*, *createdLastVal*, *createdUnit*)	None
SetFolderCriteria(*FolderName*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
SetProcessInstanceCriteria(*FromPID*, *ToPID*)	Boolean
SetReportIDCriteria(*ReportId*)	Boolean
SetUserIdCriteria(*UserId*)	Boolean

SearchAttribute Class Properties

Property	Returns
attrName	String[RO]
attrValue	String[RO]
compareOp	String[RO]

XML Generation Class Constructors

Constructor	Returns
QueryDS()	QueryDS object
RowsetDS()	RowsetDS object

QueryDS Class Methods

Method	Returns
GetXMLData(*&Query*, *&Rowset*, *XSDLocation*)	String
GetXMLSampleData(*&Query*, *&Rowset*, *Rows*, *XSDLocation*)	String
GetXSDSchema(*&Query*)	String

RowsetDS Class Methods

Method	Returns
AddXMLData(*&ParentNode*, *&Rowset*)	None
AddXSDSchema(*&ParentNode*, *&Rowset*)	None
GetXMLData(*&Rowset*, *XSDLocation*)	String
GetXSDSchema(*&Rowset*)	String

Copyright © 1988-2007, Oracle. All rights reserved.

XML Publisher Engine Classes Constructors

Constructor	Returns
EFTProcssor()	EFTProcessor object
FOProcessor()	FOProcessor object
FormProcessor()	FormProcessor object
FOUtility()	FOUtility object
PageNumber()	PageNumber object
PDFMapTool()	PDFMapTool object
PDFMerger()	PDFMerger object
Properties()	Properties object
RTFProcessor()	RTFProcessor object
Watermark()	Watermark object

ETFProcessor Class Methods

Method	Returns
GenerateOutput(*EFTFile*, *XMLDataFile*, *OutputFile*, *Error*)	Boolean
GenerateXSL(*RTFTemplateFile*, *XSLOutputFile*, *Error*)	Boolean

EFTProcessor Class Properties

Property	Returns
ConfFile	String
ConfProp	Properties object

FOProcessor Class Methods

Method	Returns
GenerateMergedOutput(*&XSLTemplateFileArray*, *&XMLDataFileArray*, *&XliffFileArray*, *OutputFileName*, *OutputFormat*, *Error*)	Boolean
GenerateMergedOutputFO(*&FOFIleArray*, *OutputFile*, *OutputFormat*, *Error*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GenerateOutput(*XSLTemplateFile*, *XMLDataFile*, *XLiffFile*, *OutputFile*, *OutputFormat*, *Error*)	Boolean
GenerateOutputFO(*FOFileName*, *OutputFile*, *OutputFormat*, *Error*)	Boolean

FOProcessor Class Properties

Property	Returns
ConfFile	String
ConfProp	Properties object
Locale	String

FormProcessor Class Methods

Method	Returns
ExtractFieldNames(*PDFTemplateFileName*, *IncludeReservedNames*, *FieldNameArray*, *Error*)	Boolean
ExtractFieldValues(*PDFTemplateFileName*, *IncludeReservedNames*, *FieldValueArray*, *Error*)	Boolean
FillForm(*PDFTemplateFile*, *XMLDataFile*, *PDFOutputFile*, *Error*)	Boolean

FormProcessor Class Properties

Property	Returns
ConfFile	String
ConfProp	Properties object
Locale	String

FOUtility Class Methods

Method	Returns
GenerateFO(*XSLFile*, *XMLDataFile*, *OutputFOFile*, *Error*)	Boolean
MergeFOs(*&FOFileArray*, *OutputFOFile*, *Error*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

FOUtility Class Property

Property	Returns
Prop	Property object

PageNumber Class Properties

Property	Returns
BackgroundFile	String
FontName	String
FontSize	Number
PositionX	Number
PositionY	Number
StartFromPageNum	Number
StartNum	Number

PDFMapTool Class Methods

Method	Returns
EnableMap(*XSDFile*, *PDFTemplateFile*, *XMLInputFile*, *PDFOutputFile*)	None
extractMap(*PDFFile*, *OutMapFile*)	None
processForm(*PDFTemplateFile*, *XMLDataFile*, *MapFile*, *OutputPDFFile*)	None

PDFMapTool Class Property

Property	Returns
Locale	Property object

PDFMerger Class Method

Method	Returns
MergePDFs(*PDFFileArray*, *PDFOutputFile*, *Error*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

PDFMerger Class Properties

Property	Returns
ConfProp	Properties object
Locale	Sting
PageNumber	PageNumber object
Watermark	Watermark object

Properties Class Methods

Method	Returns
GetProperty(*Name*, *OutValue*)	Boolean
SetProperty(*Name*, *Value*)	Boolean

RTFProcessor Class Methods

Method	Returns
GenerateXSL(*RTFTemplateFile*, *XSLOutputFile*, *Error*)	Boolean
GenerateXSLXliff(*RTFTemplateName*, *XSLOutputFile*, *XliffOutputFile*, *Error*)	Boolean

RTFProcessor Class Properties

Property	Returns
ConfFile	String
ConfProp	Properties object
Locale	String

Watermark Class Properties

Property	Returns
ImageFile	String
ImageFileLowerLeftX	Number
ImageFileLowerLeftY	Number
ImageFileUpperRightX	Number
ImageFileUpperRightY	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
PageIndex	Number
Text	String
TextAngle	Number
TextFontName	String
TextFontSize	Number
TextStartPosX	Number
TextStartPosY	Number

Session Classes

This section lists:

- Session classes methods and properties
- Component interface classes methods and properties
- Search classes methods and properties
- PortalRegistry classes methods and properties
- Query classes methods and properties
- Tree classes methods and properties

Session Classes Methods and Properties

A session object controls access to the PeopleSoft APIs as well as the environment in which they run. This section only lists the system variable, methods, and properties, as well as returns (if applicable), for the session class and the associated classes used for controlling the environment. The properties, methods, and so of the API objects (query classes, tree classes, and so on) are detailed in separate sections.

Session System Variable

System Variable	Returns
%Session	Session object

Copyright © 1988-2007, Oracle. All rights reserved.

Session Methods

Method	Returns
AdvancedSearchQueries(*GetFavorites*, *QueryName*, *QueryNameOp*, *Descr*, *DescrOp*, *FolderName*, *FolderNameOp*, *RecordName*, *RecordNameOp*, *FieldName*, *FieldNameOp*, *TreeName*, *TreeNameOp*, *QueryType*, *OwnerType*, *CaseSensitive*)	Query collection
AdvancedSearchRecords(*RecordName*, *RecordNameOp*, *Descr*, *DescrOp*, *FieldName*, *FieldNameOp*, *TreeName*, *TreeNameOp*, *CaseSensitive*)	QueryDBRecord collection
FindCompIntfcs(*partial_name*)	Component Interface collection
FindPortalRegistries(*partial_name*)	PortalRegistry collection
FindQueryDBRecords()	QueryDBRecord collection
FindQueries()	Query collection
FindQueriesDateRange(*StartDateString*, *EndDateString*)	Query collection
GetCompIntfc([**CompIntfc**.]*name*)	Component Interface object
GetLocalNode()	Node object
GetNodes()	Node collection
GetPortalRegistry()	PortalRegistry object
GetQuery()	Query object
GetQuerySecurityProfile()	QuerySecurityProfile object
GetRemoteNodes()	Node collection
GetSearchIndexes()	SearchIndexes collection
GetSearchQuery()	SearchQuery object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
GetTree()	Tree object
GetTreeStructure()	Tree structure
SearchQueryDBRecord(*SearchType*, *Pattern*, *CaseSensitive*):	QueryDBRecord collection
SearchPrivateQueries(*QueryType, UserId, SearchType*, *Pattern*, *CaseSensitive*)	Query collection
SearchPublicQueries(*QueryType, SearchType*, *Pattern*, *CaseSensitive*)	Query collection

Session Properties

Property	Returns
ErrorPending	Boolean [RO]
PSMessages	PSMessage collection object
PSMessageMode	Number
RegionalSettings	Regional settings object
Repository	API Repository object
SuspendFormating	Boolean
TraceSettings	Trace settings object
WarningPending	Boolean [RO]

PSMessage Collection

Method	Returns
DeleteAll()	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
DeleteItem(*number*)	Boolean
First()	PSMessage object
Item(*number*)	PSMessage object
Next()	PSMessage object

PSMessage Collection Property

Property	Returns
Count	Number [RO]

PSMessage Properties

Property	Returns
ExplainText	String [RO]
MessageNumber	Number [RO]
MessageSetNumber	Number [RO]
Source	String [RO]
Text	String [RO]

RegionalSettings Properties

Property	Returns
ClientTimeZone	String
CurrencyFormat	Number
CurrencySymbol	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
DateFormat	Number
DateSeparator	String
DecimalSymbol	String
DigitGroupingSymbol	String
LanguageCD	String
UseLocalTime	Boolean
1159Separator	String
2359Separator	String

TraceSettings Properties

Property	Returns
API	Boolean
COBOLStmtTimings	Boolean
ConDisRollbackCommit	Boolean
DBSpecificCalls	Boolean
ManagerInfo	Boolean
NetworkServices	Boolean
NonSSBs	Boolean
OutputUNICODE	Boolean
PCExtFcnCalls	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
PCFcnReturnValues	Boolean
PCFetchedValues	Boolean
PCIntFcnCalls	Boolean
PCListProgram	Boolean
PCParameterValues	Boolean
PCProgramStatements	Boolean
PCStack	Boolean
PCStartOfPrograms	Boolean
PCTraceProgram	Boolean
PCVariableAssignments	Boolean
RowFetch	Boolean
SQLStatement	Boolean
SQLStatementVariables	Boolean
SSBs	Boolean
SybBindInfo	Boolean
SybFetchInfo	Boolean
TraceFile	String

API Repository Classes Methods and Properties

This section lists the method, properties, as well as the returns (if applicable) for the API Repository classes.

Copyright © 1988-2007, Oracle. All rights reserved.

Repository Properties

Property	Returns
Bindings	Bindings collection object RO
NameSpaces	Namespaces collection RO

Bindings Collection Property

Property	Returns
Count	Number RO

Bindings Collections Method

Method	Returns
Item(*number*)	Binding object

Bindings Property

Property	Returns
Name	String RO

Namespaces Collection Property

Property	Returns
Count	String RO

Namespaces Collection Methods

Method	Returns
Item(*Number*)	Namespaces object
ItemByName(*Name*)	Namespaces object

Namespaces Properties

Property	Returns
Classes	ClassInfo object RO
Name	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

ClassInfo Collection Property

Property	Returns
Count	Number [RO]

ClassInfo Collection Methods

Method	Returns
Item(*Number*)	ClassInfo object
ItemByName(*Name*)	ClassInfo object

ClassInfo Properties

Property	Returns
Documentation	String [RO]
Methods	MethodInfo collection object [RO]
Name	String [RO]
Properties	PropertyInfo collection object [RO]

MethodInfo Collection Property

Property	Returns
Count	String [RO]

MethodInfo Collection Methods

Method	Returns
Item(*Number*)	MethodInfo object
ItemByName(*Name*)	MethodInfo object

MethodInfo Properties

Property	Returns
Arguments	PropertyInfo collection object [RO]
Documentation	String [RO]
Name	String [RO]
Type	String [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

PropertyInfo Collection Properties

Property	Returns
Count	String RO

PropertyInfo Collection Methods

Method	Returns
Item(*Number*)	PropertyInfo object
ItemByName(*Name*)	PropertyInfo object

PropertyInfo Properties

Property	Returns
Documentation	String RO
Name	String RO
Type	String RO
Usage	Number RO

Component Interface Class Methods and Properties

This section lists the methods and properties, as well as returns (if applicable), for the component interface classes.

Component Interface Collection Methods

Method	Returns
First()	Component Interface structure (no data)
Item(*number*)	Component Interface structure (no data)
Next()	Component Interface structure (no data)

Component Interface Collection Property

Property	Retuns
Count	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

Component Interface Methods

Method	Returns
Cancel()	Boolean
CopyRowset(*&rowset* [,*IntialRow*] [, *record_list*])	None
CopyRowsetDelta(*&rowsetI*, [,*IntialRow*] [, *record_list*])	None
CopySetupRowset(*&rowset* [,*IntialRow*] [, *record_list*])	None
CopySetupRowsetDelta(*&rowsetI*, [,*IntialRow*] [, *record_list*])	None
Create()	Component Interface with data
Find()	Component Interface collection
Get()	Component Interface with data
GetPropertyByName(*string*)	Depends on property
Save()	Boolean
SetPropertyByName(*string, value*)	None

Component Interface Properties

Property	Returns
ComponentName	String [RO]
CreateKeyInfoCollection	Component InterfacePropertyInfo collection object [RO]
FindKeyInfoCollection	Component InterfacePropertyInfo collection object [RO]
GetDummyRows	Boolean
GetHistoryItems	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
GetKeyInfoCollection	Component InterfacePropertyInfo collection object RO
InteractiveMode	Boolean
PropertyInfoCollection	Component InterfacePropertyInfo collection object RO
StopOnFirstError	Boolean
Every user-defined property in a component interface definition can be used like a property on the instantiated object.	Depends on property

CompIntfPropInfo Collection Methods

Method	Returns
First()	Component InterfacePropertyInfo object
Item(*number*)	Component InterfacePropertyInfo object
Next()	Component InterfacePropertyInfo object

CompIntfPropInfo Collection Property

Property	Returns
Count	Number RO

CompIntfPropInfo Properties

Property	Returns
Format	String RO
IsCollection	Boolean RO
IsReadOnly	Boolean RO
Key	Boolean RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
LabelLong	String RO
LabelShort	String RO
Length	Number RO
Name	String RO
Prompt	Boolean RO
PropertyInfoCollection	Component InterfacePropertyInfo collection object
Required	Boolean RO
Type	String RO
Xlat	Boolean RO
YesNo	Boolean RO

Data Collection Methods

Method	Returns
CurrentItem()	Data collection item
DeleteItem(*number*)	None
GetEffectiveItem(*DateString*, *SeqNo*)	Data collection item
GetEffectiveItemNum(*DateString*, *SeqNo*)	Number
InsertItem(*number*)	Data collection item
Item(*number*)	Data collection item
ItemByKeys(*key_values*)	Data collection item

Copyright © 1988-2007, Oracle. All rights reserved.

Data Collection Properties

Property	Returns
Count	Number [RO]
CurrentItemNumber	Number [RO]

Search Classes Methods and Properties

This section lists the methods and properties, as well as returns (if applicable), for the search classes.

Note. Search class objects are instantiated from both the session and PortalRegisty objects.

SearchQuery Methods

Method	Returns
Execute(*Start*, *Size*)	SearchResult collection
Parse()	ParseResult collection

SearchQuery Properties

Property	Returns
HitCount	Number [RO]
Indexes	String
KnowledgeBase	String
Language	String
ProcessedCount	Number [RO]
QueryText	String
RequestedFields	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ScorePrecision	String
SortSpecifications	String

ParseResult Collection Methods

Method	Returns
First()	ParseResult object
Next()	ParseResult object or Null

ParseResult Collection Properties

Property	Returns
ErrorCount	Number [RO]
WarningCount	Number [RO]

ParseResult Properties

Property	Returns
Message	String [RO]
Severity	String [RO]

SearchResult Collection Methods

Method	Returns
First()	SearchResult
Item(*number*)	SearchResult
Next()	SearchResult

Copyright © 1988-2007, Oracle. All rights reserved.

SearchResult Collection Property

Property	Returns
Count	Number [RO]

SearchResult Properties

Property	Returns
Key	String [RO]
Score	String [RO]
ScoreAsNumber	Floating point number [RO]
SearchFields	SearchField collection [RO]

SearchField Collection Methods

Method	Returns
First()	SearchField
ItemByName(*Name*)	SearchField
Next()	SearchField

SearchField Collection Property

Property	Returns
Count	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

SearchField Properties

Property	Returns
Name	String [RO]
Value	String [RO]

SearchIndexes Collection Methods

Method	Returns
First()	SearchIndex object
ItemByName(*Name*)	SearchIndex object
Next()	SearchIndex object

SearchIndexes Collection Property

Property	Returns
Count	Number [RO]

SearchIndex Method

Method	Returns
Save()	Boolean

SearchIndex Properties

Property	Returns
ExtraOptions	Any
FSOpts	Search FS Options object [RO]
HTTPOpts	Search HTTP Options object [RO]
Languages	Search Language collection [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Location	String
Name	String RO
RecOpts	SearchRecord Options collection RO
Schedules	Search Schedule collection RO
Type	String

SearchRecord Options Properties

Property	Returns
Fields	SearchRecordField collection RO
Filter	String
IncrementalView	String
RecName	String RO
VeggieKey	String
ZoneOptions	String

SearchRecordField Collection Methods

Method	Returns
DeleteItem(*Record, Field*)	Boolean
First()	SearchRecordField object
InsertItem(*Record, Field*)	SearchRecordField object
ItemByName(*Record, Field*)	SearchRecordField object
Next()	SearchRecordField object

Copyright © 1988-2007, Oracle. All rights reserved.

SearchRecordField Collection Property

Property	Returns
Count	Number [RO]

SearchRecordField Properties

Property	Returns
FieldName	String [RO]
IsAttachment	Boolean
IsVerityField	Boolean
IsWordIndex	Boolean
RecordName	String [RO]

SearchLanguage Collection Methods

Method	Returns
DeleteItem(*Name*)	Boolean
First()	Search Language object
InsertItem(*LanguageCode, MapLanguageCode*)	Search Language object
ItemByName(*LanguageCode*)	Search Language object
Next()	Search Language object

SearchLanguage Collection Property

Property	Returns
Count	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

SearchLanguage Properties

Property	Returns
LanguageCd	String [RO]
MapLanguageCd	String [RO]

SearchSchedule Collection Methods

Method	Returns
DeleteItem(*Name*)	Boolean
First()	Search Schedule object
InsertItem(*RunCntrlID)*	Search Schedule object
ItemByName(*RunCntrlID*)	Search Schedule object
Next()	Search Schedule object

SearchSchedule Collection Property

Property	Returns
Count	Number [RO]

SearchSchedule Properties

Property	Returns
BuildType	String
RunCntrID	String [RO]
RunRecurrance	String
ServerName	String

Copyright © 1988-2007, Oracle. All rights reserved.

SearchHTTPOptions Properties

Property	Returns
AllowHTTPS	Boolean
DomainLimit	String
GlobList	String
GlobListType	String
LinkDepth	Number
MIMEList	String
MIMEListType	String
ProxyHost	String
ProxyPort	Number
StartOpts	Search Start Options collection RO

SearchFSOptions Properties

Property	Returns
GlobList	String
GlobListType	String
MIMEList	String
MIMEListType	String
StartOpts	Search Start Options collection RO

Copyright © 1988-2007, Oracle. All rights reserved.

SearchStartOptions Collection Methods

Method	Returns
DeleteItem(*Value*)	Boolean
First()	Search Start Options object
InsertItem(*Value*)	Search Start Options object
ItemByName(*Value*)	Search Start Options object
Next()	Search Start Options object

SearchStartOptions Collection Property

Property	Returns
Count	Number [RO]

SearchStartOptions Properties

Property	Returns
IsDomainRestricted	Boolean
IsHostRestricted	Boolean
Value	String [RO]

PortalRegistry Classes Methods and Properties

This section lists the methods, and properties, as well as returns (if applicable), for the PortalRegistry classes.

PortalRegistry Methods

Method	Returns
BuildSearchIndex(*Language*)	Boolean
Close()	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
CopyObject(*sourcePortalName*, *sourceRefType*, *sourceObjName*, *targetPortalName*, *targetPrntFldrName*, *copyChildren*)	Boolean
Create(*RegistryName*)	Boolean
CreateContentRefLink(*LinkName*, *LinkLabel*, *LinkParent*, *CRefPortalName*, *CRefObjectName*)	CRefLink object
CreateRemote(*PortalName*, *RemoteNodeName*)	Boolean
Delete(*RegistryName*)	Boolean
DeleteHomepage()	Boolean
FindCRefByName(*Name*)	ContentReference object
FindCRefByURL(*URL*)	ContentReference object
FindCrefForURL(*URL*)	ContentReference object
FindCRefLinkByName(*LinkName*)	Content Reference Link object
FindFolderByName(*Name*)	Folder object
FindPgltByName(*PageletName*)	Pagelet object
GetAbsoluteContentURL(*NodeName*, *URL*)	ContentReference object
GetDefaultHPTabOID()	String
GetQualifiedURL(*ContentProvider*, *RelativeURL*)	URL string
GetSearchQuery()	SearchQuery object
GrantPermissionForComponent(*MenuName*, *ComponentName*, *Market*, *PermListName*, *NodeName*)	Boolean
GrantPermissionForScript(*RecordName*, *FieldName*, *EventName*, *FuncName*, *PermListName*, *NodeName*)	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
Open(*RegistryName*)	Boolean
PermissionListDelete(*PermListName*)	Boolean
PermissionListSaveAs(*PermListSourceName*, *PermListTargetName*)	Boolean
RevokePermissionForComponent(*MenuName*, *ComponentName*, *Market*, *PermListName*, *NodeName*)	Boolean
RevokePermissionForScript(*RecordName*, *FieldName*, *EventName*, *FuncName*, *PermListName*, *NodeName*)	Boolean
Save()	Boolean

PortalRegistry Properties

Property	Returns
DefaultTemplate	String
Description	String
DisableFolderNavigation	Boolean
Favorites	Favorite Collection RO
FolderNavObject	String
Homepage	Homepage object RO
IsFolderNavigation	Booldan
Name	String RO
NodeTemplates	NodeTemplate Collection RO
OwnerId	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
PageletCategories	PageletCategories Collection RO
Portals	Portal collection RO
RootFolder	Folder object RO
TabDefinitions	TabDefinitions Collection RO
TemplateObject	ContentReference object RO

PortalRegistry Collection Methods

Method	Returns
First()	PortalRegistry object
Item(*number*)	PortalRegistry object
Next()	PortalRegistry object

PortalRegistry Property

Property	Returns
Count	Number RO

Node Properties

Property	Returns
ActiveNode	Boolean RO
AppRelease	String RO
ContentURI	String RO
DefaultPortalName	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Description	String RO
Name	String RO
NodePassword	String RO
NodeType	String RO
PortalURI	String RO
ToolsRelease	String RO

Node Collection Methods

Method	Returns
First()	Node object
ItemByName(*NodeName*)	Node object
Next()	Node object

Node Collection Property

Property	Returns
Count	Number RO

Remote Node Collection Methods

Method	Returns
First()	Node object
ItemByName(*NodeName*)	Node object
Next()	Node object

Copyright © 1988-2007, Oracle. All rights reserved.

Remote Node Collection Property

Property	Returns
Count	Number [RO]

Portal Method

Method	Returns
Save()	Boolean

Portal Properties

Property	Returns
HostNodeName	String
IsLocal	Boolean [RO]
Name	String [RO]

Portal Collection Methods

Method	Returns
First()	Portal object
ItemByName(*PortalName*)	Portal object
Next()	Portal object

Portal Collection Property

Property	Returns
Count	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Node Template Properties

Property	Returns
DefaultTemplate	String
Name	String
TemplateObject	Template object [RO]

Node Template Collection Methods

Method	Returns
DeleteItem(*NodeName*)	Boolean
First()	NodeTemplate object
InsertItem(*NodeName*)	NodeTemplate object
ItemByName(*NodeName*)	NodeTemplate object
Next()	NodeTemplate Object

Node Template Collection Property

Property	Returns
Count	Number [RO]

Folder Method

Method	Returns
Save()	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Folder Properties

Property	Returns
Attributes	Attribute collection object [RO]
Author	String [RO]
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionValue collection [RO]
CascadedRolePermissions	RolePermissionValue collection [RO]
ContentRefs	ContentReference Collection [RO]
CreationDate	String [RO]
Description	String
Folders	Folder collection [RO]
IsMobile	Boolean
IsVisible	Boolean [RO]
Label	String
Name	String [RO]
OwnerID	String
ParentName	String [RO]
Path	String [RO]
Permissions	PermissionValue collection [RO]
Product	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
RolePermissions	RolePermissionValue collection [RO]
SequenceNumber	Number
PublicAccess	Boolean
ValidFrom	String
ValidTo	String

Folder Collection Methods

Method	Returns
DeleteItem(*FolderName*)	Boolean
First()	Folder object
InsertItem(*FolderName*, *Label*)	Folder object
ItemByName(*FolderName*)	Folder object
Next()	Folder object

Folder Collection Property

Property	Returns
Count	Number [RO]

ContentReference Methods

Methods	Returns
CreateLink(*Name*, *Label*)	Content Reference Link object
Save()	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

ContentReference Properties

Property	Returns
AbsoluteContentURL	String [RO]
AbsolutePortalURL	String [RO]
AssignedPagelets	AssignedPagelets collection [RO]
Attributes	Attribute collection object [RO]
Author	String [RO]
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionValue collection [RO]
CascadedRolePermissions	RolePermissionValue collection [RO]
ContentProvider	String
CreationDate	String [RO]
Data	String
Description	String
HtmlText	String [RO]
IsMobile	Boolean
IsVisible	Boolean [RO]
Label	String
Links	Link collection [RO]
Name	String [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
OwnerId	String
ParentName	String [RO]
Path	String [RO]
Permissions	PermissionValue collection [RO]
Product	String
PublicAccess	Boolean
RelativeURL	String [RO]
RolePermissions	RolePermissionValue collection [RO]
SequenceNumber	Number
StorageType	String
Template	String
TemplateObject	ContentReference object [RO]
TemplateType	String
URL	String
URLType	String
UsageType	String
ValidFrom	String
ValidTo	String

Copyright © 1988-2007, Oracle. All rights reserved.

ContentReference Collection Methods

Method	Returns
DeleteItem(*ContentReferenceName*)	Boolean
First()	ContentReference object
InsertItem(*ContentReferenceName*, *ContentReferenceLabel, Node, URL*)	ContentReference object
ItemByName(*ContentReferenceName*)	ContentReference object
Next()	ContentReference object

ContentReference Collection Property

Property	Returns
Count	Number [RO]

AttributeValue Properties

Property	Returns
Label	String
Name	String [RO]
Translatable	Boolean
Value	String

AttributeValue Collection Methods

Method	Returns
DeleteItem(*AttributeValueName*)	Boolean
First()	AttributeValue object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
InsertItem(*AttributeValueName*)	AttributeValue object
ItemByName(*AttributeValueName*)	AttributeValue object
Next()	AttributeValue object

AttributeValue Collection Property

Property	Returns
Count	Number [RO]

PermissionValue Properties

Property	Returns
Cascade	Boolean
Name	String

PermissionValue Collection Methods

Method	Returns
DeleteItem(*PermissionValueName*)	Boolean
First()	PermissionValue object
InsertItem(*PermissionValueName*)	PermissionValue object
ItemByName(*PermissionValueName*)	PermissionValue object
Next()	PermissionValue object

Copyright © 1988-2007, Oracle. All rights reserved.

PermissionValue Collection Property

Property	Returns
Count	Number [RO]

RolePermissionValue Collection Methods

Method	Returns
DeleteItem(*PermissionValueName*)	Boolean
First()	PermissionValue object
InsertItem(*PermissionValueName*)	PermissionValue object
ItemByName(*PermissionValueName*)	PermissionValue object
Next()	PermissionValue object

RolePermissionValue Collection Property

Property	Returns
Count	Number [RO]

ContentReference Link Methods

Method	Returns
Delete()	Boolean
Save()	Boolean

Content Reference Link Properties

Property	Returns
AbsoluteContentURL	String [RO]
AbsolutePortalURL	String [RO]
Attributes	Attribute Collection
Author	String [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
AuthorAccess	Boolean RO
Authorized	Boolean RO
CascadedPermissions	PermissionValue collection RO
ContentProvider	String RO
CreationDate	String RO
Data	String
Description	String
IsMobile	Boolean
IsVisible	Boolean RO
Label	String
Name	String
OwnerId	String
ParentName	String
Path	String RO
Permissions	PermissionValue Collection RO
Product	String
PublicAccess	Boolean RO
RelativeURL	String RO
SequenceNumber	String
Template	String
TemplateObject	String RO
TemplateType	String
URL	String
URLType	String RO
ValidFrom	Date
ValidTo	Date

Copyright © 1988-2007, Oracle. All rights reserved.

Link Collection Methods

Method	Returns
First	Link object
Next	Link object

Link Collection Property

Property	Returns
Count	Number [RO]

Link Class Properties

Property	Returns
LinksObjectName	String [RO]
LinksObjectType	String [RO]
LinksPortalName	String [RO]

TabDefinition Method

Method	Returns
Save()	Boolean

TabDefinition Properties

Property	Returns
AssignedPagelets	AssignedPagelets collection [RO]
AvailableCategories	AvailableCategories collection [RO]
AvailablePagelets	AvailablePagelets collection [RO]
Attributes	AttributeValue collection object [RO]
Author	String [RO]
AuthorAccess	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Authorized	Boolean
ColumnLayout	String
CreationDate	String RO
Description	String
DynamicCategories	DynamicCategories collection RO
HelpID	String
HtmlText	String RO
IsHideActionBar	Boolean
IsLayoutLocked	Boolean
IsRenamable	Boolean
Label	String
LayoutBehavior	String
Name	String RO
OwnerId	String
Product	String
PublicAccess	Boolean
QualifiedURL	String RO
SequenceNumber	Number
StyleSheet	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ValidFrom	String
ValidTo	String

TabDefinition Collection Methods

Method	Returns
DeleteItem(*TabDefinitionName*)	Boolean
First()	TabDefinition object
InsertItem(*TabDefinitionName*)	TabDefinition object
ItemByName(*TabDefinitionName*)	TabDefinition object
Next()	TabDefinition object

TabDefinition Collection Property

Property	Returns
Count	Number [RO]

AssignedPagelet Properties

Property	Returns
Column	Number
LayoutBehavior	String
PageletName	String [RO]
Row	Number

Copyright © 1988-2007, Oracle. All rights reserved.

AssignedPagelet Collection Methods

Method	Returns
DeleteItem(*PageletName*)	Boolean
First()	Pagelet object
InsertItem(*PageletName*, *Column*, *Row*, *LayoutBehavior*)	Pagelet object
ItemByName(*PageletName*)	Pagelet object
Next()	Pagelet object

AssignedPagelet Collection Property

Property	Returns
Count	Number [RO]

AvailableCategory Properties

Property	Returns
AvailablePagelets	AvailablePagelets Collection [RO]
CategoryName	String [RO]

AvailableCategory Collection Methods

Method	Returns
First()	AvailableCategory object
ItemByName(*Name*)	AvailableCategory object
Next()	AvailableCategory object

Copyright © 1988-2007, Oracle. All rights reserved.

AvailableCategory Collection Property

Property	Returns
Count	Number [RO]

AvailablePagelet Properties

Property	Returns
CategoryLabel	String [RO]
CategoryName	String [RO]
Column	Number [RO]
LayoutBehavior	String [RO]
PageletLabel	String [RO]
PageletName	String [RO]
Row	Number [RO]

AvailablePagelet Collection Methods

Method	Returns
First()	AvailablePagelet object
ItemByName(*Name*)	AvailablePagelet object
Next()	AvailablePagelet object

AvailablePagelet Collection Property

Property	Returns
Count	Number [RO]

DynamicCategory Collection Methods

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
DeleteItem(*Name*)	Boolean
First()	String
InsertItem(*Name*)	String
ItemByName(*Name*)	String
Next()	String

DynamicCategory Collection Property

Property	Returns
Count	Number [RO]

PageletCategory Method

Method	Returns
Save()	Boolean

PageletCategory Properties

Property	Returns
Attributes	AttributeValues collection object [RO]
Author	String [RO]
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionList object [RO]
CreationDate	String [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Description	String
Label	String
Name	String RO
OwnerId	String
Pagelets	Pagelets Collection RO
Permissions	PermissionList Collection RO
Product	String
PublicAccess	Boolean
SequenceNumber	Number

PageletCategory Collection Methods

Method	Returns
DeleteItem(*Name*)	Boolean
First()	String
InsertItem(*Name*, *Label*)	String
ItemByName(*Name*)	String
Next()	String

PageletCategory Collection Property

Property	Returns
Count	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

Pagelet Method

Method	Returns
Save()	Boolean

Pagelet Properties

Property	Returns
Attributes	Attribute collection object RO
Author	String RO
AuthorAccess	Boolean
Authorized	Boolean
CascadedPermissions	PermissionList Collection RO
ContentProvider	Node object RO
CreationDate	String RO
DefaultColumn	Number
Description	String
HelpID	String
IsHideMinimize	Boolean
Label	String
Name	String RO
OwnerId	String
ParentName	String RO
Permissions	PermissionList Collection RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Product	String
PublicAccess	Boolean
QualifiedURL	String [RO]
SequenceNumber	Number
URL	String
URLType	String

Pagelet Collection Methods

Method	Returns
DeleteItem(*Name*)	Boolean
First()	Pagelet object
InsertItem(*Name*, *Label*, *NodeName*, *URL*)	Pagelet object
ItemByName(*Name*)	Pagelet object
Next()	Pagelet object

Pagelet Collection Property

Property	Returns
Count	Number [RO]

UserHomepage Method

Method	Returns
Save()	

Copyright © 1988-2007, Oracle. All rights reserved.

UserHomepage Properties

Property	Returns
Greeting	String
UserId	String [RO]
UserTab	UserTab Collection[RO]

UserTab Properties

Property	Returns
ColumnLayout	Number
Label	String
QualifiedURL	String [RO]
SelectedPagelets	SelectedPagelets collection[RO]
SequenceNumber	Number
TabName	String [RO]

UserTab Collection Methods

Method	Returns
DeleteItem(*Name*)	Boolean
First()	UserTab object
InsertItem(*Name*)	UserTab object
ItemByName(*Name*)	UserTab object
Next()	UserTab object

Copyright © 1988-2007, Oracle. All rights reserved.

UserTab Collection Property

Property	Returns
Count	Number [RO]

SelectedPagelet Properties

Property	Returns
CategoryName	String [RO]
Column	Number
IsMinimized	Boolean
PageletName	String [RO]
Row	Number

SelectedPagelet Collection Methods

Method	Returns
DeleteItem(*Name*)	Boolean
First()	SelectedPagelet object
InsertItem(*Name*)	SelectedPagelet object
ItemByName(*Name*)	SelectedPagelet object
Next()	SelectedPagelet object

SelectedPagelet Collection Property

Property	Returns
Count	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Favorites Properties

Property	Returns
CRefName	String [RO]
Label	String
QualifiedURL	String [RO]
SequenceNumber	Number
URL	String

Favorites Collection Methods

Method	Returns
DeleteItem(*FavoriteLabel*)	Boolean
First()	Favorite object
InsertItem(*FavoriteLabel*, *FavoriteName*)	Favorite object
ItemByLabel(*FavoriteLabel*)	Favorite object
Next()	Favorite object

Favorites Collection Property

Property	Returns
Count	Number [RO]

Query Classes Methods and Properties

This section lists the functions, methods, and properties, as well as returns (if applicable), for the query classes.

Copyright © 1988-2007, Oracle. All rights reserved.

Query Collection Methods

Method	Returns
First()	Query object
Item(*Number*)	Query object
ItemByName(*Name*)	Query object
Next()	Query object

Query Collection Property

Property	Returns
Count	Number [RO]

Query Methods

Method	Returns
AddQuerySelect()	QuerySelect object
AddPrompt(*PromptName*)	Prompt object
Close()	None
Create(*QueryName*, *Pubic*, *Type*, *Description*, *LongDescription*)	Query object
Delete()	Number
DeletePrompt(*PromptName*)	Number
FindExpression(*Number*)	Expression object
FormatResultString(*&Rowset*, *Output_Format*, *StartRow*, *EndRow*)	String
Open(*QueryName*, *Public*, *Update*)	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
Rename(*NewQueryName*)	Number
RunToFile(*&PromptRecord*, *Destination*, *OutputFormat*, *MaxRows*)	Number
RunToRowset(*&PromptRecord*, *MaxRows*)	Rowset object
Save()	Number

Query Properties

Property	Returns
Approved	String
ApprovedDtTm	String [RO]
ApprovedUserId	String
CreateDtTm	String [RO]
CreateUserId	String [RO]
Description	String
ExecAppName	String
ExecLogging	Boolean
LastSQLErrorCode	Number [RO]
LastUpdDttm	String [RO]
LastUpdOprId	String [RO]
LongDescription	String
Metadata	Metadata collection [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Name	String RO
PDFFont	String
Prompts	QueryPrompt collection RO
PromptRecord	Record object RO
PublicPrivate	String
QuerySelect	QuerySelect object RO
QueryStatistics	QueryStatistics object RO
RunTimePrompts	QueryPrompt Collection RO
SQL	String RO
Type	Number

QuerySelect Methods

Method	Returns
AddAllFields(*QueryRecord*)	Number
AddCriteria(*Name*)	QueryCriteria object
AddExpression(*Name*)	QueryExpression object
AddHavingCriteria(*Name*)	QueryCriteria object
AddQueryOutputField(*QueryRecord*, *index*)	QueryOutputField object
AddQueryRecord(*QueryRecordName*)	QueryRecord object
AddQuerySelectedField(*QueryRecord*, *index*)	QuerySelectedField object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
DeleteCriteria(*index*)	Number
DeleteExpression(*Index*)	Number
DeleteField(*Index)*	Number
DeleteHavingCriteria(*Index*)	Number
DeleteRecord(*Index*)	Number

QuerySelect Properties

Property	Returns
Criteria	QueryCriteria collection RO
Distinct	Boolean
Expressions	QueryExpression collection RO
HavingCriteria	QueryCriteria collection RO
ParentSelectNum	Number RO
QueryOutputFields	QueryField collection RO
QueryRecords	QueryRecord collection RO
QuerySelectedFields	QueryField collection RO
QuerySelects	QuerySelect object RO
SelectNum	Number RO
SelectType	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

QuerySelect Collection Methods

Method	Returns
AddUnion()	QuerySelect object
DeleteQuerySelect(*SelNumber*)	Number
First()	QuerySelect object
Item(*Number*)	QuerySelect object
ItemBySelNum(*SelNumber*)	QuerySelect object
Next()	QuerySelect object

QuerySelect Collection Properties

Property	Returns
Count	Number [RO]

QueryRecord Collection Methods

Method	Returns
First()	QueryRecord object
Item(*Number*)	QueryRecord object
ItemByAlias(*Alias*)	QueryRecord object
Next()	QueryRecord object

QueryRecord Collection Property

Property	Returns
Count	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

QueryRecord Method

Method	Returns
GetField(*Index*)	QueryField object

QueryRecord Properties

Property	Returns
Description	String [RO]
JoinAlias	String
JoinFieldName	String
JoinType	Number
Name	String [RO]
QueryFields	QueryFields collection [RO]
RecordAlias	String

QueryField Collection Methods

Method	Returns
First()	QueryField object
Item(*Number*)	QueryField object
ItemByNameAndAlias(*Name*, *RecordAlias*)	QueryField object
ItemByExpNum(*ExpNum*)	QueryField object
Next()	QueryField object

Copyright © 1988-2007, Oracle. All rights reserved.

QueryField Collection Property

Property	Returns
Count	Number [RO]

QueryField Methods

Method	Returns
AddTranslateExpression(*ExpressionName*)	QueryExpression object
AddTranslateField(*FieldName*)	QueryField object

QueryField Properties

Property	Returns
Aggregate	Number
ColumnNumber	Number
Description	String [RO]
Decimal	Number [RO]
ExpNum	Number
Flag	Number [RO]
Format	Number
HeadingText	String
HeadingType	String
HeadingUniqueFieldName	String
Length	Number [RO]
LongName	String [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Name	String RO
OrderByDirection	Number
OrderByNumber	Number
QueryRecord	QueryRecord object RO
RecordAlias	String RO
ShortName	String RO
TranslateEffDtLogic	String
TranslateExpression	Expression object RO
TranslateField	QueryField RO
TranslateOption	String
Type	String

QueryCriteria Collection Methods

Method	Returns
First()	QueryCriteria object
Item(*Number*)	QueryCriteria object
ItemByName(*CriteriaName*)	QueryCriteria object
Next()	QueryCriteria object

Copyright © 1988-2007, Oracle. All rights reserved.

QueryCriteria Collection Property

Property	Returns
Count	Number [RO]

QueryCriteria Methods

Method	Returns
AddExpr1Expression()	QueryExpression object
AddExpr1Field(*QueryRecordAlias*, *FieldName*)	QueryField object
AddExpr2Expression()	QueryExpression object
AddExpr2Field1(*QueryRecordAlias*, *FieldName*)	QueryField object
AddExpr2Field2(*QueryRecordAlias*, *FieldName*)	QueryField object
AddExpr2List()	QueryList object
AddExpr2Subquery()	QuerySelect object

QueryCriteria Properties

Property	Returns
Expr1Expression	QueryExpression object [RO]
Expr1Field	QueryField object [RO]
Expr1Type	Number
Expr2Constant1	String
Expr2Constant2	String
Expr2Expression1	QueryExpression object [RO]
Expr2Expression2	QueryExpression object [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Expr2Field1	QueryField object RO
Expr2Field2	QueryField object RO
Expr2List	QueryList object RO
Expr2Subquery	QuerySelect object RO
Expr2Type	Number
Logical	Number
LParenLvl	Number
Name	String RO
Negation	Boolean RO
Operator	Number
R1ExprNum	Number
R2ExprNum	Number
R1ExprType	Number
R2ExprType	Number
RParenLvl	Number

QueryExpression Collection Methods

Method	Returns
First()	QueryExpression object
Item(*Number*)	QueryExpression object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
ItemByName(*ExpressionName*)	QueryExpression object
Next()	QueryExpression object

QueryExpression Collection Property

Property	Returns
Count	Number [RO]

QueryExpression Properties

Property	Returns
Aggregate	Number
BindFlag	Number
Decimal	Number
ExpNum	Number
Length	Number
Name	String [RO]
OutputField	QueryField object
RightExprFlag	Number
SelectedField	QueryField object
Text	String
Type	Number

Copyright © 1988-2007, Oracle. All rights reserved.

QueryList Methods

Method	Returns
AddListValue(*Value*, *IsPrompt*)	QueryListValue object
First()	QueryListValue object
Item(*Number*)	QueryListValue object
Next()	QueryListValue object

QueryList Property

Property	Returns
Count	Number [RO]

QueryListValue Properties

Property	Returns
IsPrompt	Boolean [RO]
Value	String [RO]

QueryRecordHierarchy Collection Methods

Method	Returns
First()	QueryRecordHierarchy object
Item(*Number*)	QueryRecordHierarchy object
ItemByName(*Name*)	QueryRecordHierarchy object
Next()	QueryRecordHierarchy object

Copyright © 1988-2007, Oracle. All rights reserved.

QueryRecordHierarchy Collection Property

Property	Returns
Count	Number [RO]

QueryRecordHierarchy Properties

Property	Returns
Description	String [RO]
Level	Number [RO]
Name	String [RO]
ParentFlag	Number [RO]

Query Metadata Collection Methods

Method	Returns
First()	Query Metadata object
Item(*Number*)	Query Metadata object
ItemByName(*Name*)	Query Metadata object
Next()	Query Metadata object

Query Metadata Collection Property

Property	Returns
Count	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Query Metadata Properties

Property	Returns
Name	String RO
Value	String RO

QueryStatistics Properties

Property	Returns
AvgExecTime	String RO
AvgFetchTime	String RO
AvgNumRows	Number RO
ExecCount	Number RO
LastExecDtTm	String RO

QuerySecurityProfile Properties

Property	Returns
AllowAnyJoin	Boolean RO
AllowDistinct	Boolean RO
AllowExpressions	Boolean RO
AllowSubqueries	Boolean RO
AllowUnions	Boolean RO
ApprovePrivateQuery	Boolean RO
ApprovePublicQuery	Boolean RO
CanCreatePublic	Boolean RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
CanCreateWorkFlow	Boolean RO
CanModifyQuery	Boolean RO
CanRunQuery	Boolean RO
CanRunToCrystal	Boolean RO
CanRunToExcel	Boolean RO
LimitUnapproved	Boolean RO
MaxInTreeCriteria	Boolean RO
MaxJoins	Boolean RO
MaxRowsToFetch	Boolean RO
MaxUnapprovedRows	Boolean RO

QueryDBRecord Collection Methods

Method	Returns
First()	QueryDBRecord object
Item(*Number*)	QueryDBRecord object
ItemByName(*Name*)	QueryDBRecord object
Next()	QueryDBRecord object

QueryDBRecord Collection Property

Property	Returns
Count	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

QueryDBRecord Methods

Method	Returns
QueryDBRecordFieldByIndex(*Index*)	QueryDBRecordField object
QueryDBRecordFieldByName(*Name*)	QueryDBRecordField object

QueryDBRecord Properties

Property	Returns
Description	String RO
Name	String RO
QueryDBRecordFields	QueryDBRecordFields collection
RecordHierarchy	QueryRecordHierarchy Collection RO

QueryDBRecordField Collection Methods

Method	Returns
First()	QueryDBRecordField object
Item(*Number*)	QueryDBRecordField object
ItemByName(*Name*)	QueryDBRecordField object
Next()	QueryDBRecordField object
Sort(*SortCriteria*)	None

QueryDBRecordField Collection Property

Property	Returns
Count	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

QueryDBRecordField Properties

Property	Returns
Decimal	Number [RO]
Description	String [RO]
Flag	Number [RO]
Format	Number [RO]
Length	Number [RO]
LongName	String [RO]
LookupTableName	String [RO]
LookupTableRecord	QueryDBRecord object [RO]
Name	String [RO]
ShortName	String [RO]
Type	Number [RO]

QueryPrompt Collection Methods

Method	Returns
First()	QueryPrompt object
Item(*Number*)	QueryPrompt object
ItemByName(*Name*)	QueryPrompt object
Next()	QueryPrompt object

Copyright © 1988-2007, Oracle. All rights reserved.

QueryPrompt Collection Property

Property	Returns
Count	Number [RO]

QueryPrompt Properties

Property	Returns
EditType	Number
FieldDecimal	Number
FieldFormat	Number
FieldLength	Number
FieldName	String
FieldType	Number
HeadingText	String
HeadingType	Number
LangCount	Number [RO]
Name	String [RO]
PromptRecordFieldName	String [RO]
PromptTable	String
UniquePromptName	String
UseCount	Number [RO]

Tree Classes Methods and Properties

This section lists the methods and properties, as well as returns (if applicable), for the tree classes.

Copyright © 1988-2007, Oracle. All rights reserved.

Branch Collection Properties

Property	Returns
Count	Number [RO]

Leaf Methods

Method	Returns
Cut()	Number
Delete()	Number
DeleteByRange(*RangeFrom*, *RangeTo*)	Number
InsertDynSib()	Number
InsertSib(*RangeFrom*, *RangeTo*)	Leaf object
MoveAsChild(*Node*)	Number
MoveAsChildByName(*NodeName*)	Number
MoveAsSib(*Leaf*)	Number
MoveAsSibByRange(*RangeFrom*, *RangeTo*)	Number
PasteSib()	Number
UpdateRanges(*RangeFrom*, *RangeTo*)	Number

Leaf Properties

Property	Returns
DisplayLevelNumber	Number [RO]
Dynamic	Boolean
HasNextSib	Boolean [RO]
HasPrevSib	Boolean [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ImageName	String
IsChanged	Boolean RO
IsCut	Boolean RO
IsDeleted	Boolean RO
IsInserted	Boolean RO
NextSib	Leaf object RO
Parent	Node object RO
PrevSib	Leaf object RO
RangeFrom	String
RangeTo	String
TreeBranchName	String RO
TreeEffDt	Date
TreeName	String RO
TreeSetID	String RO
TreeUserKeyValue	String RO

Level Collection Methods

Method	Returns
Add(*LevelName*)	Level object
Item*(LevelName*, *LevelNumber*)	Level object
Remove()	Number

Level Collection Properties

Property	Returns
Count	Number RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
First	Level object RO
Last	Level object RO
Next	Level object RO

Level Properties

Property	Returns
AllValuesAudit	Boolean
Description	String
Name	String
Number	Number
TreeBranchName	String RO
TreeEffDt	Date
TreeName	String RO
TreeSetID	String RO
TreeUserKeyValue	String RO

Node Methods

Method	Returns
Branch()	Number
Cut()	Number
Delete()	Number
DeleteByName(*NodeName*)	Number
Expand(*ExpandType*)	Number
InsertChildLeaf(*RangeFrom*, *RangeTo*)	Leaf object

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
InsertChildNode(*NodeName*)	Node object
InsertChildRecord(*NodeName*)	Node object
InsertDynChildLeaf()	Leaf object
InsertSib(*NodeName*)	Node object
InsertSibRecord(*NodeName*)	Node object
MoveAsChild(*Node*)	Number
MoveAsChildByName(*NodeName*)	Number
MoveAsSib(*Node*)	Number
MoveAsSibByName(*NodeName*)	Number
PasteChild()	Number
PasteSib()	Number
RefreshDescription()	Number
Rename(*NodeName*)	Number
SwitchLevel(*NewLevelNumber*)	Number
Unbranch()	Number

Node Properties

Property	Returns
AllChildCount	Number [RO]
AllChildNodeCount	Number [RO]
ChildLeafCount	Number [RO]
ChildNodeCount	Number [RO]

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
ColImageName	String
Description	String RO
DisplayLevelNumber	Number RO
ExpImageName	String
FirstChildLeaf	Leaf object RO
FirstChildNode	Node object RO
HasChildLeaves	Boolean RO
HasChildNodes	Boolean RO
HasChildren	Boolean RO
HasNextSib	Boolean RO
HasPrevSib	Boolean RO
IsBranched	Boolean RO
IsChanged	Boolean RO
IsCut	Boolean RO
IsDeleted	Boolean RO
IsInserted	Boolean RO
IsRoot	Boolean RO
LastChildLeaf	Leaf object RO
LastChildNode	Node object RO
LevelNumber	Number
Name	String
NextSib	Leaf object RO
Parent	Node object RO
PrevSib	Leaf object RO
State	Number RO
TreeBranchName	String RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
TreeEffDt	Date
TreeName	String RO
TreeSetID	String RO
TreeUserKeyValue	String RO
Type	String RO

Tree Methods

Method	Returns
Audit()	Number
AuditByName(*SetID, UserKeyValue, TreeName, EffDt, BranchName*)	Number
Close()	Number
Copy(*FromSetId, FromUserKeyValue, FromTreeName, FromEffDt, FromBranchName, ToSetId, ToUserKeyValue, ToTreeName, ToEffDt, ToBranchName*)	Number
Create(*SetID, UserKeyValue, TreeName, EffDt, StructureName*)	Number
Delete(*SetID, UserKeyValue, TreeName, EffDt, BranchName*)	Number
Exists(*SetID, UserKeyValue, TreeName, EffDt, BranchName*)	Number
FindLeaf(*RangeFrom*, *RangeTo*)	Leaf object
FindNode(*NodeName, Description*)	Node object
FindRoot()	Node object
InsertRoot(*NodeName*)	Node object
LeafExists(*RangeFrom, RangeTo*)	Number

 Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
LockTree()	Number
Open(*SetID, UserKeyValue, TreeName, EffDt, BranchName, Update*)	Number
OpenForExport(*SetID*, *UserKeyValue*, *TreeName EffDt*)	Number.
OpenWholeTree(*SetID, UserKeyValue, TreeName, EffDt*)	Number
NodeExists(*NodeName*)	Number
Rename(*FromSetId, FromUserKeyValue, FromTreeName, FromEffDt, FromBranchName, ToTreeName*)	Number
Save()	Number
SaveAs(*SetID, UserKeyValue, TreeName, EffDt, BranchName*)	Number
SaveAsDraft(*SetID, UserKeyValue, TreeName, EffDt, BranchName*)	Number
SaveDraft()	Number
SetImportMode(*Nodes_Number*, *Leaves_Number*)	Number
TreeLocksNumber(*setID*, *UserKeyValue*, *TreeName*, *EffDt*)	Number
UnlockTree()	Number
UpdateLock()	Number

Tree Properties

Property	Returns
AllValues	Boolean
AuditDetails	Boolean

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
Branches	Branch collection object RO
BranchImageName	String
BranchLevel	Number RO
BranchName	String RO
Category	String
Description	String
DuplicateLeaves	Boolean
EffDt	Date
HasDetailRanges	Boolean RO
HasLockedBranches	Boolean RO
IsBranched	Boolean RO
IsChanged	Boolean RO
IsOpen	Boolean RO
IsQueryTree	Boolean RO
IsValid	Boolean RO
IsVersionChanged	Boolean RO
IsWholeTree	Boolean RO
KeyBranchName	String RO
KeyEffDt	String RO
KeyName	String RO
KeySetID	String RO
KeyUserKeyValue	String RO
LeafCount	Number RO
LeafImageName	String
LeafOnClipboard	Boolean RO
LevelCount	Number RO
Levels	Level collection object RO

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
LevelUse	String
LockOwner	String, RO
Name	String
NodeColImage	String
NodeCount	Number RO
NodeExpImage	String
NodeOnClipboard	Boolean RO
ParentLevel	Number RO
ParentName	String RO
PerformanceMethod	String
PerformanceSelector	String
PerformanceSelectorOption	String
SetId	String
Status	String
Structure	Tree structure object RO
StructureName	String RO
TreeImageName	String
UserKeyValue	String
UseUpdateReservation	Boolean RO

Tree Structure Methods

Method	Returns
Close()	Number
Copy(*FromStructId, ToStructId*)	Number
Create(*StructId, Type*)	Number

Copyright © 1988-2007, Oracle. All rights reserved.

Method	Returns
Delete(*StructId*)	Number
Open(*StructId, Update*)	Number
Rename(*FromStructId, ToStructId*)	Number
Save()	Number

Tree Structure Properties

Property	Returns
Description	String
DetailComponent	String
DetailField	String
DetailMenu	String
DetailMenuBar	String
DetailMenuItem	String
DetailMultiNavigate	Boolean
DetailPage	String
DetailRecord	String
IndirectionMethod	String
KeyName	String RO
LevelComponent	String
LevellMenu	String
LevelMenuBar	String
LevelMenuItem	String
LevelPage	String
LevelRecord	String
Name	String
NodeComponent	String

Copyright © 1988-2007, Oracle. All rights reserved.

Property	Returns
NodeField	String
NodelMenu	String
NodeMenuBar	String
NodeMenuItem	String
NodeMultiNavigate	Boolean
NodePage	String
NodeRecord	String
NodeUserKeyField	String
SummarySetId	String
SummaryLevelNumber	String
SummaryTreeName	String
SummaryUserKeyValue	String
Type	String

Mapping of Functions to Methods and Properties

The first column in the following table lists built-in PeopleCode functions that existed prior to PeopleTools 8.0. The second column lists the new method or property that should be used instead of the function. The existing functions still work in PeopleTools 8.0, however, they are being retained for backward compatibility only. New applications should be created using the new classes, methods, and properties.

Existing function name	Use instead
ActiveRowCount	ActiveRowCount Rowset property
ClearSearchDefault	SearchDefault Field property
ClearSearchEdit	SearchEdit Field property
CompareLikeFields	CompareFields Record method
CopyFields	CopyFieldsTo or CopyChangedFieldsTo Record method

Copyright © 1988-2007, Oracle. All rights reserved.

Existing function name	Use instead
CopyRow	CopyTo Row method
CurrEffDt	DeleteEnabled Rowset property
CurrEffRowNum	RowNumber Row property, in combination with the GetCurrEffRow Rowset method
CurrEffSeq	EffSeq Rowset property
CurrentRowNumber	RowNumber Row property
DeleteRecord	Delete Record method
DeleteRow	DeleteRow Rowset method
FetchValue	Value Field property
FieldChanged	IsChanged Field property
GetRelField	GetRelated Field method
GetStoredFormat	StoredFormat Field property
Gray	Enabled Field property
Hide	Visible Field property
HideRow	Visible Row property
HideScroll	HideAllRows Rowset method
InsertRow	InsertRow Rowset method
IsHidden	Visible Row property
NextEffDt	GetNextEffRow().REC.FIELD.Value
NextRelEffDt	GetNextEffRow.()REC.FIELD.GetRelated (rec.field).value

Copyright © 1988-2007, Oracle. All rights reserved.

Existing function name	Use instead
PriorEffDt	GetPriorEffRow().REC.Field.Value
PriorRelEffDt	GetPriorEffRow().REC.FIELD.GetRelated (rec.field).value
RecordChanged	IsChanged Record property
RecordDeleted	IsDeleted Record property
RecordNew	IsNewRecord property
RemoteCall for Application Engine	CallAppEngine function
RowFlush	FlushRow Rowset method
RowScrollSelect	Select Rowset method
RowScrollSelectNew	SelectNew Rowset method
ScrollFlush	Flush Rowset method
ScrollSelect	Select Rowset method
ScrollSelectNew	SelectNew Rowset method
SetDefault	SearchDefault Field property
SetDefaultAll	SetDefault Rowset method
SetDefaultNext	GetNextEffRow().REC.FIELD.SetDefault()
SetDefaultNextRel	GetNextEffRow().REC.Field.GetRelated (REC.FIELD).SetDefault()
SetDefaultPrior	GetPriorEffRow().REC.FIELD.SetDefault()
SetDefaultPriorRel	GetPriorEffRow().REC.Field.GetRelated (REC.FIELD).SetDefault()

Copyright © 1988-2007, Oracle. All rights reserved.

Existing function name	Use instead
SetDisplayFormat	DisplayFormat Field property
SetLabel	Label Field property
SetSearchDefault	SearchDefault Field property
SetSearchEdit	SearchEdit Field property
SetTracePC	If using API (Session object), use Trace Setting Class properties.
SetTraceSQL	If using API (Session object), use Trace Setting Class properties.
SortScroll	Sort Rowset method
TotalRowCount	RowCount Rowset property
Ungray	Enabled Field property
UnHide	Visible Field property
UnHideRow	Visible Row property
UnhideScroll	ShowAllRows Rowset method
UpdateValue	Value Field property

Mapping of Old Names to New Names

In PeopleTools 8.1, the names of some of the definitions in PeopleSoft Application Designer changed. The names of related built-in functions have changed accordingly.

The first table lists the old terms and new terms.

In the second table, the left column lists the old names of the functions, system variables, or reserved words. The right column lists the new names. The existing functions, system variables, and reserved words in this table still work in PeopleTools 8.0, however, they are being retained for backward compatibility only. New applications should be created using the new functions, system variables, and reserved words.

 Copyright © 1988-2007, Oracle. All rights reserved.

Old Term	New Term
Operator	User
Panel	Page
Panel Group	Component
Business Component	Component Interface

Deprecated Function, System Variable or Reserved Word	New Function, System Variable or Reserved Word
DoModalPanelGroup built-in function	DoModalComponent built-in function
GetNextNumberWithGaps	GetNextNumberWithGapsCommit built-in function
IsModalPanelGroup built-in function	IsModalComponent built-in function
IsOperatorInClass built-in function	IsUserInPermissionList built-in function
PanelGroupChanged built-in function	ComponentChanged built-in function
ScheduleProcess built-in function	CreateProcessRequest built-in function
SetNextPanel built-in function	SetNextPage built-in function
TransferPanel built-in function	TransferPage built-in function
%OperatorClass System Variable	%PrimaryPermissionList System Variable
%OperatorID System Variable	%UserId System Variable
%OperatorRowLevelSecurityClass System Variable	%RowSecurityPermissionList System Variable
%Panel System Variable	%Page System Variable
%PanelGroup System Variable	%Component System Variable
PANEL reserved word	PAGE reserved word

Copyright © 1988-2007, Oracle. All rights reserved.

Deprecated Function, System Variable or Reserved Word	New Function, System Variable or Reserved Word
PANELGROUP reserved word	COMPONENT reserved word
PanelGroup variable declaration	Component variable declaration

Business Components are now named Component Interfaces. For Component Interfaces, the old reserved word, methods, and system variables *are no longer valid*. You *must* use the new reserved word, methods, or system variables.

No Longer Valid	Use Instead
COMPONENT reserved word	COMPINTFC reserved word
GetComponent method	GetCompIntfc method
FindComponent method	FindCompIntfcs method
%Component system variable	%CompIntfc system variable

Deprecated Functions

The following functions are deprecated, and no longer work in release 8.4 and beyond.

Function	Description
%MessageAgent	Used with Message Agent activities
ChDir	Used only on the client with DOS
ChDrive	Used only on the client with DOS
CheckMenuItem	Used with menu items
Codeb	Used before true Unicode integration
DBCSTrim	Used before true Unicode integration

 Copyright © 1988-2007, Oracle. All rights reserved.

Function	Description
Findb	Used before true Unicode integration
GetControl	Used with ActiveX controls
GetControlOccurence	Used with ActiveX controls
GetCwd	Used only on the client with DOS
GetMessageInstance	Used with messaging
GetPubContractInstance	Used with messaging
GetSelectedTreeNode	Used with dynamic trees
GetSubContractInstance	Used with messaging
GetTreeNodeParent	Used with dynamic trees
GetTreeNodeRecordName	Used with dynamic trees
Lenb	Used before true Unicode integration
RefreshTree	Used with dynamic trees
ScheduleProcess	Used with Peoplesoft Process Scheduler
Substringb	Used before true Unicode integration
UnCheckMenuItem	Used with menu items
WinEscape	Used only on client
WinExec	Used only on client

Deprecated Methods and Properties

The following are the methods and properties that have been deprecated.

Copyright © 1988-2007, Oracle. All rights reserved.

Method or Property	Use Instead
Session Class Connect method	%Session system variable
Session Class FindTree, FindTreeStructure methods	GetTree, GetTreeStructure session class methods (respectively)
Tree collection	All tree collection methods and properties have been deprecated. Use GetTree to access a tree instead.
Tree structure collection	All tree structure collection methods and properties have been deprecated. Use GetTreeStructure to access a tree structure instead.
Item tree branch collection method, and the First, Last, and Next tree branch collection properties	
Tree class methods: • FindNextDisplayObject • FindPreviousDisplayObject	
Tree class properties: • CheckLeafUserData • DisplayLeaf • DisplayNode • IsReadOnly	

Deprecated Messaging PeopleCode Functions, Methods and Properties

In PeopleTools 8.48, messaging changed significantly. Many of the messaging PeopleCode functions, methods and properties are either deprecated or no longer supported, or their syntax changed.

See Also

Chapter 22, “Message Classes,” page 729

Enterprise PeopleTools 8.49 PeopleBook: PeopleSoft Integration Broker, “Understanding PeopleSoft Integration Broker”

Changed Messaging PeopleCode

The following are still valid, however, their syntax changed significantly.

 Copyright © 1988-2007, Oracle. All rights reserved.

- CreateMessage built-in function
- GetMessageErrors IntBroker class method
- SetErrorMessage IntBroker class method

Deprecated Messaging PeopleCode Built-in Functions

The following table lists the deprecated messaging PeopleCode built-in functions, as well as the method that should be used instead.

Deprecated Function	Replacement Method
CancelPubHeaderXmlDoc	IntBroker class method Cancel, specifying a message type of %IntBroker_BRK
CancelPubXmlDoc	IntBroker class method Cancel, specifying a message type of %IntBroker_PUB
CancelSubXmlDoc	IntBroker class method Cancel, specifying a message type of %IntBroker_SUB
ConnectorRequest	IntBroker class method ConnectorRequest
ConnectorRequestURL	IntBroker class method ConnectorRequestURL
GetMessage	IntBroker class method GetMessage
GetMessageXmlDoc	Message class method GetXmlDoc
GetPubHeaderXmlDoc	IntBroker class method GetMessage specifying a message type of %IntBroker_BRK
GetPubXmlDoc	IntBroker class method GetMessage specifying a message type of %IntBroker_PUB
GetSubXmlDoc	IntBroker class method GetMessage specifying a message type of %IntBroker_SUB
GetSyncLogData	IntBroker class method GetSyncLogData
InBoundPublishXmlDoc	IntBroker class method InboundPublish
IsMessageActive	IntBroker class method IsOperationActive
PublishXmlDoc	IntBroker class method Publish
ReSubmitPubHeaderXmlDoc	IntBroker class method Resubmit specifying a message type of %IntBroker_BRK
ReSubmitPubXmlDoc	IntBroker class method Resubmit specifying a message type of %IntBroker_PUB
ReSubmitSubXmlDoc	IntBroker class method Resubmit specifying a message type of %IntBroker_SUB

Copyright © 1988-2007, Oracle. All rights reserved.

Deprecated Function	Replacement Method
SetChannelStatue	IntBroker class method SetQueueStatus
SyncRequestXmlDoc	IntBroker class method SyncRequest
UpdateXmlDoc	IntBroker class method UpdateXmlDoc

The following built-in functions are no longer supported. You will receive an error if you try to use them in a PeopleCode program.

- CreateWSDLMessage
- GetArchPubHeaderXmlDoc
- GetArchPubXmlDoc
- GetArchSubXmlDoc
- GetArchSubXmlDoc
- GetArchSyncLogData
- GetSyncLogData

Note. ReturnToServer is a special case of a built-in function that's no longer supported. The deprecated handler for OnRequest subscriptions cannot be upgraded. ReturnToServer can only be used in an OnRequest event fired using the deprecated handler. This means that ReturnToServer no longer works and is not valid in any case other than when the code has already been written and used in a deprecated handler.

Deprecated Message Class Methods and Properties

The following are the message class methods and properties that have been deprecated.

Method	Replacement Method
GetQueryString	ConnectorInfo collection method GetQueryStringArgName
InBoundPublish	IntBroker class method InBoundPublish
Publish	IntBroker class method Publish
SetQueryString	ConnectorInfo collection method AddQueryStringArg
SetStatus	IntBroker class method SetStatus
SyncRequest	IntBroker class method SyncRequest
Update	IntBroker class method Update

The following are the message class properties that have been deprecated.

 Copyright © 1988-2007, Oracle. All rights reserved.

Property	Replacement Property
ChannelName	Message class property QueueName
Cookies	IBInfo class property Cookies
DefaultMessageVersion	Message class property OperationVersion
GUID	Message class property TransactionId
IsActive	Message class property IsOperationActive
PubId	Message class property QueueSeqId
SubName	Message class property ActionName
SubscriptionProcessId	Message class property TransactionId
MessageChannel	IBInfo class property MessageQueue
MessageType	IBInfor class property OperationType

The following message class properties are no longer supported. You will receive an error if you try to use them in your PeopleCode program.

- MessageDetail
- PublicationId

Additional Messaging Deprecated Methods and Properties

The following additional methods and properties are either deprecated or no longer supported:

- The IBInfo class method LoadConnectorPropFromTrx is no longer supported.
- The IBInfo class MessageChannel property has been deprecated. Use the IBInfo class MessageQueue property instead.
- The IBInfo class PublicationID property is no longer supported.

Integration Broker Deprecated Methods

The Integration Broker Web Service Discovery class is no longer supported. The following methods are empty and should not be used.

- FindCIWebServices
- FindMsgWebServices
- FindWebServices
- GetCIWebServiceWSDL
- GetMsgWebServiceWSDL
- GetWSDL

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleCode Syntax Quick Reference

The following is a description of the PeopleCode syntax, as derived from the PeopleCode parser. This description uses these metacharacters:

Metacharacter	Description
[ ]	Brackets are used to indicate optional portions.
...	Ellipses are used to indicate that the preceding part (often optional) repeats an arbitrary number of times.
|	Vertical bars are used to separate alternatives. They indicate that one of the alternatives should be present. The alternatives extend only to the immediately enclosing parentheses or brackets (if any).
()	Parentheses are used to group portions of definitions to limit the scope of alternatives (|) or optionality (...).
'symbols'	Quoted characters in **keyword style** are keywords or punctuation in the language. They should appear literally, without the quotation marks.

The following is the PeopleCode language syntax quick reference:

Term	Syntax
Program	ImportList (DeclList TopStList | ClassDefn)
ImportList	[ImportDecl **';'**]...
ImportDecl	**'import'** PackageName [**':'** **'*'**]
PackageName	Id [**':'** Id]...
ClassDefn	ClassDecl [**';'** ExtDecls] [**';'** MethodDefns]
ClassDecl	**'class'**| **'interface'**' Id [Extends] [ClassPublics] [ClassProtecteds] [ClassPrivates] **'end-class'** | **'end-interface'**
Extends	**'extends'** | **'implements'** QualifiedId
QualifiedId	[PackageName **':'**] Id
ClassPublics	ClassPublic [**';'** ClassPublic]...
ClassPublic	MethodDecl | PropertyDecl
MethodDecl	**'method'** Id **'('** [MethodParameters] **')'** [**'abstract'**] [**'returns'** PeopleCodeType]

Copyright © 1988-2007, Oracle. All rights reserved.

Term	Syntax
MethodParameters	MethodParameter [**‘,’** MethodParameter]...
MethodParameter	**‘&’** Id **‘as’** PeopleCodeType [**‘out’**]
PropertyDecl	**‘property’** PeopleCodeType Id [(**‘get’** [**‘set’**] | **’abstract’**) | **‘readonly’**]
ClassPrivates	**‘private’** [ClassPrivate [**‘;’** ClassPrivate]...]
ClassPrivate	MethodDecl | InstanceDecl | ConstantDecl
ClassProtecteds	**‘protected’** [ClassProtected [**‘;’** ClassProtected]...]
ClassProtected	MethodDecl | InstanceDecl | ConstantDecl
InstanceDecl	**‘instance’** VarDeclare
ExtDecls	ExtDecl [**‘;’** ExtDecl]...
MethodDefns	MethodDefn [**‘;’** MethodDefn]...
MethodDefn	MethodMethod | GetMethod | SetMethod
MethodMethod	**‘method’** Id StList **‘end-method’**
GetMethod	**‘get’** Id StList **‘end-get’**
SetMethod	**‘set’** Id StList **‘end-set’**
TopStList	TopStmt [**‘;’** TopStmt]...
TopStmt	Stmt
StList	Stmt [**‘;’** Stmt]...
Stme	[LValue [Assign] | If | Evaluate | While | Repeat | For | Accept | Break | Return | Exit | Error | Warning | Try | Throw | LocalDecl]
Assign	**‘=’** Expression
If	**‘if’** LogicalExpression **‘then’** StList [**‘else’** StList] **‘end-if’**
Evaluate	**‘evaluate’** Expression [WhenExpr... StList]... [**‘when-other’** StList] **‘end-evaluate’**
WhenExpr	**‘when’** [RelOp] Expression
While	**‘while’** LogicalExpression StList **‘end-while’**
Repeat	**‘repeat’** StList **‘until’** LogicalExpression

Copyright © 1988-2007, Oracle. All rights reserved.

Term	Syntax
For	**'for'** Variable **'='** Expression **'to'** Expression [**'step'** Expression] StList **'end-for'**
Accept	**'accept'**
Return	**'return'** [Expression]
Break	**'break'**
Exit	**'exit'** [Expression]
Warning	**'warning'** Expression
Error	**'error'** Expression
Try	**'try'** StList (**'catch'** QualifiedId **'&'** Id StList)... **'end-try'**
Throw	**'throw'** Expresssion
LocalDecl	**'local'** PeopleCodeType **'&'** Id [(**','** **'&'** Id)... | **'='** Expression]
Continue	**'Continue'**
LogicalExpression	LogicalTerm [**'or'** LogicalTerm]...
LogicalTerm	Relation [**'and'** Relation]...
Relation	**'not'** Relation | Expression [Relop Expression]
Relop	**'not'** Relop | **'<'** | **'<='** | **'='** | **'>='** | **'>'** | **'!='** | **'<>'**
Expression	Term [(**'+'** | **'-'** | **'|'**) Term]...
Term	Power [(**'*'** | **'/'**) Power]...
Power	Primary [**'**'** Primary]...
Primary	LValue [**'as'** QualifiedId] | Number | String | **'true'** | **'false'** | **'null'** | **'-'** Primary | **'create'** QualifiedId Call
LValue	LValueStart [LValueObject | LValueSubscript]...
LValueStart	Variable | Id Call | **'@'** Primary | **'%'** BuiltinVar | **'('** LogicalExpression **')'**
LValueObject	**'.'** Id [Call] | Call
LValueSubscript	**'['** Expression [**','** Expression]... **']'**
Variable	**'&'** Id | Id [**'.'** (Id | String)] | '^'

Copyright © 1988-2007, Oracle. All rights reserved.

Term	Syntax
Call	**‘(’** [Expression [**‘,’** Expression]...] **‘)’**
Constant	Number | String | **‘true’** | **‘false’** | **’null’** | QualifiedId **‘:’** Id
DecList	[Decl **‘;’**]...
Decl	**‘local’** VarDeclare | **‘function’** Function | ConstDecl | ExtDecl
VarDeclare	PeopleCodeType **‘&’** Id [**‘,’** **‘&’** Id]...
Function	Id [**‘(’** InternalParameters **‘)’**] [**‘returns’** PeopleCodeType] [**‘noexport’**] [**‘doc’** String] TopStList **‘end-function’**
InternalParameters	[InternalParameter [**‘,’** InternalParameter]...]
InternalParameter	**‘&’** Id [**‘as’** PeopleCodeType]
ConstDecl	**‘constant’** **‘&’** Id **‘=’** (Number | String | **‘true’** | **‘false’** | **’null’**)
ExtDecl	**‘global’** VarDeclare | **‘component’** VarDeclare | **‘declare’** Declare
Declare	**‘function’** Id (**‘library’** DeclareLibrary | **‘peoplecode’** DeclarePC)
DeclareLibrary	String [**‘alias’** String] [**‘(’** ExternalParameters **‘)’**] [**‘returns’** ExternalType [**‘as’** PeopleCodeType]]
ExternalParameters	[ExternalParameter [**‘,’** ExternalParameter]...]
ExternalParameter	ExternalType [**‘value’** | **‘ref’**] [**‘as’** PeopleCodeType]
DeclarePC	Variable EventType
PeopleCode Type	[**‘array’** **‘of’**]... (**‘number’** | **‘string’** | **‘date’** | **‘time’** | **‘datetime’** | **‘any’** | **‘boolean’** | **‘object’** | **‘array’** | **‘integer’** | **‘float’** | QualifiedId)
ExternalType	**‘boolean’** | **‘integer’** | **‘long’** | **‘uinteger’** | **‘ulong’** | **‘string’** | **‘lstring’** | **‘ustring’** | **‘float’** | **‘double’**

Copyright © 1988-2007, Oracle. All rights reserved.

 Copyright © 1988-2007, Oracle. All rights reserved.

Glossary of PeopleSoft Enterprise Terms

absence entitlement	This element defines rules for granting paid time off for valid absences, such as sick time, vacation, and maternity leave. An absence entitlement element defines the entitlement amount, frequency, and entitlement period.
absence take	This element defines the conditions that must be met before a payee is entitled to take paid time off.
academic career	In PeopleSoft Enterprise Campus Solutions, all course work that a student undertakes at an academic institution and that is grouped in a single student record. For example, a university that has an undergraduate school, a graduate school, and various professional schools might define several academic careers—an undergraduate career, a graduate career, and separate careers for each professional school (law school, medical school, dental school, and so on).
academic institution	In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.
academic organization	In PeopleSoft Enterprise Campus Solutions, an entity that is part of the administrative structure within an academic institution. At the lowest level, an academic organization might be an academic department. At the highest level, an academic organization can represent a division.
academic plan	In PeopleSoft Enterprise Campus Solutions, an area of study—such as a major, minor, or specialization—that exists within an academic program or academic career.
academic program	In PeopleSoft Enterprise Campus Solutions, the entity to which a student applies and is admitted and from which the student graduates.
accounting class	In PeopleSoft Enterprise Performance Management, the accounting class defines how a resource is treated for generally accepted accounting practices. The Inventory class indicates whether a resource becomes part of a balance sheet account, such as inventory or fixed assets, while the Non-inventory class indicates that the resource is treated as an expense of the period during which it occurs.
accounting date	The accounting date indicates when a transaction is recognized, as opposed to the date the transaction actually occurred. The accounting date and transaction date can be the same. The accounting date determines the period in the general ledger to which the transaction is to be posted. You can only select an accounting date that falls within an open period in the ledger to which you are posting. The accounting date for an item is normally the invoice date.
accounting split	The accounting split method indicates how expenses are allocated or divided among one or more sets of accounting ChartFields.
accumulator	You use an accumulator to store cumulative values of defined items as they are processed. You can accumulate a single value over time or multiple values over time. For example, an accumulator could consist of all voluntary deductions, or all company deductions, enabling you to accumulate amounts. It allows total flexibility for time periods and values accumulated.
action reason	The reason an employee's job or employment information is updated. The action reason is entered in two parts: a personnel action, such as a promotion, termination, or change from one pay group to another—and a reason for that action. Action reasons are used by PeopleSoft Enterprise Human Resources, PeopleSoft Enterprise Benefits

Copyright © 1988-2007, Oracle. All rights reserved.

Administration, PeopleSoft Enterprise Stock Administration, and the COBRA Administration feature of the Base Benefits business process.

action template In PeopleSoft Enterprise Receivables, outlines a set of escalating actions that the system or user performs based on the period of time that a customer or item has been in an action plan for a specific condition.

activity In PeopleSoft Enterprise Learning Management, an instance of a catalog item (sometimes called a class) that is available for enrollment. The activity defines such things as the costs that are associated with the offering, enrollment limits and deadlines, and waitlisting capacities.

In PeopleSoft Enterprise Performance Management, the work of an organization and the aggregation of actions that are used for activity-based costing.

In PeopleSoft Enterprise Project Costing, the unit of work that provides a further breakdown of projects—usually into specific tasks.

In PeopleSoft Workflow, a specific transaction that you might need to perform in a business process. Because it consists of the steps that are used to perform a transaction, it is also known as a step map.

address usage In PeopleSoft Enterprise Campus Solutions, a grouping of address types defining the order in which the address types are used. For example, you might define an address usage code to process addresses in the following order: billing address, dormitory address, home address, and then work address.

adjustment calendar In PeopleSoft Enterprise Campus Solutions, the adjustment calendar controls how a particular charge is adjusted on a student's account when the student drops classes or withdraws from a term. The charge adjustment is based on how much time has elapsed from a predetermined date, and it is determined as a percentage of the original charge amount.

administrative function In PeopleSoft Enterprise Campus Solutions, a particular functional area that processes checklists, communication, and comments. The administrative function identifies which variable data is added to a person's checklist or communication record when a specific checklist code, communication category, or comment is assigned to the student. This key data enables you to trace that checklist, communication, or comment back to a specific processing event in a functional area.

admit type In PeopleSoft Enterprise Campus Solutions, a designation used to distinguish first-year applications from transfer applications.

agreement In PeopleSoft Enterprise eSettlements, provides a way to group and specify processing options, such as payment terms, pay from a bank, and notifications by a buyer and supplier location combination.

allocation rule In PeopleSoft Enterprise Incentive Management, an expression within compensation plans that enables the system to assign transactions to nodes and participants. During transaction allocation, the allocation engine traverses the compensation structure from the current node to the root node, checking each node for plans that contain allocation rules.

alternate account A feature in PeopleSoft Enterprise General Ledger that enables you to create a statutory chart of accounts and enter statutory account transactions at the detail transaction level, as required for recording and reporting by some national governments.

analysis database In PeopleSoft Enterprise Campus Solutions, database tables that store large amounts of student information that may not appear in standard report formats. The analysis database tables contain keys for all objects in a report that an application program can use to reference other student-record objects that are not contained in the printed report. For instance, the analysis database contains data on courses that are considered

Copyright © 1988-2007, Oracle. All rights reserved.

	for satisfying a requirement but that are rejected. It also contains information on courses captured by global limits. An analysis database is used in PeopleSoft Enterprise Academic Advisement.
Application Messaging	PeopleSoft Application Messaging enables applications within the PeopleSoft Enterprise product family to communicate synchronously or asynchronously with other PeopleSoft Enterprise and third-party applications. An application message defines the records and fields to be published or subscribed to.
AR specialist	Abbreviation for *receivables specialist.* In PeopleSoft Enterprise Receivables, an individual in who tracks and resolves deductions and disputed items.
arbitration plan	The arbiter when multiple price rules match the transaction. This plan determines the order in which the price rules are applied to the transaction base price.
assessment rule	In PeopleSoft Enterprise Receivables, a user-defined rule that the system uses to evaluate the condition of a customer's account or of individual items to determine whether to generate a follow-up action.
asset class	An asset group used for reporting purposes. It can be used in conjunction with the asset category to refine asset classification.
attribute/value pair	In PeopleSoft Enterprise Directory Interface, relates the data that makes up an entry in the directory information tree.
auction event	In PeopleSoft Strategic Sourcing, a sourcing event where bidders actively compete against one another to achieve the best price or score.
audience	In PeopleSoft Enterprise Campus Solutions, a segment of the database that relates to an initiative, or a membership organization that is based on constituent attributes rather than a dues-paying structure. Examples of audiences include the Class of '65 and Undergraduate Arts & Sciences.
authentication server	A server that is set up to verify users of the system.
base time period	In PeopleSoft Enterprise Business Planning, the lowest level time period in a calendar.
benchmark job	In PeopleSoft Enterprise Workforce Analytics Solution, a benchmark job is a job code for which there is corresponding salary survey data from published, third-party sources.
bid response	In PeopleSoft Strategic Sourcing, the response by a bidder to an event.
billing career	In PeopleSoft Enterprise Campus Solutions, the one career under which other careers are grouped for billing purposes if a student is active simultaneously in multiple careers.
bio bit or bio brief	In PeopleSoft Enterprise Campus Solutions, a report that summarizes information stored in the system about a particular constituent. You can generate standard or specialized reports.
book	In PeopleSoft Enterprise Asset Management, used for storing financial and tax information, such as costs, depreciation attributes, and retirement information on assets.
branch	A tree node that rolls up to nodes above it in the hierarchy, as defined in PeopleSoft Tree Manager.
budgetary account only	An account used by the system only and not by users; this type of account does not accept transactions. You can only budget with this account. Formerly called "system-maintained account."

Copyright © 1988-2007, Oracle. All rights reserved.

budget check In commitment control, the processing of source transactions against control budget ledgers, to see if they pass, fail, or pass with a warning.

budget control In commitment control, budget control ensures that commitments and expenditures don't exceed budgets. It enables you to track transactions against corresponding budgets and terminate a document's cycle if the defined budget conditions are not met. For example, you can prevent a purchase order from being dispatched to a vendor if there are insufficient funds in the related budget to support it.

budget period The interval of time (such as 12 months or 4 quarters) into which a period is divided for budgetary and reporting purposes. The ChartField allows maximum flexibility to define operational accounting time periods without restriction to only one calendar.

business activity The name of a subset of a detailed business process. This might be a specific transaction, task, or action that you perform in a business process.

business event In PeopleSoft Enterprise Receivables, defines the processing characteristics for the Receivable Update process for a draft activity.

In PeopleSoft Enterprise Sales Incentive Management, an original business transaction or activity that may justify the creation of a PeopleSoft Enterprise Incentive Management event (a sale, for example).

business process A standard set of 17 business processes are defined and maintained by the PeopleSoft Enterprise product families and are supported by the Business Process Engineering group. An example of a business process is Order Fulfillment, which is a business process that manages sales orders and contracts, inventory, billing, and so forth.

See also *detailed business process.*

business unit constraints In PeopleSoft Strategic Sourcing, these constraints apply to a selected Strategic Sourcing business unit. Spend is tracked across all of the events within the selected Strategic Sourcing business unit.

business task The name of the specific function depicted in one of the business processes.

business unit A corporation or a subset of a corporation that is independent with regard to one or more operational or accounting functions.

buyer In PeopleSoft Enterprise eSettlements, an organization (or business unit, as opposed to an individual) that transacts with suppliers (vendors) within the system. A buyer creates payments for purchases that are made in the system.

buy event In PeopleSoft Strategic Sourcing, for event creators, the purchase of goods or services, most typically associated with a request for quote, proposal, or reverse auction.For bidders, the sale of goods or services.

campus In PeopleSoft Enterprise Campus Solutions, an entity that is usually associated with a distinct physical administrative unit, that belongs to a single academic institution, that uses a unique course catalog, and that produces a common transcript for students within the same academic career.

cash drawer A repository for monies and payments taken locally.

catalog item In PeopleSoft Enterprise Learning Management, a specific topic that a learner can study and have tracked. For example, "Introduction to Microsoft Word." A catalog item contains general information about the topic and includes a course code, description, categorization, keywords, and delivery methods. A catalog item can have one or more learning activities.

catalog map In PeopleSoft Enterprise Catalog Management, translates values from the catalog source data to the format of the company's catalog.

Copyright © 1988-2007, Oracle. All rights reserved.

catalog partner In PeopleSoft Enterprise Catalog Management, shares responsibility with the enterprise catalog manager for maintaining catalog content.

categorization Associates partner offerings with catalog offerings and groups them into enterprise catalog categories.

category In PeopleSoft Enterprise Campus Solutions, a broad grouping to which specific comments or communications (contexts) are assigned. Category codes are also linked to 3C access groups so that you can assign data-entry or view-only privileges across functions.

channel In PeopleSoft MultiChannel Framework, email, chat, voice (computer telephone integration [CTI]), or a generic event.

ChartField A field that stores a chart of accounts, resources, and so on, depending on the PeopleSoft Enterprise application. ChartField values represent individual account numbers, department codes, and so forth.

ChartField balancing You can require specific ChartFields to match up (balance) on the debit and the credit side of a transaction.

ChartField combination edit The process of editing journal lines for valid ChartField combinations based on user-defined rules.

ChartKey One or more fields that uniquely identify each row in a table. Some tables contain only one field as the key, while others require a combination.

checkbook In PeopleSoft Enterprise Promotions Management, enables you to view financial data (such as planned, incurred, and actual amounts) that is related to funds and trade promotions.

checklist code In PeopleSoft Enterprise Campus Solutions, a code that represents a list of planned or completed action items that can be assigned to a staff member, volunteer, or unit. Checklists enable you to view all action assignments on one page.

claimback In the wholesale distribution industry, a contract between supplier and distributor, in which monies are paid to the distributor on the sale of specified products or product groups to targeted customers or customer groups.

class In PeopleSoft Enterprise Campus Solutions, a specific offering of a course component within an academic term.

See also *course.*

Class ChartField A ChartField value that identifies a unique appropriation budget key when you combine it with a fund, department ID, and program code, as well as a budget period. Formerly called *sub-classification.*

clearance In PeopleSoft Enterprise Campus Solutions, the period of time during which a constituent in PeopleSoft Enterprise Contributor Relations is approved for involvement in an initiative or an action. Clearances are used to prevent development officers from making multiple requests to a constituent during the same time period.

clone In PeopleCode, to make a unique copy. In contrast, to *copy* may mean making a new reference to an object, so if the underlying object is changed, both the copy and the original change.

cohort In PeopleSoft Enterprise Campus Solutions, the highest level of the three-level classification structure that you define for enrollment management. You can define a cohort level, link it to other levels, and set enrollment target numbers for it.

See also *population* and *division*.

Copyright © 1988-2007, Oracle. All rights reserved.

collection To make a set of documents available for searching in Verity, you must first create at least one collection. A collection is set of directories and files that allow search application users to use the Verity search engine to quickly find and display source documents that match search criteria. A collection is a set of statistics and pointers to the source documents, stored in a proprietary format on a file server. Because a collection can only store information for a single location, PeopleTools maintains a set of collections (one per language code) for each search index object.

collection rule In PeopleSoft Enterprise Receivables, a user-defined rule that defines actions to take for a customer based on both the amount and the number of days past due for outstanding balances.

comm key See *communication key.*

communication key In PeopleSoft Enterprise Campus Solutions, a single code for entering a combination of communication category, communication context, communication method, communication direction, and standard letter code. Communication keys (also called *comm keys* or *speed keys*) can be created for background processes as well as for specific users.

compensation object In PeopleSoft Enterprise Incentive Management, a node within a compensation structure. Compensation objects are the building blocks that make up a compensation structure's hierarchical representation.

compensation structure In PeopleSoft Enterprise Incentive Management, a hierarchical relationship of compensation objects that represents the compensation-related relationship between the objects.

component interface A component interface is a set of application programming interfaces (APIs) that you can use to access and modify PeopleSoft Enterprise database information using a program instead of the PeopleSoft client.

condition In PeopleSoft Enterprise Receivables, occurs when there is a change of status for a customer's account, such as reaching a credit limit or exceeding a user-defined balance due.

configuration parameter catalog Used to configure an external system with PeopleSoft Enterprise. For example, a configuration parameter catalog might set up configuration and communication parameters for an external server.

configuration plan In PeopleSoft Enterprise Incentive Management, configuration plans hold allocation information for common variables (not incentive rules) and are attached to a node without a participant. Configuration plans are not processed by transactions.

constituents In PeopleSoft Enterprise Campus Solutions, friends, alumni, organizations, foundations, or other entities affiliated with the institution, and about which the institution maintains information. The constituent types delivered with PeopleSoft Enterprise Contributor Relations Solutions are based on those defined by the Council for the Advancement and Support of Education (CASE).

constraint A business policy or rule that affects how a sourcing event is awarded. There are three types of constraints: business, global, and event.

content reference Content references are pointers to content registered in the portal registry. These are typically either URLs or iScripts. Content references fall into three categories: target content, templates, and template pagelets.

context In PeopleCode, determines which buffer fields can be contextually referenced and which is the current row of data on each scroll level when a PeopleCode program is running.

In PeopleSoft Enterprise Campus Solutions, a specific instance of a comment or communication. One or more contexts are assigned to a category, which you link to

 Copyright © 1988-2007, Oracle. All rights reserved.

	3C access groups so that you can assign data-entry or view-only privileges across functions. In PeopleSoft Enterprise Incentive Management, a mechanism that is used to determine the scope of a processing run. PeopleSoft Enterprise Incentive Management uses three types of context: plan, period, and run-level.
control table	Stores information that controls the processing of an application. This type of processing might be consistent throughout an organization, or it might be used only by portions of the organization for more limited sharing of data.
cost plus contract line	A rate-based contract line associated with a fee component of Award, Fixed, Incentive, or Other. Rate-based contract lines associated with a fee type of None are not considered cost-plus contract lines.
cost plus pricing	In PeopleSoft Enterprise Pricer, a pricing method that begins with cost of goods as the basis.
cost profile	A combination of a receipt cost method, a cost flow, and a deplete cost method. A profile is associated with a cost book and determines how items in that book are valued, as well as how the material movement of the item is valued for the book.
cost row	A cost transaction and amount for a set of ChartFields.
counter sale	A face-to-face customer transaction where the customer typically selects items from the storefront or picks up products that they ordered ahead of time. Customers pay for the goods at the counter and take the goods with them instead of having the goods shipped from a warehouse.
course	In PeopleSoft Enterprise Campus Solutions, a course that is offered by a school and that is typically described in a course catalog. A course has a standard syllabus and credit level; however, these may be modified at the class level. Courses can contain multiple components such as lecture, discussion, and lab. See also *class.*
course share set	In PeopleSoft Enterprise Campus Solutions, a tag that defines a set of requirement groups that can share courses. Course share sets are used in PeopleSoft Enterprise Academic Advisement.
current learning	In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's in-progress learning activities and programs.
data acquisition	In PeopleSoft Enterprise Incentive Management, the process during which raw business transactions are acquired from external source systems and fed into the operational data store (ODS).
data cube	In PeopleSoft Analytic Calculation Engine, a data cube is a container for one kind of data (such as Sales data) and works with in tandem with one or more dimensions. Dimensions and data cubes in PeopleSoft Analytic Calculation Engine are unrelated to dimensions and online analytical processing (OLAP) cubes in PeopleSoft Cube Manager.
data elements	Data elements, at their simplest level, define a subset of data and the rules by which to group them. For Workforce Analytics, data elements are rules that tell the system what measures to retrieve about your workforce groups.
dataset	A data grouping that enables role-based filtering and distribution of data. You can limit the range and quantity of data that is displayed for a user by associating dataset rules with user roles. The result of dataset rules is a set of data that is appropriate for the user's roles.

Copyright © 1988-2007, Oracle. All rights reserved.

delivery method

In PeopleSoft Enterprise Learning Management, identifies the primary type of delivery method in which a particular learning activity is offered. Also provides default values for the learning activity, such as cost and language. This is primarily used to help learners search the catalog for the type of delivery from which they learn best. Because PeopleSoft Enterprise Learning Management is a blended learning system, it does not enforce the delivery method.

In PeopleSoft Enterprise Supply Chain Management, identifies the method by which goods are shipped to their destinations (such as truck, air, and rail). The delivery method is specified when creating shipment schedules.

delivery method type

In PeopleSoft Enterprise Learning Management, identifies how learning activities can be delivered—for example, through online learning, classroom instruction, seminars, books, and so forth—in an organization. The type determines whether the delivery method includes scheduled components.

detailed business process

A subset of the business process. For example, the detailed business process named Determine Cash Position is a subset of the business process called Cash Management.

dimension

In PeopleSoft Analytic Calculation Engine, a dimension contains a list of one kind of data that can span various contexts, and it is a basic component of an analytic model. Within the analytic model, a dimension is attached to one or more data cubes. In PeopleSoft Cube Manager, a dimension is the most basic component of an OLAP cube and specifies the PeopleSoft metadata to be used to create the dimension's rollup structure. Dimensions and data cubes in PeopleSoft Analytic Calculation Engine are unrelated to dimensions and OLAP cubes in PeopleSoft Cube Manager.

direct receipt

Items shipped from a warehouse or vendor to another warehouse.

direct ship

Items shipped from the vendor or warehouse directly to the customer (formerly referred to as *drop ship*).

directory information tree

In PeopleSoft Enterprise Directory Interface, the representation of a directory's hierarchical structure.

division

In PeopleSoft Enterprise Campus Solutions, the lowest level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and Admissions for enrollment management. You can define a division level, link it to other levels, and set enrollment target numbers for it.

See also *population* and *cohort*.

document sequencing

A flexible method that sequentially numbers the financial transactions (for example, bills, purchase orders, invoices, and payments) in the system for statutory reporting and for tracking commercial transaction activity.

dynamic detail tree

A tree that takes its detail values—dynamic details—directly from a table in the database, rather than from a range of values that are entered by the user.

edit table

A table in the database that has its own record definition, such as the Department table. As fields are entered into a PeopleSoft Enterprise application, they can be validated against an edit table to ensure data integrity throughout the system.

effective date

A method of dating information in PeopleSoft Enterprise applications. You can predate information to add historical data to your system, or postdate information in order to enter it before it actually goes into effect. By using effective dates, you don't delete values; you enter a new value with a current effective date.

EIM ledger

Abbreviation for *Enterprise Incentive Management ledger.* In PeopleSoft Enterprise Incentive Management, an object to handle incremental result gathering within the scope of a participant. The ledger captures a result set with all of the appropriate traces to the data origin and to the processing steps of which it is a result.

Copyright © 1988-2007, Oracle. All rights reserved.

elimination set In PeopleSoft Enterprise General Ledger, a related group of intercompany accounts that is processed during consolidations.

entry event In PeopleSoft Enterprise General Ledger, Receivables, Payables, Purchasing, and Billing, a business process that generates multiple debits and credits resulting from single transactions to produce standard, supplemental accounting entries.

equitization In PeopleSoft Enterprise General Ledger, a business process that enables parent companies to calculate the net income of subsidiaries on a monthly basis and adjust that amount to increase the investment amount and equity income amount before performing consolidations.

equity item limit In PeopleSoft Enterprise Campus Solutions, the amounts of funds set by the institution to be awarded with discretionary or gift funds. The limit could be reduced by amounts equal to such things as expected family contribution (EFC) or parent contribution. Students are packaged by Equity Item Type Groups and Related Equity Item Types. This limit can be used to assure that similar student populations are packaged equally.

event A predefined point either in the Component Processor flow or in the program flow. As each point is encountered, the event activates each component, triggering any PeopleCode program that is associated with that component and that event. Examples of events are FieldChange, SavePreChange, and RowDelete.

In PeopleSoft Enterprise Human Resources, also refers to an incident that affects benefits eligibility.

event constraints In PeopleSoft Strategic Sourcing, these constraints are associated with a specific sourcing event. Spend is tracked within the selected event.

event propagation process In PeopleSoft Enterprise Sales Incentive Management, a process that determines, through logic, the propagation of an original PeopleSoft Enterprise Incentive Management event and creates a derivative (duplicate) of the original event to be processed by other objects. PeopleSoft Enterprise Enterprise Sales Incentive Management uses this mechanism to implement splits, roll-ups, and so on. Event propagation determines who receives the credit.

exception In PeopleSoft Enterprise Receivables, an item that either is a deduction or is in dispute.

exclusive pricing In PeopleSoft Enterprise Order Management, a type of arbitration plan that is associated with a price rule. Exclusive pricing is used to price sales order transactions.

fact In PeopleSoft Enterprise applications, facts are numeric data values from fields from a source database as well as an analytic application. A fact can be anything you want to measure your business by, for example, revenue, actual, budget data, or sales numbers. A fact is stored on a fact table.

financial aid term In PeopleSoft Enterprise Campus Solutions, a combination of a period of time that the school determines as an instructional accounting period and an academic career. It is created and defined during the setup process. Only terms eligible for financial aid are set up for each financial aid career.

financial sanctions For U.S. based companies and their foreign subsidiaries, a federal regulation from the Office of Foreign Assets Control (OFAC) requires that vendors be validated against a Specially Designated Nationals (SDN) list prior to payment.

For PeopleSoft Payables, eSettlements, Cash Management, and Order to Cash, you can validate your vendors against any financial sanctions list (for example, the SDN list, a European Union list, and so on).

forecast item A logical entity with a unique set of descriptive demand and forecast data that is used as the basis to forecast demand. You create forecast items for a wide range of uses, but they ultimately represent things that you buy, sell, or use in your organization and for which you require a predictable usage.

Copyright © 1988-2007, Oracle. All rights reserved.

fund In PeopleSoft Enterprise Promotions Management, a budget that can be used to fund promotional activity. There are four funding methods: top down, fixed accrual, rolling accrual, and zero-based accrual.

gap In PeopleSoft Enterprise Campus Solutions, an artificial figure that sets aside an amount of unmet financial aid need that is not funded with Title IV funds. A gap can be used to prevent fully funding any student to conserve funds, or it can be used to preserve unmet financial aid need so that institutional funds can be awarded.

generic process type In PeopleSoft Process Scheduler, process types are identified by a generic process type. For example, the generic process type SQR includes all SQR process types, such as SQR process and SQR report.

gift table In PeopleSoft Enterprise Campus Solutions, a table or so-called *donor pyramid* describing the number and size of gifts that you expect will be needed to successfully complete the campaign in PeopleSoft Enterprise Contributor Relations. The gift table enables you to estimate the number of donors and prospects that you need at each gift level to reach the campaign goal.

GDS Abbreviation for *Global Distribution System.* Broad-based term to describe all computer reservation systems for making travel plans.

GL business unit Abbreviation for *general ledger business unit.* A unit in an organization that is an independent entity for accounting purposes. It maintains its own set of accounting books.

See also *business unit.*

GL entry template Abbreviation for *general ledger entry template.* In PeopleSoft Enterprise Campus Solutions, a template that defines how a particular item is sent to the general ledger. An item-type maps to the general ledger, and the GL entry template can involve multiple general ledger accounts. The entry to the general ledger is further controlled by high-level flags that control the summarization and the type of accounting—that is, accrual or cash.

GL Interface process Abbreviation for *General Ledger Interface process.* In PeopleSoft Enterprise Campus Solutions, a process that is used to send transactions from PeopleSoft Enterprise Student Financials to the general ledger. Item types are mapped to specific general ledger accounts, enabling transactions to move to the general ledger when the GL Interface process is run.

global constraints In PeopleSoft Strategic Sourcing, these constraints apply across multiple Strategic Sourcing business units. Spend is tracked across all of the events from the multiple Strategic Sourcing business units.

group In PeopleSoft Enterprise Billing and Receivables, a posting entity that comprises one or more transactions (items, deposits, payments, transfers, matches, or write-offs).

In PeopleSoft Enterprise Human Resources Management and Supply Chain Management, any set of records that are associated under a single name or variable to run calculations in PeopleSoft business processes. In PeopleSoft Enterprise Time and Labor, for example, employees are placed in groups for time reporting purposes.

ideal response In PeopleSoft Strategic Sourcing, a question that requires the response to match the ideal value for the bid to be considered eligible for award. If the response does not match the ideal value, you can still submit the bid, but it will be disqualified and ineligible for award.

incentive object In PeopleSoft Enterprise Incentive Management, the incentive-related objects that define and support the PeopleSoft Enterprise Incentive Management calculation process and results, such as plan templates, plans, results data, and user interaction objects.

Copyright © 1988-2007, Oracle. All rights reserved.

incentive rule In PeopleSoft Enterprise Sales Incentive Management, the commands that act on transactions and turn them into compensation. A rule is one part in the process of turning a transaction into compensation.

incur In PeopleSoft Enterprise Promotions Management, to become liable for a promotional payment. In other words, you owe that amount to a customer for promotional activities.

initiative In PeopleSoft Enterprise Campus Solutions, the basis from which all advancement plans are executed. It is an organized effort targeting a specific constituency, and it can occur over a specified period of time with specific purposes and goals. An initiative can be a campaign, an event, an organized volunteer effort, a membership drive, or any other type of effort defined by the institution. Initiatives can be multipart, and they can be related to other initiatives. This enables you to track individual parts of an initiative, as well as entire initiatives.

inquiry access In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user only to view data.

See also *update access.*

institution In PeopleSoft Enterprise Campus Solutions, an entity (such as a university or college) that is independent of other similar entities and that has its own set of rules and business processes.

integration A relationship between two compatible integration points that enables communication to take place between systems. Integrations enable PeopleSoft Enterprise applications to work seamlessly with other PeopleSoft Enterprise applications or with third-party systems or software.

integration point An interface that a system uses to communicate with another PeopleSoft Enterprise application or an external application.

integration set A logical grouping of integrations that applications use for the same business purpose. For example, the integration set ADVANCED_SHIPPING_ORDER contains all of the integrations that notify a customer that an order has shipped.

item In PeopleSoft Enterprise Inventory, a tangible commodity that is stored in a business unit (shipped from a warehouse).

In PeopleSoft Enterprise Demand Planning, Inventory Policy Planning, and Supply Planning, a noninventory item that is designated as being used for planning purposes only. It can represent a family or group of inventory items. It can have a planning bill of material (BOM) or planning routing, and it can exist as a component on a planning BOM. A planning item cannot be specified on a production or engineering BOM or routing, and it cannot be used as a component in a production. The quantity on hand will never be maintained.

In PeopleSoft Enterprise Receivables, an individual receivable. An item can be an invoice, a credit memo, a debit memo, a write-off, or an adjustment.

item shuffle In PeopleSoft Enterprise Campus Solutions, a process that enables you to change a payment allocation without having to reverse the payment.

itinerary In PeopleSoft Expenses, a collection of travel reservations. Itineraries can have reservations that are selected and reserved with the travel vendor. These itineraries are not yet paid for and can be referred to as *pending reservations.* Reservations that have been paid for are referred to as *confirmed reservations.*

joint communication In PeopleSoft Enterprise Campus Solutions, one letter that is addressed jointly to two people. For example, a letter might be addressed to both Mr. Sudhir Awat and Ms. Samantha Mortelli. A relationship must be established between the two individuals in the database, and at least one of the individuals must have an ID in the database.

Copyright © 1988-2007, Oracle. All rights reserved.

keyword In PeopleSoft Enterprise Campus Solutions, a term that you link to particular elements within PeopleSoft Enterprise Student Financials, Financial Aid, and Contributor Relations. You can use keywords as search criteria that enable you to locate specific records in a search dialog box.

KPI An abbreviation for *key performance indicator.* A high-level measurement of how well an organization is doing in achieving critical success factors. This defines the data value or calculation upon which an assessment is determined.

KVI Abbreviation for *Known Value Item.* Term used for products or groups of products where the selling price cannot be reduced or increased.

landlord In PeopleSoft Real Estate Management, an entity that owns real estate and leases the real estate to tenants.

LDIF file Abbreviation for *Lightweight Directory Access Protocol (LDAP) Data Interchange Format file.* Contains discrepancies between PeopleSoft Enterprise data and directory data.

learner group In PeopleSoft Enterprise Learning Management, a group of learners who are linked to the same learning environment. Members of the learner group can share the same attributes, such as the same department or job code. Learner groups are used to control access to and enrollment in learning activities and programs. They are also used to perform group enrollments and mass enrollments in the back office.

learning components In PeopleSoft Enterprise Learning Management, the foundational building blocks of learning activities. PeopleSoft Enterprise Learning Management supports six basic types of learning components: web-based, session, webcast, test, survey, and assignment. One or more of these learning component types compose a single learning activity.

learning environment In PeopleSoft Enterprise Learning Management, identifies a set of categories and catalog items that can be made available to learner groups. Also defines the default values that are assigned to the learning activities and programs that are created within a particular learning environment. Learning environments provide a way to partition the catalog so that learners see only those items that are relevant to them.

learning history In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's completed learning activities and programs.

lease In PeopleSoft Real Estate Management, a legally binding agreement between a landlord and a tenant, where the tenant rents all or part of a physical property from the landlord.

lease abstract In PeopleSoft Real Estate Management, a summarized version of the complete lease contract with only the important terms. The lease abstract usually fits on one page and does not include legal terminology.

ledger mapping You use ledger mapping to relate expense data from general ledger accounts to resource objects. Multiple ledger line items can be mapped to one or more resource IDs. You can also use ledger mapping to map dollar amounts (referred to as *rates*) to business units. You can map the amounts in two different ways: an actual amount that represents actual costs of the accounting period, or a budgeted amount that can be used to calculate the capacity rates as well as budgeted model results. In PeopleSoft Enterprise Warehouse, you can map general ledger accounts to the EW Ledger table.

library section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan (or template) and that is available for other plans to share. Changes to a library section are reflected in all plans that use it.

line In PeopleSoft Strategic Sourcing, an individual item or service upon which there can be a bid.

Copyright © 1988-2007, Oracle. All rights reserved.

linked section In PeopleSoft Enterprise Incentive Management, a section that is defined in a plan template but appears in a plan. Changes to linked sections propagate to plans using that section.

linked variable In PeopleSoft Enterprise Incentive Management, a variable that is defined and maintained in a plan template and that also appears in a plan. Changes to linked variables propagate to plans using that variable.

LMS Abbreviation for *learning management system.* In PeopleSoft Enterprise Campus Solutions, LMS is a PeopleSoft Enterprise Student Records feature that provides a common set of interoperability standards that enable the sharing of instructional content and data between learning and administrative environments.

load In PeopleSoft Enterprise Inventory, identifies a group of goods that are shipped together. Load management is a feature of PeopleSoft Enterprise Inventory that is used to track the weight, the volume, and the destination of a shipment.

local functionality In PeopleSoft Enterprise HRMS, the set of information that is available for a specific country. You can access this information when you click the appropriate country flag in the global window, or when you access it by a local country menu.

location Locations enable you to indicate the different types of addresses—for a company, for example, one address to receive bills, another for shipping, a third for postal deliveries, and a separate street address. Each address has a different location number. The primary location—indicated by a *1*—is the address you use most often and may be different from the main address.

logistical task In PeopleSoft Enterprise Services Procurement, an administrative task that is related to hiring a service provider. Logistical tasks are linked to the service type on the work order so that different types of services can have different logistical tasks. Logistical tasks include both preapproval tasks (such as assigning a new badge or ordering a new laptop) and postapproval tasks (such as scheduling orientation or setting up the service provider email). The logistical tasks can be mandatory or optional. Mandatory preapproval tasks must be completed before the work order is approved. Mandatory postapproval tasks, on the other hand, must be completed before a work order is released to a service provider.

market template In PeopleSoft Enterprise Incentive Management, additional functionality that is specific to a given market or industry and is built on top of a product category.

mass change In PeopleSoft Enterprise Campus Solutions, mass change is a SQL generator that can be used to create specialized functionality. Using mass change, you can set up a series of Insert, Update, or Delete SQL statements to perform business functions that are specific to the institution.

See also *3C engine.*

match group In PeopleSoft Enterprise Receivables, a group of receivables items and matching offset items. The system creates match groups by using user-defined matching criteria for selected field values.

MCF server Abbreviation for *PeopleSoft MultiChannel Framework server.* Comprises the universal queue server and the MCF log server. Both processes are started when *MCF Servers* is selected in an application server domain configuration.

merchandising activity In PeopleSoft Enterprise Promotions Management, a specific discount type that is associated with a trade promotion (such as off-invoice, billback or rebate, or lump-sum payment) that defines the performance that is required to receive the discount. In the industry, you may know this as an offer, a discount, a merchandising event, an event, or a tactic.

Copyright © 1988-2007, Oracle. All rights reserved.

meta-SQL Meta-SQL constructs expand into platform-specific SQL substrings. They are used in functions that pass SQL strings, such as in SQL objects, the SQLExec function, and PeopleSoft Application Engine programs.

metastring Metastrings are special expressions included in SQL string literals. The metastrings, prefixed with a percent (%) symbol, are included directly in the string literals. They expand at run time into an appropriate substring for the current database platform.

multibook In PeopleSoft Enterprise General Ledger, multiple ledgers having multiple-base currencies that are defined for a business unit, with the option to post a single transaction to all base currencies (all ledgers) or to only one of those base currencies (ledgers).

multicurrency The ability to process transactions in a currency other than the business unit's base currency.

national allowance In PeopleSoft Enterprise Promotions Management, a promotion at the corporate level that is funded by nondiscretionary dollars. In the industry, you may know this as a national promotion, a corporate promotion, or a corporate discount.

NDP Abbreviation for *Non-Discountable Products.* Term used for products or groups of products where the selling price cannot be decreased.

need In PeopleSoft Enterprise Campus Solutions, the difference between the cost of attendance (COA) and the expected family contribution (EFC). It is the gap between the cost of attending the school and the student's resources. The financial aid package is based on the amount of financial need. The process of determining a student's need is called *need analysis.*

node-oriented tree A tree that is based on a detail structure, but the detail values are not used.

Optimization Engine A PeopleTools component that Strategic Sourcing leverages to evaluate bids and determine an ideal award allocation. The award recommendation is based on maximizing the value while adhering to purchasing and company objectives and constraints.

pagelet Each block of content on the home page is called a pagelet. These pagelets display summary information within a small rectangular area on the page. The pagelet provide users with a snapshot of their most relevant PeopleSoft Enterprise and non-PeopleSoft Enterprise content.

participant In PeopleSoft Enterprise Incentive Management, participants are recipients of the incentive compensation calculation process.

participant object Each participant object may be related to one or more compensation objects.

See also *compensation object.*

partner A company that supplies products or services that are resold or purchased by the enterprise.

pay cycle In PeopleSoft Enterprise Payables, a set of rules that define the criteria by which it should select scheduled payments for payment creation.

payment shuffle In PeopleSoft Enterprise Campus Solutions, a process allowing payments that have been previously posted to a student's account to be automatically reapplied when a higher priority payment is posted or the payment allocation definition is changed.

pending item In PeopleSoft Enterprise Receivables, an individual receivable (such as an invoice, a credit memo, or a write-off) that has been entered in or created by the system, but hasn't been posted.

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleCode PeopleCode is a proprietary language, executed by the PeopleSoft Enterprise component processor. PeopleCode generates results based on existing data or user actions. By using various tools provided with PeopleTools, external services are available to all PeopleSoft Enterprise applications wherever PeopleCode can be executed.

PeopleCode event See *event.*

PeopleSoft Pure Internet Architecture The fundamental architecture on which PeopleSoft 8 applications are constructed, consisting of a relational database management system (RDBMS), an application server, a web server, and a browser.

performance measurement In PeopleSoft Enterprise Incentive Management, a variable used to store data (similar to an aggregator, but without a predefined formula) within the scope of an incentive plan. Performance measures are associated with a plan calendar, territory, and participant. Performance measurements are used for quota calculation and reporting.

period context In PeopleSoft Enterprise Incentive Management, because a participant typically uses the same compensation plan for multiple periods, the period context associates a plan context with a specific calendar period and fiscal year. The period context references the associated plan context, thus forming a chain. Each plan context has a corresponding set of period contexts.

person of interest A person about whom the organization maintains information but who is not part of the workforce.

personal portfolio In PeopleSoft Enterprise Campus Solutions, the user-accessible menu item that contains an individual's name, address, telephone number, and other personal information.

phase A level 1 task, meaning that if a task had subtasks, the level 1 task would be considered the phase.

pickup quantity The product quantity that the customer is taking with them from the counter sales environment.

plan In PeopleSoft Enterprise Sales Incentive Management, a collection of allocation rules, variables, steps, sections, and incentive rules that instruct the PeopleSoft Enterprise Incentive Management engine in how to process transactions.

plan context In PeopleSoft Enterprise Incentive Management, correlates a participant with the compensation plan and node to which the participant is assigned, enabling the PeopleSoft Enterprise Incentive Management system to find anything that is associated with the node and that is required to perform compensation processing. Each participant, node, and plan combination represents a unique plan context—if three participants are on a compensation structure, each has a different plan context. Configuration plans are identified by plan contexts and are associated with the participants that refer to them.

plan template In PeopleSoft Enterprise Incentive Management, the base from which a plan is created. A plan template contains common sections and variables that are inherited by all plans that are created from the template. A template may contain steps and sections that are not visible in the plan definition.

planned learning In PeopleSoft Enterprise Learning Management, a self-service repository for all of a learner's planned learning activities and programs.

planning instance In PeopleSoft Enterprise Supply Planning, a set of data (business units, items, supplies, and demands) constituting the inputs and outputs of a supply plan.

population In PeopleSoft Enterprise Campus Solutions, the middle level of the three-level classification structure that you define in PeopleSoft Enterprise Recruiting and

Copyright © 1988-2007, Oracle. All rights reserved.

	Admissions for enrollment management. You can define a population level, link it to other levels, and set enrollment target numbers for it. See also *division* and *cohort.*
portal registry	In PeopleSoft Enterprise applications, the portal registry is a tree-like structure in which content references are organized, classified, and registered. It is a central repository that defines both the structure and content of a portal through a hierarchical, tree-like structure of folders useful for organizing and securing content references.
predecessor task	A task that you must complete before you start another task.
price breaks	In PeopleSoft Strategic Sourcing, a price discount or surcharge that a bidder may apply based on the quantity awarded.
price components	In PeopleSoft Strategic Sourcing, the various components, such as material costs, labor costs, shipping costs, and so on that make up the overall bid price.
price list	Enables you to select products and conditions for which the price list applies to a transaction. During a transaction, the system either determines the product price based on the predefined search hierarchy for the transaction or uses the product's lowest price on any associated, active price lists. This price is used as the basis for any further discounts and surcharges.
price rule	The conditions that must be met for adjustments to be applied to the base price. Multiple rules can apply when conditions of each rule are met.
price rule conditions	Conditions that select the price-by fields, the values for the price-by fields, and the operator that determines how the price-by fields relate to the transaction.
price rule key	The fields that are available to define price rule conditions (which are used to match a transaction) on the price rule.
primacy number	In PeopleSoft Enterprise Campus Solutions, a number that the system uses to prioritize financial aid applications when students are enrolled in multiple academic careers and academic programs at the same time. The Consolidate Academic Statistics process uses the primacy number indicated for both the career and program at the institutional level to determine a student's primary career and program. The system also uses the number to determine the primary student attribute value that is used when you extract data to report on cohorts. The lowest number takes precedence.
primary name type	In PeopleSoft Enterprise Campus Solutions, the name type that is used to link the name stored at the highest level within the system to the lower-level set of names that an individual provides.
process category	In PeopleSoft Process Scheduler, processes that are grouped for server load balancing and prioritization.
process group	In PeopleSoft Enterprise Financials, a group of application processes (performed in a defined order) that users can initiate in real time, directly from a transaction entry page.
process definition	Process definitions define each run request.
process instance	A unique number that identifies each process request. This value is automatically incremented and assigned to each requested process when the process is submitted to run.
process job	You can link process definitions into a job request and process each request serially or in parallel. You can also initiate subsequent processes based on the return code from each prior request.

Copyright © 1988-2007, Oracle. All rights reserved.

process request A single run request, such as a Structured Query Report (SQR), a COBOL or Application Engine program, or a Crystal report that you run through PeopleSoft Process Scheduler.

process run control A PeopleTools variable used to retain PeopleSoft Process Scheduler values needed at runtime for all requests that reference a run control ID. Do not confuse these with application run controls, which may be defined with the same run control ID, but only contain information specific to a given application process request.

product A PeopleSoft Enterprise or third-party product. PeopleSoft organizes its software products into product families and product lines. Interactive Services Repository contains information about every release of every product that PeopleSoft sells, as well as products from certified third-party companies. These products appear with the product name and release number.

product adds The pricing functionality where buying product A gets product B for free or at a price (formerly referred to as *giveaways*).

product bidding In PeopleSoft Strategic Sourcing, the placing of a bid on behalf of the bidder, up or down to the bidder's specified amount, so that the bidder can be the leading bidder.

product category In PeopleSoft Enterprise Incentive Management, indicates an application in the PeopleSoft Enterprise Incentive Management suite of products. Each transaction in the PeopleSoft Enterprise Incentive Management system is associated with a product category.

product family A group of products that are related by common functionality. The family names that can be searched using Interactive Service Repository are Oracle's PeopleSoft Enterprise, PeopleSoft EnterpriseOne, PeopleSoft World, and third-party, certified partners.

product line The name of a PeopleSoft Enterprise product line or the company name of a third-party certified partner. Integration Services Repository enables you to search for integration points by product line.

programs In PeopleSoft Enterprise Learning Management, a high-level grouping that guides the learner along a specific learning path through sections of catalog items. PeopleSoft Enterprise Learning Systems provides two types of programs—curricula and certifications.

progress log In PeopleSoft Enterprise Services Procurement, tracks deliverable-based projects. This is similar to the time sheet in function and process. The service provider contact uses the progress log to record and submit progress on deliverables. The progress can be logged by the activity that is performed, by the percentage of work that is completed, or by the completion of milestone activities that are defined for the project.

project transaction In PeopleSoft Enterprise Project Costing, an individual transaction line that represents a cost, time, budget, or other transaction row.

promotion In PeopleSoft Enterprise Promotions Management, a trade promotion, which is typically funded from trade dollars and used by consumer products manufacturers to increase sales volume.

prospects In PeopleSoft Enterprise Campus Solutions, students who are interested in applying to the institution.

In PeopleSoft Enterprise Contributor Relations, individuals and organizations that are most likely to make substantial financial commitments or other types of commitments to the institution.

proxy bidding In PeopleSoft Strategic Sourcing, the placing of a bid on behalf of the bidder, up or down to the bidder's specified amount, so that the bidder can be the leading bidder.

Copyright © 1988-2007, Oracle. All rights reserved.

publishing In PeopleSoft Enterprise Incentive Management, a stage in processing that makes incentive-related results available to participants.

rating components In PeopleSoft Enterprise Campus Solutions, variables used with the Equation Editor to retrieve specified populations.

record group A set of logically and functionally related control tables and views. Record groups help enable TableSet sharing, which eliminates redundant data entry. Record groups ensure that TableSet sharing is applied consistently across all related tables and views.

record input VAT flag Abbreviation for *record input value-added tax flag.* Within PeopleSoft Enterprise Purchasing, Payables, and General Ledger, this flag indicates that you are recording input VAT on the transaction. This flag, in conjunction with the record output VAT flag, is used to determine the accounting entries created for a transaction and to determine how a transaction is reported on the VAT return. For all cases within Purchasing and Payables where VAT information is tracked on a transaction, this flag is set to Yes. This flag is not used in PeopleSoft Enterprise Order Management, Billing, or Receivables, where it is assumed that you are always recording only output VAT, or in PeopleSoft Enterprise Expenses, where it is assumed that you are always recording only input VAT.

record output VAT flag Abbreviation for *record output value-added tax flag.*

See *record input VAT flag.*

recname The name of a record that is used to determine the associated field to match a value or set of values.

recognition In PeopleSoft Enterprise Campus Solutions, the recognition type indicates whether the PeopleSoft Enterprise Contributor Relations donor is the primary donor of a commitment or shares the credit for a donation. Primary donors receive hard credit that must total 100 percent. Donors that share the credit are given soft credit. Institutions can also define other share recognition-type values such as memo credit or vehicle credit.

reference data In PeopleSoft Enterprise Sales Incentive Management, system objects that represent the sales organization, such as territories, participants, products, customers, and channels.

reference object In PeopleSoft Enterprise Incentive Management, this dimension-type object further defines the business. Reference objects can have their own hierarchy (for example, product tree, customer tree, industry tree, and geography tree).

reference transaction In commitment control, a reference transaction is a source transaction that is referenced by a higher-level (and usually later) source transaction, in order to automatically reverse all or part of the referenced transaction's budget-checked amount. This avoids duplicate postings during the sequential entry of the transaction at different commitment levels. For example, the amount of an encumbrance transaction (such as a purchase order) will, when checked and recorded against a budget, cause the system to concurrently reference and relieve all or part of the amount of a corresponding pre-encumbrance transaction, such as a purchase requisition.

regional sourcing In PeopleSoft Enterprise Purchasing, provides the infrastructure to maintain, display, and select an appropriate vendor and vendor pricing structure that is based on a regional sourcing model where the multiple ship to locations are grouped. Sourcing may occur at a level higher than the ship to location.

relationship object In PeopleSoft Enterprise Incentive Management, these objects further define a compensation structure to resolve transactions by establishing associations between compensation objects and business objects.

remote data source data Data that is extracted from a separate database and migrated into the local database.

Copyright © 1988-2007, Oracle. All rights reserved.

REN server Abbreviation for *real-time event notification server* in PeopleSoft MultiChannel Framework.

requester In PeopleSoft Enterprise eSettlements, an individual who requests goods or services and whose ID appears on the various procurement pages that reference purchase orders.

reservations In PeopleSoft Expenses, travel reservations that have been placed with the travel vendor.

reversal indicator In PeopleSoft Enterprise Campus Solutions, an indicator that denotes when a particular payment has been reversed, usually because of insufficient funds.

RFI event In PeopleSoft Strategic Sourcing, a request for information.

RFx event In PeopleSoft Strategic Sourcing, a request for proposal or request for a quote event when bidders submit their overall best bids and during which bidders do not actively compete against one another.

role Describes how people fit into PeopleSoft Workflow. A role is a class of users who perform the same type of work, such as clerks or managers. Your business rules typically specify what user role needs to do an activity.

role user A PeopleSoft Workflow user. A person's role user ID serves much the same purpose as a user ID does in other parts of the system. PeopleSoft Workflow uses role user IDs to determine how to route worklist items to users (through an email address, for example) and to track the roles that users play in the workflow. Role users do not need PeopleSoft user IDs.

roll up In a tree, to roll up is to total sums based on the information hierarchy.

run control A run control is a type of online page that is used to begin a process, such as the batch processing of a payroll run. Run control pages generally start a program that manipulates data.

run control ID A unique ID to associate each user with his or her own run control table entries.

run-level context In PeopleSoft Enterprise Incentive Management, associates a particular run (and batch ID) with a period context and plan context. Every plan context that participates in a run has a separate run-level context. Because a run cannot span periods, only one run-level context is associated with each plan context.

saved bid In PeopleSoft Strategic Sourcing, a bid that has been created but not submitted. Only submitted bids are eligible for award.

score In PeopleSoft Strategic Sourcing, the numerical sum of answers (percentages) to bid factors on an event. Scores appear only to bidders on auction events.

SCP SCBM XML message Abbreviation for *Supply Chain Planning Supply Chain Business Modeler Extensible Markup Language message.* Supply Chain Business Modeler uses XML as the format for all data that it imports and exports.

search query You use this set of objects to pass a query string and operators to the search engine. The search index returns a set of matching results with keys to the source documents.

search/match In PeopleSoft Enterprise Campus Solutions and PeopleSoft Enterprise Human Resources Management Solutions, a feature that enables you to search for and identify duplicate records in the database.

seasonal address In PeopleSoft Enterprise Campus Solutions, an address that recurs for the same length of time at the same time of year each year until adjusted or deleted.

Copyright © 1988-2007, Oracle. All rights reserved.

section In PeopleSoft Enterprise Incentive Management, a collection of incentive rules that operate on transactions of a specific type. Sections enable plans to be segmented to process logical events in different sections.

security event In commitment control, security events trigger security authorization checking, such as budget entries, transfers, and adjustments; exception overrides and notifications; and inquiries.

sell event In PeopleSoft Strategic Sourcing, for event creators, the sale of goods or services most typically associated with forward auctions. For bidders, the purchase of goods or services.

serial genealogy In PeopleSoft Enterprise Manufacturing, the ability to track the composition of a specific, serial-controlled item.

serial in production In PeopleSoft Enterprise Manufacturing, enables the tracing of serial information for manufactured items. This is maintained in the Item Master record.

service impact In PeopleSoft Enterprise Campus Solutions, the resulting action triggered by a service indicator. For example, a service indicator that reflects nonpayment of account balances by a student might result in a service impact that prohibits registration for classes.

service indicator In PeopleSoft Enterprise Campus Solutions, indicates services that may be either withheld or provided to an individual. Negative service indicators indicate holds that prevent the individual from receiving specified services, such as check-cashing privileges or registration for classes. Positive service indicators designate special services that are provided to the individual, such as front-of-line service or special services for disabled students.

session In PeopleSoft Enterprise Campus Solutions, time elements that subdivide a term into multiple time periods during which classes are offered. In PeopleSoft Enterprise Contributor Relations, a session is the means of validating gift, pledge, membership, or adjustment data entry . It controls access to the data entered by a specific user ID. Sessions are balanced, queued, and then posted to the institution's financial system. Sessions must be posted to enter a matching gift or pledge payment, to make an adjustment, or to process giving clubs or acknowledgements.

In PeopleSoft Enterprise Learning Management, a single meeting day of an activity (that is, the period of time between start and finish times within a day). The session stores the specific date, location, meeting time, and instructor. Sessions are used for scheduled training.

session template In PeopleSoft Enterprise Learning Management, enables you to set up common activity characteristics that may be reused while scheduling a PeopleSoft Enterprise Learning Management activity—characteristics such as days of the week, start and end times, facility and room assignments, instructors, and equipment. A session pattern template can be attached to an activity that is being scheduled. Attaching a template to an activity causes all of the default template information to populate the activity session pattern.

setup relationship In PeopleSoft Enterprise Incentive Management, a relationship object type that associates a configuration plan with any structure node.

share driver expression In PeopleSoft Enterprise Business Planning, a named planning method similar to a driver expression, but which you can set up globally for shared use within a single planning application or to be shared between multiple planning applications through PeopleSoft Enterprise Warehouse.

short-term customer A customer not in the system who is entered during sales order entry using a template.

Copyright © 1988-2007, Oracle. All rights reserved.

single signon With single signon, users can, after being authenticated by a PeopleSoft Enterprise application server, access a second PeopleSoft Enterprise application server without entering a user ID or password.

source key process In PeopleSoft Enterprise Campus Solutions, a process that relates a particular transaction to the source of the charge or financial aid. On selected pages, you can drill down into particular charges.

source transaction In commitment control, any transaction generated in a PeopleSoft Enterprise or third-party application that is integrated with commitment control and which can be checked against commitment control budgets. For example, a pre-encumbrance, encumbrance, expenditure, recognized revenue, or collected revenue transaction.

sourcing objective For constraints, the option to designate whether a business rule is required (mandatory) or is only recommended (target).

speed key See *communication key.*

SpeedChart A user-defined shorthand key that designates several ChartKeys to be used for voucher entry. Percentages can optionally be related to each ChartKey in a SpeedChart definition.

SpeedType A code representing a combination of ChartField values. SpeedTypes simplify the entry of ChartFields commonly used together.

staging A method of consolidating selected partner offerings with the offerings from the enterprise's other partners.

standard letter code In PeopleSoft Enterprise Campus Solutions, a standard letter code used to identify each letter template available for use in mail merge functions. Every letter generated in the system must have a standard letter code identification.

statutory account Account required by a regulatory authority for recording and reporting financial results. In PeopleSoft Enterprise, this is equivalent to the Alternate Account (ALTACCT) ChartField.

step In PeopleSoft Enterprise Sales Incentive Management, a collection of sections in a plan. Each step corresponds to a step in the job run.

storage level In PeopleSoft Enterprise Inventory, identifies the level of a material storage location. Material storage locations are made up of a business unit, a storage area, and a storage level. You can set up to four storage levels.

subcustomer qualifier A value that groups customers into a division for which you can generate detailed history, aging, events, and profiles.

Summary ChartField You use summary ChartFields to create summary ledgers that roll up detail amounts based on specific detail values or on selected tree nodes. When detail values are summarized using tree nodes, summary ChartFields must be used in the summary ledger data record to accommodate the maximum length of a node name (20 characters).

summary ledger An accounting feature used primarily in allocations, inquiries, and PS/nVision reporting to store combined account balances from detail ledgers. Summary ledgers increase speed and efficiency of reporting by eliminating the need to summarize detail ledger balances each time a report is requested. Instead, detail balances are summarized in a background process according to user-specified criteria and stored on summary ledgers. The summary ledgers are then accessed directly for reporting.

summary time period In PeopleSoft Enterprise Business Planning, any time period (other than a base time period) that is an aggregate of other time periods, including other summary time periods and base time periods, such as quarter and year total.

Copyright © 1988-2007, Oracle. All rights reserved.

summary tree A tree used to roll up accounts for each type of report in summary ledgers. Summary trees enable you to define trees on trees. In a summary tree, the detail values are really nodes on a detail tree or another summary tree (known as the *basis* tree). A summary tree structure specifies the details on which the summary trees are to be built.

syndicate To distribute a production version of the enterprise catalog to partners.

system function In PeopleSoft Enterprise Receivables, an activity that defines how the system generates accounting entries for the general ledger.

system source The system source identifies the source of a transaction row in the database. For example, a transaction that originates in PeopleSoft Enterprise Expenses contains a system source code of BEX (Expenses Batch).

When PeopleSoft Enterprise Project Costing prices the source transaction row for billing, the system creates a new row with a system source code of PRP (Project Costing pricing), which represents the system source of the new row. System source codes can identify sources that are internal or external to the PeopleSoft Enterprise system. For example, processes that import data from Microsoft Project into PeopleSoft Enterprise applications create transaction rows with a source code of MSP (Microsoft Project).

TableSet A means of sharing similar sets of values in control tables, where the actual data values are different but the structure of the tables is the same.

TableSet sharing Shared data that is stored in many tables that are based on the same TableSets. Tables that use TableSet sharing contain the SETID field as an additional key or unique identifier.

target currency The value of the entry currency or currencies converted to a single currency for budget viewing and inquiry purposes.

task A deliverable item on the detailed sourcing plan.

tax authority In PeopleSoft Enterprise Campus Solutions, a user-defined element that combines a description and percentage of a tax with an account type, an item type, and a service impact.

template A template is HTML code associated with a web page. It defines the layout of the page and also where to get HTML for each part of the page. In PeopleSoft Enterprise, you use templates to build a page by combining HTML from a number of sources. For a PeopleSoft Enterprise portal, all templates must be registered in the portal registry, and each content reference must be assigned a template.

tenant In PeopleSoft Real Estate Management, an entity that leases real estate from a landlord.

territory In PeopleSoft Enterprise Sales Incentive Management, hierarchical relationships of business objects, including regions, products, customers, industries, and participants.

third party A company or vendor that has extensive PeopleSoft Enterprise product knowledge and whose products and integrations have been certified and are compatible with PeopleSoft Enterprise applications.

tiered pricing Enables different portions of a schedule to be priced differently from one another.

time span A relative period, such as year-to-date or current period, that various PeopleSoft General Ledger functions and reports can use when a rolling time frame, rather than a specific date, is required.

total cost In PeopleSoft Strategic Sourcing, the estimated dollar cost (sum of real price dollars and potential “soft” or non-price dollars) of a particular award approach.

Copyright © 1988-2007, Oracle. All rights reserved.

travel group In PeopleSoft Expenses, the organization's travel rules and polices that are associated with specific business units, departments, or employees. You must define at least one travel group when setting up the PeopleSoft Expenses travel feature. You must define and associate at least one travel group with a travel vendor.

travel partner In PeopleSoft Expenses, the travel vendor with which the organization has a contractual relationship.

3C engine Abbreviation for *Communications, Checklists, and Comments engine.* In PeopleSoft Enterprise Campus Solutions, the 3C engine enables you to automate business processes that involve additions, deletions, and updates to communications, checklists, and comments. You define events and triggers to engage the engine, which runs the mass change and processes the 3C records (for individuals or organizations) immediately and automatically from within business processes.

3C group Abbreviation for *Communications, Checklists, and Comments group.* In PeopleSoft Enterprise Campus Solutions, a method of assigning or restricting access privileges. A 3C group enables you to group specific communication categories, checklist codes, and comment categories. You can then assign the group inquiry-only access or update access, as appropriate.

trace usage In PeopleSoft Enterprise Manufacturing, enables the control of which components will be traced during the manufacturing process. Serial- and lot-controlled components can be traced. This is maintained in the Item Master record.

transaction allocation In PeopleSoft Enterprise Incentive Management, the process of identifying the owner of a transaction. When a raw transaction from a batch is allocated to a plan context, the transaction is duplicated in the PeopleSoft Enterprise Incentive Management transaction tables.

transaction state In PeopleSoft Enterprise Incentive Management, a value assigned by an incentive rule to a transaction. Transaction states enable sections to process only transactions that are at a specific stage in system processing. After being successfully processed, transactions may be promoted to the next transaction state and "picked up" by a different section for further processing.

Translate table A system edit table that stores codes and translate values for the miscellaneous fields in the database that do not warrant individual edit tables of their own.

tree The graphical hierarchy in PeopleSoft Enterprise systems that displays the relationship between all accounting units (for example, corporate divisions, projects, reporting groups, account numbers) and determines roll-up hierarchies.

tuition lock In PeopleSoft Enterprise Campus Solutions, a feature in the Tuition Calculation process that enables you to specify a point in a term after which students are charged a minimum (or *locked*) fee amount. Students are charged the locked fee amount even if they later drop classes and take less than the normal load level for that tuition charge.

unclaimed transaction In PeopleSoft Enterprise Incentive Management, a transaction that is not claimed by a node or participant after the allocation process has completed, usually due to missing or incomplete data. Unclaimed transactions may be manually assigned to the appropriate node or participant by a compensation administrator.

universal navigation header Every PeopleSoft Enterprise portal includes the universal navigation header, intended to appear at the top of every page as long as the user is signed on to the portal. In addition to providing access to the standard navigation buttons (like Home, Favorites, and signoff) the universal navigation header can also display a welcome message for each user.

update access In PeopleSoft Enterprise Campus Solutions, a type of security access that permits the user to edit and update data.

Copyright © 1988-2007, Oracle. All rights reserved.

	See also *inquiry access*.
user interaction object	In PeopleSoft Enterprise Sales Incentive Management, used to define the reporting components and reports that a participant can access in his or her context. All PeopleSoft Enterprise Sales Incentive Management user interface objects and reports are registered as user interaction objects. User interaction objects can be linked to a compensation structure node through a compensation relationship object (individually or as groups).
variable	In PeopleSoft Enterprise Sales Incentive Management, the intermediate results of calculations. Variables hold the calculation results and are then inputs to other calculations. Variables can be plan variables that persist beyond the run of an engine or local variables that exist only during the processing of a section.
VAT exception	Abbreviation for *value-added tax exception.* A temporary or permanent exemption from paying VAT that is granted to an organization. This terms refers to both VAT exoneration and VAT suspension.
VAT exempt	Abbreviation for *value-added tax exempt.* Describes goods and services that are not subject to VAT. Organizations that supply exempt goods or services are unable to recover the related input VAT. This is also referred to as exempt without recovery.
VAT exoneration	Abbreviation for *value-added tax exoneration.* An organization that has been granted a permanent exemption from paying VAT due to the nature of that organization.
VAT suspension	Abbreviation for *value-added tax suspension.* An organization that has been granted a temporary exemption from paying VAT.
warehouse	A PeopleSoft Enterprise data warehouse that consists of predefined ETL maps, data warehouse tools, and DataMart definitions.
weight or weighting	In PeopleSoft Strategic Sourcing, how important the line or question is to the overall event. Weighting is used to score and analyze bids. For RFx and RFI events, weightings may or may not appear to bidders.
work order	In PeopleSoft Enterprise Services Procurement, enables an enterprise to create resource-based and deliverable-based transactions that specify the basic terms and conditions for hiring a specific service provider. When a service provider is hired, the service provider logs time or progress against the work order.
worker	A person who is part of the workforce; an employee or a contingent worker.
workset	A group of people and organizations that are linked together as a set. You can use worksets to simultaneously retrieve the data for a group of people and organizations and work with the information on a single page.
worksheet	A way of presenting data through a PeopleSoft Enterprise Business Analysis Modeler interface that enables users to do in-depth analysis using pivoting tables, charts, notes, and history information.
worklist	The automated to-do list that PeopleSoft Workflow creates. From the worklist, you can directly access the pages you need to perform the next action, and then return to the worklist for another item.
XML link	The XML Linking language enables you to insert elements into XML documents to create a links between resources.
XML schema	An XML definition that standardizes the representation of application messages, component interfaces, or business interlinks.
XPI	Abbreviation for *eXtended Process Integrator.* PeopleSoft XPI is the integration infrastructure that enables both real-time and batch communication with JD Edwards EnterpriseOne applications.

Copyright © 1988-2007, Oracle. All rights reserved.

yield by operation In PeopleSoft Enterprise Manufacturing, the ability to plan the loss of a manufactured item on an operation-by-operation basis.

zero-rated VAT Abbreviation for *zero-rated value-added tax.* A VAT transaction with a VAT code that has a tax percent of zero. Used to track taxable VAT activity where no actual VAT amount is charged. Organizations that supply zero-rated goods and services can still recover the related input VAT. This is also referred to as exempt with recovery.

Copyright © 1988-2007, Oracle. All rights reserved.

Copyright © 1988-2007, Oracle. All rights reserved.

Index

Numerics/Symbols

1159Separator property 1531
2359Separator property 1532
2D bar chart 323
2D histogram chart 323
2D horizontal bar chart 324
2D horizontal percent bar chart 325
2D horizontal stacked bar chart 324
2D line chart 325
2D percent bar 326
2D pie 327
2D scatter 327
  uses 327
2D stacked bar 328
3D bar 328
3D percent bar 329
3D pie 329
3D stacked bar 330

A

AbsoluteContentURL property
  Content Reference class 1072
  Content Reference Links class 1102
AbsolutePortalURL property
  Content Reference class 1072
  Content Reference Links class 1103
abstract
  declaring methods/properties 164
  understanding base classes 184
AcceptedTaskList property
  AgentPhysQueueTasks class 1724
  PhysicalQueue class 1730
actiondttm property 903
ActionName property 769
ActivateModel method 979
ActivateObjective method 979
Active property
  DataSourceDefn class 1815
ActiveNode property 1046
ActiveRowCount property 1445
Add method 1630
AddAllFields method 1284
AddArgument method 117
AddAttachment method
  IBInfo class 802
  MCFOutboundEmail class 688
AddAttribute method
  Incoming Business Interlink class 303
  XmlNode class 1765
AddAttributeNS method 1766
AddBody method 1546
AddBodyPart method 686
AddCDataSection method 1767
AddComment method
  Incoming Business Interlink class 304
  XmlNode class 1768
AddConnectorProperties method 819
AddCriteria method 1284
AddCube method 48
AddCubeCollection method 49
AddDimension method 49
AddDistributionOption method
  PostReport class 1186
  ProcessRequest class 1203
AddDoc method 261
AddDropDownItem method 493
AddElement method 1769
AddElementNS method 1770
AddEntityReference method 1771
AddEnvelope method 1547
AddExplicitDimensionSet method 50
AddExpr1Expression method 1315
AddExpr1Field method 1315
AddExpr2Expression method 1316
AddExpr2Field1 method 1316
AddExpr2Field2 method 1317
AddExpr2List method 1318
AddExpr2Subquery method 1318
AddExpression method 1285
AddFault method 1550
AddHavingCriteria method 1285
AddHeader method
  MCFBodyPart class 644
  MCFOutboundEmail class 689
  SOAPDoc class 1551
adding
  RuleExpression 102, 112
AddInputRow method 262
additional documentation cxiv
AddListValue method 1334

Copyright © 1988-2007, Oracle. All rights reserved.

AddMember method 21
AddMethod method 1552
AddModel method 140
AddNextDoc method 264
AddNode method 1772
AddOrganizer method 50
AddParm method 1554
AddProcessInstruction method
  Incoming Business Interlink class 305
  XmlNode class 1773
AddPrompt method 1264
AddQueryOutputField method 1286
AddQueryRecord method 1286
AddQuerySelect method 1263
AddQuerySelectedField method 1287
AddQueryStringArg method 820
AddRecord method 141
AddReference method 1595
AddReferenceType method 1595
AddRuleExpression method
  ExpressionBlock class 112
  RuleDefn class 102
AddSearchFieldCriteria method 1853
AddStep method 5
AddText method
  Incoming Business Interlink class 306
  XmlNode class 1774
AddTranslateExpression method 1305
AddTranslateField method 1305
AddUnion method 1293
AddUser method 724
AddUserFunction method 51
AddValue method 266
AddXMLData method 1861
AddXSDSchema method 1862
AdvancedSearchQueries method 1249
AdvancedSearchRecords method 1252
AESection
  class 1
    See Also AESection class
  objects 2
    See Also AESection objects
AESection class
  accessing 2
  example 3
  list of functions, methods, properties, returns 1901
  methods 5
    See Also AESection class methods
  properties 10
    See Also AESection class properties
  understanding 1
AESection class methods
  AddStep 5
  Close 6
  Open 6
  Save 8
  SetSQL 8
  SetTemplate 9
AESection class properties 10
AESection objects
  declaring/scope 4
  deleting steps 3
  opening/getting 2
Agent class
  list of methods, properties, returns 1991
  methods 1717
  properties 1719
    See Also Agent class properties
  understanding 1717
Agent class properties
  AgentID 1719
  AgentProps 1720
  AgentTasks 1720
  Buddy 1721
  Language 1721
  Name 1721
  NickName 1721
  PhysicalQueueID 1721
  TotalPhysicalQueues 1722
Agent constructor 1710
Agent property 1730
AgentID property
  Agent class 1719
  AgentPhysQueueProps class 1722
  AgentPhysQueueTasks class 1725
  Task class 1737
  TaskList class 1739
AgentPhysQueueProps class
  list of properties, returns 1992
  properties 1722
  understanding 1722
AgentPhysQueueProps constructor 1711
AgentPhysQueueTasks class
  list of methods, properties, returns 1992
  methods 1723
  properties 1724
  understanding 1723
AgentPhysQueueTasks constructor 1711
AgentProps property 1720

Copyright © 1988-2007, Oracle. All rights reserved.

agents
Agent class 1717
See Also Agent class
Agent constructor 1710
Agent property 1730
AgentPhysQueueProps class 1722, 1723
See Also AgentPhysQueueProps class; AgentPhysQueueTasks class
Mobile Agent 827
See Also Mobile Agent
AgentTasks property 1720
aggregate
calculating 84
dimension 81
getting persistent 90
mappings 87, 91
record name 96
sequence 75
setting 83
setting persistent 94
user function 75
Aggregate property
QueryExpression class 1331
QueryField class 1306
AggregateRecName property 96
AggregateSequence property 75
AggregationUserFunction property 75
algorithms
accessing PortalRegistry objects 1001
ConflictAlgorithm property 1600
merge algorithm 1590
using for conflicts 1589
AliasName property 769
AllChildCount property 1646
AllChildNodeCount property 1646
AllowAnyJoin property 1346
AllowDistinct property 1346
AllowExpressions property 1346
AllowHTTPS property 1499
AllowRecipientEntry property 1829
AllowSubqueries property 1347
AllowUnions property 1347
AllowViewerEntry property 1829
AllValues property 1676
AllValuesAudit property 1632
analytic calculation engine
running asynchronously 12
analytic calculation engine classes
built-in functions 14
calling built-in functions 113
creating expression blocks 112
data types 13
error handling 13
establishing hierarchies 12
referencing members 118
scope 13
Analytic Calculation Engine Metadata classes
constructor 47
creating 47
data types 45
error handling 45
importing 46
understanding 43
using 44
analytic grid
error handling 132
populating 134
analytic grid columns
specifying names 133
analytic grids 130
activating 135
displaying grid lines 136
displaying slicer 136
labels 136
scope 132
secondary pages 130
analytic instances
analytic type definition name 21
asynchronous status 14
class properties 21
copying 15
creating 916
deleting 16, 919, 936
determining presence in optimization metadata 941
getting ACE model 17
getting list of 938
ID property 21
inserting 939
loading 17
loading into analytic servers 917
loading into optimization engines 917
messages 21
running asynchronously 19
running synchronously 19
status 15
terminating 20
unloading 20

Copyright © 1988-2007, Oracle. All rights reserved.

analytic model
accessing
cube collection names 61
cube collections 61
cube names 62
cubes 60
dimension names 63
dimensions 62
explicit dimension set names 64
explicit dimension sets 63
iterations 73
messages 74
organizer names 65
organizers 64
trees 27
user function names 66
user functions 65
value changes 73
adding
cube collections 49
cubes 48
dimension members 21
dimensions 49
explicit dimension sets 50
organizer 50
user functions 51
attaching trees 22
calculating cubes 24
cell properties 25
checking validity 73
class methods 48
class properties 72
copying
cube collections 52
cubes 52
dimensions 53
explicit dimension sets 53
model definitions 54
user functions 54
creating 55, 124
deleting
cube collections 57
cubes 56
dimensions 57
explicit dimension sets 58
models 55
organizers 58
user functions 59
describing 72, 73
detaching trees 24
getting 60
getting a reference to 17
getting CubeCollection objects 25
getting members 26
getting the definition 47
messages property 29
metadata classes example 123
naming 74
recalculating loaded model 28
renaming
cube collections 68
cubes 67
definitions 67
dimensions 68
explicit dimension sets 69
organizers 70
user functions 70
resolving circular dependencies 74
saving 71
setting circular formula warnings 72
validating 72
analytic servers
loading analytic instances 917
analytic type classes
constructors 140
creating 139
error handling 138
example 152
getting the definition 140
importing 139
using 138
analytic type definitions
adding models 140
adding records 141
class properties 147
copying 142
creating 142
deleting 143
deleting analytic instances 919
deleting models 143
deleting records 144
GetModel method 144
GetModelNames method 145
GetRecord method 145
GetRecordNames method 146
instantiating 142, 144
name of 21
renaming 146
saving 147
using callback records 963

Copyright © 1988-2007, Oracle. All rights reserved.

analytic type objects
data types 138
scope 138
AnalyticGrid class
data type 132
GetColumn method 133
GetCubeCollection method 133
LoadData method 134
methods 132
properties 135
SetAnalyticInstance method 134
AnalyticInstance class
methods 14
properties 21
AnalyticModel class
methods 21
property 29
AnalyticModelDefn class
constructor 47
creating instance of 48
methods 48
properties 72
AnalyticType property 21
AnalyticTypeDefn class
constructor 140
methods 140
AnalyticTypeDefn class properties
AppClassPath property 147
Comments property 147
Description 148
Name 148
OwnerID 148
AnalyticTypeModelDefn class
properties 148
Name 148
Type 148
AnalyticTypeRecordDefn class
methods 149
properties 150
AOL instant messaging 723
API instantiation methods 1518
API property 1533
API Repository
accessing via PeopleCode 195
accessing via Visual Basic 199
understanding 193
API Repository classes
Bindings 202
See Also Bindings class
ClassInfo 207
See Also ClassInfo class
list of methods, properties, returns 2015
MethodInfo 209
See Also MethodInfo class
Namespaces 204
See Also Namespaces class
PropertyInfo 212
See Also PropertyInfo class
Repository 201
See Also Repository class
API Repository collections
Bindings collection 201
See Also Bindings collection
ClassInfo collection 205
See Also ClassInfo collection
MethodInfo collection 208
See Also MethodInfo collection
Namespaces collection 203
See Also Namespaces collection
PropertyInfo collection 210
See Also PropertyInfo collection
AppClassPath property 147
application class functions
Import 188
Method 190
application class language constructs
Class 186
Get 187
Interface 189
Set 190
application classes
Agent class 1717
See Also Agent class
AgentPhysQueueProps class 1722
See Also AgentPhysQueueProps class
AgentPhysQueueTasks class 1723
See Also AgentPhysQueueTasks class
BPEL 241
See Also BPEL classes
Broadcast class 1725
See Also Broadcast class
calling via Java 623
declaring 185
declaring abstract methods /properties 164
declaring constants 177
See Also constants
declaring external functions 168
declaring external interfaces 156

Copyright © 1988-2007, Oracle. All rights reserved.

declaring private instance-variables /methods 160
declaring protected properteis and methods 157
defining methods 162
designing base classes 183
downcasting 179
extending 156
functions 186
See Also application class functions
global variables 178
handling exceptions 180
See Also Exception class
importing names 170, 185
instance variables 178
language constructs 186
See Also application class language constructs
LogicalQueue class 1727
See Also LogicalQueue class
mail classes 629
See Also mail classes
MCFFactory class 1728
See Also MCFFactory class
naming 155, 156, 170
notification 879
See Also Notification classes
overriding properties 175
PhysicalQueue class 1728
See Also PhysicalQueue class
placing variable declarations 177
PrcsApi class 1228
referencing superclasses 185
scope 185
self-referencing 171
structure 154
superclass referencing 178
Task class 1733
See Also Task class
TaskList class 1738
See Also TaskList class
understanding 153
understanding collections 177
See Also PeopleCode collections
understanding comments 182
See Also comments
universal queue classes 1707
See Also universal queue classes
using 154
using properties 174
using variables 177
See Also variables
Util class 1740
See Also Util class
utility classes 184
Application Classes 1916
Application Engine
File Layout example 566
logging errors 1513
programs 916
See Also Application Engine programs
running optimization transactions 919
running PeopleCode 917
sections 1
See Also AESection class
using SQL objects 1570
using with Business Interlink 260
Application Engine programs
editing OPT_CALL 931
editing PT_OPTCALL 927
terminating 915, 916
using optimization PeopleCode 916
application fundamentals cxiii
application packages
naming 170
PT_BPEL 242
PT_MCF_MAIL 629
PT_WF_NOTIFICATION 880
PT_WF_WORKLIST 880
using for classes 153
application servers
managing 618
running optimization transactions 918
running PeopleCode 917
using optimization PeopleCode 915
ApplicationData property 1737
Approved property 1276
ApproveDtTm property 1276
ApproveOprId property 1276
ApprovePrivateQuery property 1347
ApprovePublicQuery property 1347
ApproveUserId property 1276
AppServerDomain property 810
AppsRelease property 1046
ArchiveAfter property 1829
arguments
adding 117
get method 118
Arguments property 210
Array class

Copyright © 1988-2007, Oracle. All rights reserved.

creating 215
functions, methods, properties, returns 1917
methods 223
See Also Array class methods
properties 238
understanding 215
Array class methods
Clone 223
Find 223
Get 224
Join 225
Next 227
Pop 228
Push 229
Replace 231
Reverse 232
Set 233
Shift 234
Sort 234
Subarray 236
Substitute 237
Unshift 238
Array class properties
Dimension 238
Len 239
array objects
creating empty 218
creating/populating multi-dimensional 219
declaring 222
flattening and promoting 221
populating 217
removing items 218
scope of 222
arrays
Array class 215
See Also Array class
copying between PeopleCode and Java 626
Java 626
removing elements 231
understanding 215
See Also array class
AssignedPagelet
class 1123
See Also AssignedPagelet class
collection 1124
See Also AssignedPagelet collection
Content Reference class AssignedPagelets property 1072
TabDefinition class AssignedPagelets property 1114
AssignedPagelet class
list of, properties, returns 2047
properties 1123
See Also AssignedPagelet class properties
understanding 1123
AssignedPagelet class properties
Column 1123
LayoutBehavior 1123
PageletName 1124
Row 1124
AssignedPagelet collection
list of methods, properties, returns 2048
methods 1124
See Also AssignedPagelet collection methods
properties 1127
understanding 1124
AssignedPagelet collection methods
DeleteItem 1124
First 1125
InsertItem 1125
ItemByName 1126
Next 1127
AssignedPagelets property
Content Reference class 1072
TabDefinition class 1114
AssignedTaskList property
AgentPhysQueueTasks class 1724
PhysicalQueue class 1730
assignment class
creating 104
Expression property 105
GenerateRule method 104
method 104
properties 105
Variable property 105
AsyncFFSend class
method 244
understanding 244
AsyncFFSend class method
OnRequestSend 244
AsyncFFSend constructor 243
asynchronous mode
loading analytic instances 917
running optimization transactions 918

Copyright © 1988-2007, Oracle. All rights reserved.

running transactions 924
understanding the RunAsynch method 954
AttachCube method 86
AttachDimension method
CubeDefn class 78
ExplicitDimensionSet class 76
AttachList property 672
AttachmentRoot property 665
attachments
IBInfo class 802, 803, 804, 805, 809
IsAttachment property 1491
number of 815
ResponseAsAttachment property 817
using events 1593
AttachmentURL property 650
AttachPart method 99
AttachSizes property 672
AttachTree methods 22
Attribute collection
methods 1089
See Also Attribute collection methods
properties 1092
understanding 1089
Attribute collection methods
DeleteItem 1089
First 1090
InsertItem 1090
ItemByName 1091
Next 1092
Attribute collection properties 1092
AttributeCount property 313
attributes
AddAttributeNS method 1766
Attribute collection 1089
See Also Attribute collection
AttributeCount property 313
AttributesCount property 1797
AttributeValue class 1087
Content Reference class Attributes property 1073
Folder class Attributes property 1061
GetAttributeName method 1778
GetAttributeValue method 1779
GetPeerDefaultAttributesByName method 843
Incoming Business Interlink class AddAttribute method 303
Incoming Business Interlink class GetAttributeName method 309, 310
Pagelet class Attributes property 1146
PageletCategory class Attributes property 1138
PeerDefaultAttributes class 873
TabDefinition class Attributes property 1115
understanding PortalRegistry 1000
using 1179
XmlNode class AddAttribute method 1765
Attributes property
Content Reference class 1073
Content Reference Links class 1103
Folder class 1061
Pagelet class 1146
PageletCategory class 1138
TabDefinition class 1115
AttributesCount property 1797
AttributeValue class
list of properties, returns 2041
properties 1087
See Also AttributeValue class properties
understanding 1087
AttributeValue class properties
Label 1087
Name 1088
Translatable 1088
Value 1089
AttributeValue collection 2041
attrName property 1857
attrValue property 1857
Audit method 1654
AuditByName method 1655
AuditDetails property 1676
authentication
AuthTokenDomain property 599
AuthType property 600
Servlet property 605
signing on to PeopleSoft 1512
Author property
Content Reference class 1073
Content Reference Links class 1103
Folder class 1061
Pagelet class 1146
PageletCategory class 1139
TabDefinition class 1115
AuthorAccess property
Content Reference class 1073
Content Reference Links class 1103

Copyright © 1988-2007, Oracle. All rights reserved.

Folder class 1061
Pagelet class 1146
PageletCategory class 1139
TabDefinition class 1115
understanding 1001
Authorized property
Content Reference class 1073
Content Reference Links class 1103
Folder class 1061
Pagelet class 1147
PageletCategory class 1139
AuthorizeEmailAttach method 676
AuthTokenDomain property 599
AuthType property 600
AvailableCategories property 1114
AvailableCategory
class 1128
See Also AvailableCategory class
collection 1128
See Also AvailableCategory collection
AvailableCategory class
list of properties, returns 2048
properties 1128
understanding 1128
AvailableCategory collection
list of methods, properties, returns 2048
methods 1129
See Also AvailableCategory collection methods
properties 1130
understanding 1128
AvailableCategory collection methods
First 1129
ItemByName 1129
Next 1130
AvailablePagelet
class 1130
See Also AvailablePagelet class
collection 1132
See Also AvailablePagelet collection
objects 1130
AvailablePagelet class
list of, properties, returns 2049
properties 1131
See Also AvailablePagelet class properties
understanding 1130
AvailablePagelet class properties
CategoryLabel 1131
CategoryName 1131
Column 1131
LayoutBehavior 1131
PageletLabel 1132
PageletName 1132
Row 1132
AvailablePagelet collection
list of methods, properties, returns 2049
methods 1132
See Also AvailablePagelet collection methods
properties 1134
understanding 1132
AvailablePagelet collection methods
First 1133
ItemByName 1133
Next 1133
AvailablePagelets property
AvailableCategory class 1128
TabDefinition class 1115
AvgExecTime property 1345
AvgFetchTime property 1345
AvgNumRows property 1345
axes
Chart class methods 362
See Also Chart class axes methods
Chart class properties 364
See Also Chart class axes properties
AxisEndDateTime property 394

B

BackgroundFile property 1883
BackupServer property 709
BackupSMTPPort property 694
BackupSMTPServer property 695
BackupSMTPUserName property 695
BackupSMTPUserPassword property 695
BackupUserName property 709
BackupUserPassword property 709
bar charts 333
base classes
adding members 179
designing 183
MCFFactory class 1728
See Also MCFFactory class
BCC property
MCFEmail class 653
MCFOutboundEmail class 695
BIDoc class 1921
BIDocs 256

Copyright © 1988-2007, Oracle. All rights reserved.

BindFlag property 1331
Bindings
collection 201
See Also Bindings collection
methods 202
property 201
Bindings class
list of properties, returns 2016
properties 202
Bindings collection
list of properties, returns 2016
methods 202
properties 201
Bindings collection methods 202
Bindings collection properties 201
Bindings methods 202
Bindings properties 202
Bindings property 201
BodyNode property 1557
BounceTo property
MCFEmail class 653
MCFOutboundEmail class 695
BPEL
classes 241
See Also BPEL classes
BPEL classes 1919
AsyncFFSend 244
See Also AsyncFFSend class
BPELUtil 245
See Also BPELUtil class
constructors 243
See Also BPEL class constructors
creating BPEL objects 242
declaring data types 242
IBUtil 248
See Also IBUtil class
importing 242
scope 241
understanding 241
BPEL classes constructors
AsyncFFSend 243
BPELUtil 243
IBUtil 244
BPEL objects, creating 242
BPELUtil class
LaunchASyncBPELProcess 247
LaunchSyncBPELProcess 248
methods 245
understanding 245
BPELUtil class methods
GetASyncBPELProcessInstanceUrl 245
GetBPELProcessBrowserUrl 246
GetOperationType 246
GetSyncBPELProcessInstanceUrl 246
BPELUtil constructor 243
Branch collection
list of properties, returns 2075
property 1618
understanding 1618
Branch collection property 1618
Branch method 1634
branches
Branch method 1634
Branches property 1676
BranchImageName property 1676
collection 1618
See Also Branch collection
Branches property 1676
BranchImageName property 1676
BranchLevel property 1677
BranchName property 1677
Broadcast class
method 1725
Broadcast constructor 1712
Broadcast method 1725
BrowserPlatform property 600
browsers
BrowserPlatform property 600
BrowserType property 600
BrowserURL property 1730
BrowserVersion property 600
changing portal templates 1041
using plug-ins for iScripts 590
BrowserType property 600
BrowserURL property 1730
BrowserVersion property 600
bubble chart 327
Buddy property 1721
buffers
component 731
Record class 1377
RowsetCache class 1453
supporting batch input/output 255
supporting dynamic output 255
using Business Interlink objects 254
BuildSearchIndex method 1467
BuildType property 1498
built-in functions, *See* PeopleCode functions
deprecated 2092

Copyright © 1988-2007, Oracle. All rights reserved.

BulkExecute method 268
BulkMode property 1577
BurstFieldName property 1829
busactivity property 903
buseventname property 903
Business Interlink
  class 253
    See Also Business Interlink class
  objects 253
    See Also Business Interlink objects
Business Interlink class
  configuring parameters 317, 318
  incoming 303
    See Also Incoming Business Interlink class
  methods 254
    See Also Business Interlink methods
  properties 316
  understanding 253
  using with Application Engine 260
Business Interlink class methods
  AddDoc 261
  AddInputRow 262
  AddNextDoc 264
  AddValue 266
  BulkExecute 268
  choosing 254
  Clear 272
  Execute 273
  executing Business Interlink objects 254
  FetchIntoRecord 275
  FetchIntoRowset 277
  FetchNextRow 280
  GetCount 281
  GetDoc 282
  GetFieldCount 285
  GetFieldType 286
  GetFieldValue 287
  GetInputDocs 288
  GetNextDoc 289
  GetOutputDocs 291
  GetPreviousDoc 292
  GetStatus 294
  GetValue 295
  InputRowset 297
  MoveFirst 299
  MoveNext 300
  MoveToDoc 301
  ResetCursor 302
  supporting batch input/output 255
  supporting dynamic output 255
  supporting rowsets 255
  using flat table methods 255
  using hierarchical data (BIDocs) 256
Business Interlink objects
  declaring as global 260
  declaring/scope 260
  executing 254
  input structures 256
  output structures 257
  supporting batch input/output 255
  supporting dynamic output 255
  supporting rowsets 255
  understanding 253
  understanding states 260
  using 254
  using flat table methods 255
  using hierarchical data 256
Business Interlink plug-in configuration 317
Business Interlinks class 1920
busprocname property 903
ByPassSignOn property 600

C

CalcAggregates property 84
CalculateCube method 24
callback
  OptBase callback methods 962
callback property 150
callback records
  designating in analytic type definitions 963
  using Optbase Callback methods 962
Cancel method 426
  IntBroker class 784
CanCreatePublic property 1347
CanCreateWorkFlow property 1347
CanModifyQuery property 1348
CanRunQuery property 1348
CanRunToCrystal property 1348
CanRunToExcel property 1348
Cascade property 1002, 1093
CascadedPermissions property
  Content Reference class 1074
  Content Reference Links class 1104
  Folder class 1062
  Pagelet class 1147
  PageletCategory class 1139

Copyright © 1988-2007, Oracle. All rights reserved.

understanding 1002
CascadedRolePermissions property
Content Reference class 1074
Content Reference Links class 1104
Folder class 1062
cascading 1002
Category property 1677
CategoryId property 1830
CategoryLabel property 1131
CategoryName property
AvailableCategory class 1128
AvailablePagelet class 1131
SelectedPagelet class 1161
causes 79
CC property
MCFEmail class 654
MCFOutboundEmail class 696
CGI variables
AUTH_TYPE 600
REMOTE_USER 604
SCRIPT_NAME 605
ChangeOnInit property 1445
Channel property 890
ChannelName property 769
characters
Charset property 611
subscribing to character fields 735
using character values 521
Charset property 611, 650
MCFOutboundEmail class 696
Chart class
appearance properties 354
See Also Chart class appearance properties
axes methods 362
See Also Chart class axes methods
axes properties 364
See Also Chart class axes properties
color methods 361
See Also Chart class color methods
creating charts 401
creating charts via iScripts 407
creating Gantt charts 403
data methods 345
See Also Chart class data methods
data properties 353
See Also Chart class data properties
functions, methods, properties, returns 1922
Gantt chart DateTimeAxis Methods 397
See Also Chart class Gantt properties
Gantt chart methods 380
See Also Chart class Gantt methods
Gantt chart properies 394
See Also Chart class Gantt properties
handling errors 343
legend methods 369
legend properties 370
See Also Chart class legend properties
overlay methods 372
See Also Chart class overlay methods
overlay properties 374
See Also Chart class overlay properties
Scatter chart methods 374
See Also Chart class Scatter methods
Scatter chart properties 376
See Also Chart class Scatter chart property
selecting color values 360
title properties 367
See Also Chart class title properties
TrueXY line chart properties 377
See Also Chart class TrueXY line chart properties
understanding 319
Chart class appearance properties
GridLines 355
Height 356
IsDrillable 356
IsPlainImage 357
LineType 357
RotationAngle 358
Style 358
StyleSheet 358
Type 359
Width 359
Chart class axes methods
SetXAxisLabels 362
SetYAxisLabels 363
Chart class axes properties
XAxisCross 364
XAxisLabelOrient 365
XAxisStyle 365
XAxisTicks 366
YAxisLabelOrient 366
YAxisMax 366
YAxisMin 366
YAxisStyle 367

Copyright © 1988-2007, Oracle. All rights reserved.

YAxisTicks 367
Chart class color methods
SetColorArray 361
SetDataColor 362
Chart class data methods
Refresh method 345
SetData method 346
SetDataHints method 347
SetDataSeries method 350
SetDataURLs method 351
SetDataXAxis method 351
SetDataYAxis method 352
Chart class data properties
DataStartRow 353
DataWidth 353
ImageMap 354
Chart class Gantt DateTimeAxis methods
SetDayFormat 399
SetHourFormat 398
SetMinuteFormat 398
SetMonthFormat 400
SetSecondFormat 397
SetYearFormat 400
Chart class Gantt methods
SetActualEndDate 380
SetActualStartDate 380
SetActualTaskBarColor 381
SetChartArea 381
SetPlannedEndDate 382
SetPlannedStartDate 382
SetPlannedTaskBarColor 383
SetTableXScrollbar 383
SetTaskAppData 384
SetTaskAppDataTitles 384
SetTaskBarURL 385
SetTaskData 385
SetTaskDependencyChildID 386
SetTaskDependencyData 386
SetTaskDependencyParentID 388
SetTaskDependencyType 387
SetTaskDependencyURL 388
SetTaskExpanded 389
SetTaskHints 389
SetTaskID 390
SetTaskLabel 390
SetTaskLevel 390
SetTaskMilestone 391
SetTaskName 392
SetTaskProgress 392
SetTaskProgressBarColor 393
SetWBSNumbering 393
Chart class Gantt properties
AxisEndDateTime 394
DataEndDateTime 395
DataStartDateTime 395
ShowTaskLabels 395
TaskDependencyLineType 395
TaskMilestoneGlyph 396
TaskTitle 396
XAxisPosition 396
YAxisPosition 397
Chart class legend methods 370
Chart class legend properties
HasLegend 371
LegendMaxEntries 371
LegendPosition 371
LegendStyle 371
Chart class overlay methods
SetOLData 372
SetOLDataSeries 372
SetOLDataXAxis 373
SetOLDataYAxis 373
Chart class overlay properties
OLLineType 374
OLType 374
Chart class Scatter chart property
ShowCrossHair 376
Chart class Scatter methods
SetDataAnnotations method 375
SetDataGlyphScale 375
SetOLDataAnnotations 375
SetOLDataGlyphScale 376
Chart class title properties
MainTitle 367
MainTitleOrient 368
MainTitleStyle 368
XAxisTitle 368
XAxisTitleOrient 368
XAxisTitleStyle 369
YAxisTitle 369
YAxisTitleOrient 369
YAxisTitleStyle 369
Chart class TrueXY line chart properties
XAxisCrossPoint 377
XAxisMax 377
XAxisMin 377
XAxisPrecision 377
XAxisScaleResolution 378
YAxisCrossPoint 378
YAxisMax 379

Copyright © 1988-2007, Oracle. All rights reserved.

YAxisMin 379
YAxisPrecision 379
YAxisScaleResolution 379
chart objects 343
charts
2D bar 323
2D histogram 323
2D horizontal bar 324
2D horizontal percent bar 325
2D horizontal stacked bar 324
2D line 325
2D percent bar 326
2D pie 327
2D scatter 327
2D stacked bar 328
3D bar 328
3D percent bar 329
3D pie 329
3D stacked bar 330
bar chart considerations 333
bubble 327
Bubble chart 335
Chart class 319
See Also Chart class
component processor 333
creating 401
creating Gantt charts 403
creating via iScripts 407
examples 401
Gantt 330
label concerns 333
Line chart 335
objects 343
terminology
Gantt chart 338
general 322
translating values 333
types 322
using iScripts 334
using style sheets 331
chat 723
CheckAll method 725
CheckAsyncStatus method 14
CheckForChanges method 1596
CheckOptEngineStatus method 942
CheckStatus method 15
ChildCount property 1413
ChildLeafCount property 1647
ChildNodeCount property
Incoming Business Interlink class 313
Node class 1647
XmlNode class 1797
CI collection
list of methods, properties, returns 1963
methods 860
properties 860
understanding 835, 859
circular dependencies 80
CircularFormulaWarn property 72
Class language construct 186
classes
application 153
See Also application classes
mail 629
See Also mail classes
OptBase 962
See Also OptBase class methods
OptEngine 942
See Also OptEngine class
OptInterface 978
See Also OptInterface class methods
PeopleCode 1
See Also PeopleCode classes
Classes property 204
ClassInfo
collection 205
See Also ClassInfo collection
properties 207
ClassInfo class 2017
ClassInfo collection
list of properties, returns 2017
methods 206
properties 205
ClassInfo properties 207
Clear method
Business Interlink 272
Response class 606
ClearAttachments method
IBInfo class 803
ClearConnectorProperties method 821
ClearDeletesChanges method 1423
ClearDropDownList method 494
ClearQueryStringArgs method 821
ClientPlatform property 1599
ClientTimeZone property 1528
Clone method
Array class 223
Message class 739
Close method
AESection class 6

Copyright © 1988-2007, Oracle. All rights reserved.

File class 530
PortalRegistry class 1023
Query class 1264
SQL class 1572
Task class 1733
Tree class 1656
Tree Structure class 1690
COBOLStmtTimings property 1533
ColImageName property 1647
CollapseNode method 30
collections, *See* PeopleCode collections
colors
Chart class methods 361
See Also Chart class color methods
selecting for charts 360
Column property
AssignedPagelet class 1123
AvailablePagelet class 1131
SelectedPagelet class 1161
ColumnLayout property
TabDefinition class 1115
UserTab class 1156
ColumnNumber property 1306
comments
Incoming Business Interlink class AddComment method 304
Task class Comments property 1737
understanding application class 182
WorklistEntry class commentshort property 903
XmlNode class AddComment method 1768
XmlNode class InsertComment method 1786
Comments property
AnalyticTypeDefn class 147
CubeCollectionDefn class 96
CubeDefn class 84
DimensionDefn class 75
OrganizerDefn class 101
Task class 1737
UserFunctionDefn class 98
comments, submitting cxviii
commentshort property 903
CommitWork function 915, 916
common elements cxviii
CompareFields method 1379
compareOp property 1858
comparison class
creating 105
GenerateRule method 106
methods 106
Operand1 property 107
Operand2 property 107
properties 107
Type property 107
CompIntfPropInfo
class 2020
collection 2020
CompIntfPropInfoCollection collection methods 453
See Also CompIntfPropInfoCollection collection methods
properties 454
CompIntfPropInfoCollection collection methods
First 453
Item 453
Next 454
CompIntfPropInfoCollection objects
Format property 454
IsCollection property 455
IsReadOnly property 455
Key property 455
LabelLong property 456
LabelShort property 456
Length property 456
Name property 456
Prompt property 457
properties 454
PropertyInfoCollection property 457
Required property 457
Type property 457
understanding 451
Xlat property 458
YesNo property 458
CompIntfPropInfoCollection property 451
component buffers 731
Component Interface
accessing structure 451
classes 411
See Also Component Interface classes
collection 424
See Also Component Interface collection
creating new instances 458
examples 458
filtering 1594
getting instances 461

Copyright © 1988-2007, Oracle. All rights reserved.

GUI/online processing 422
implementing definitions 418
inserting effective-dated data 465
inserting effective-dated data via Visual Basic 468
inserting/deleting row data 469
life cycle 412
mobile PeopleCode 831
objects 411
See Also Component Interface objects
processing Search dialog boxes 422
retrieving instance lists 463
synchronizing 828
traversing via data collections 419
understanding 411
using %Menu conditions 423
using component variables 418
using mobile objects 831
using related languages 418
using the SetCursorPos method 499
Component Interface class
list of methods, properties, returns 2019
methods 426
See Also Component Interface class methods
properties 440
See Also Component Interface class properties
Component Interface class methods
Cancel 426
CopyRowset 426
CopyRowsetDelta 428
CopySetupRowset 431
CopySetupRowsetDelta 433
Create 435
Find 436
Get 437
GetPropertyByName 437
Save 439
SetPropertyByName 439
Component Interface class properties
ComponentName 440
CreateKeyInfoCollection 441
EditHistoryItems 441
FindKeyInfoCollection 441
GetDummyRows 442
GetHistoryItems 442
GetKeyInfoCollection 442
InteractiveMode 443
PropertyInfoCollection 443
StopOnFirstError 443
Component Interface classes
Component Interface 426
See Also Component Interface class
Data Item 450
See Also Data Item
list of methods, properties, returns 2018
methods 411
See Also Component Interface class methods
properties 411
See Also Component Interface class properties
reusing existing code 422
Session 423
See Also Session class
setting keys 414
understanding 411
Component Interface classes methods
setting timeouts 419
understanding 411
using effective-dated data 421
using GetEffectiveItem 421
using InsertItem 422
using standard/user-defined 415
Component Interface classes properties
accessing standard properties 416
accessing user–defined properties 417
executing a Component Interface 416
retrieving Component Interface information 416
understanding 411
Component Interface collection
list of methods, properties, returns 2018
methods 424
See Also Component Interface collection methods
properties 425
See Also Component Interface collection properties
Component Interface collection properties
CompIntfPropInfoCollection 451
Count property 425
Component Interface collections
CompIntfPropInfoCollection 451
See Also CompIntfPropInfoCollection collection
Data 443
See Also Data collection
Component Interface objects

Copyright © 1988-2007, Oracle. All rights reserved.

declaring/scope 417
understanding 411
component variables 418
ComponentId property 894
ComponentName property 440
components
buffers 731
Component Interface 411
See Also Component Interface
component processor 333
ConfFile property
EFTProcessor class 1870
FOProcessor class 1875
FormProcessor class 1879
RTFProcessor class 1894
ConflictAlgorithm property 1600
ConflictStatus property 1600
ConfProp property
EFTProcessor class 1871
FOProcessor class 1876
FormProcessor class 1880
PDFMerger class 1889
RTFProcessor class 1895
ConnDisRollbackCommit property 1533
Connect method 1516
ConnectorClassName property 825
ConnectorInfo collection
list of methods, properties, returns 1959
methods 819
See Also ConnectorInfo collection methods
objects 819
properties 825
understanding 819
ConnectorInfo collection methods
AddConnectorProperties method 819
AddQueryStringArg method 820
ClearConnectorProperties method 821
ClearQueryStringArgs method 821
DeleteConnectorProperties method 822
DeleteQueryStringArg method 822
GetConnectorPropertiesName method 823
GetConnectorPropertiesType method 823
GetConnectorPropertiesValue method 823
GetNumberOfConnectorProperties method 824
GetNumberOfQueryStringArgs method 824
GetQueryStringArgName method 824
GetQueryStringArgValue method 825
ConnectorName property 826
ConnectorOverride property 811
ConnectorRequest method 785
ConnectorRequestUrl method 786
considerations 333
constant class
creating 108
methods 109
properties 109
Type property 109
using 104
Value property 109
constants
declaring 177
Expr2Constant1 property 1320
Expr2Constant2 property 1320
using 174
constructors
analytic type classes 140
BPEL classes 243
See Also BPEL class constructors
mail classes 643
notifications classes 881
See Also Notification class constructors
PrcsApi class 1230
report manager search classes 1848
ReportManager 1849
SearchAttribute 1850
understanding 166
universal queue classes 1710
See Also universal queue classes constructors
using %This 173
XML Publisher engine classes 1864
contact information cxviii
container messages
alias names 769
copying parts 740
determining parts 776
determining structured parts 776
generating XML part string 751
getting part alias names 752
getting part name 753
getting part rowset 753
getting part version 754
getting XML 754

Copyright © 1988-2007, Oracle. All rights reserved.

part count 779
populating parts 758
Content Reference class
adding content references 1175
methods 1070
See Also Content Reference class methods
properties 1072
See Also Content Reference class properties
understanding 1070
Content Reference class methods
CreateLink 1070
Save 1071
Content Reference class properties
AbsoluteContentURL 1072
AbsolutePortalURL 1072
AssignedPagelets 1072
Attributes 1073
Author 1073
AuthorAccess 1073
Authorized 1073
CascadedPermissions 1074
CascadedRolePermissions 1074
ContentProvider 1074
CreationDate 1075
Data 1075
Description 1075
HtmlText 1075
IsMobile 1075
IsVisible 1076
Label 1076
Links 1076
Name 1076
OwnerId 1077
ParentName 1077
Path 1077
Permissions 1077
Product 1078
PublicAccess 1078
QualifiedURL 1078
RelativeURL 1078
RolePermissions 1079
SequenceNumber 1079
StorageType 1079
Template 1080
TemplateObject 1081
TemplateType 1081
URL 1081
URLType 1082
UsageType 1083
ValidFrom 1083
ValidTo 1083
Content Reference collection
methods 1084
See Also Content Reference collection methods
properties 1087
understanding 1084
Content Reference collection methods
DeleteItem 1084
First 1085
InsertItem 1085
ItemByName 1086
Next 1086
content reference links
class 1101
See Also Content Reference Links class
objects 1101
using FindCRefLinkByName method 1031
Content Reference Links class
methods 1101
properties 1102
See Also Content Reference Links class properties
understanding 1101
Content Reference Links class properties
AbsoluteContentURL 1102
AbsolutePortalURL 1103
Attributes 1103
Author 1103
AuthorAccess 1103
Authorized 1103
CascadedPermissions 1104
CascadedRolePermissions 1104
ContentProvider 1105
CreationDate 1105
Data 1105
Description 1105
IsMobile 1105
IsVisible 1106
Label 1106
Name 1106
OwnerId 1106
ParentName 1107
Path 1107
Permissions 1107
Product 1107

Copyright © 1988-2007, Oracle. All rights reserved.

PublicAccess 1108
RelativeURL 1108
RolePermissions 1108
SequenceNumber 1108
Template 1109
TemplateObject 1109
TemplateType 1109
URL 1110
URLType 1110
UsageType 1110
ValidFrom 1111
ValidTo 1111
content reference objects
using FindCRefByName method 1028
using FindCRefByURL method 1029, 1030
content reference properties
TemplateType 1015
understanding 1013
URL 1016
URLType 1015
UsageType 1013
content references
adding 1175
class 1070
See Also Content Reference class
collection 1084
See Also Content Reference collection
deleting content 1017
links 1101
See Also content reference links
naming 1017
Permissions property 1002
properties 1013
See Also content reference properties
types 1013
understanding 998
understanding nodes 999
Content RowsetCache property 1458
contentId property 1850
ContentLanguage property 696
ContentProvider class
adding a ContentProvider 1171
ContentProvider property
Content Reference class 1074
Content Reference Links class 1105
Pagelet class 1147
ContentReference class 2038
ContentReference collection 2041
ContentReference Link class 2043
ContentRefs property 1062
ContentType property
MCFBodyPart class 650
MCFOutboundEmail class 696
Notification class 886
ContentTypes property 666
ContentURI property
Node class 1047
Request class 600
Context property 487
ConversationID property 811
conversion, PeopleCode/Java 624
Cookie class
list of properties, returns 1942
properties 612
See Also Cookie class properties
understanding 595, 612
Cookie class properties
Domain property 612
MaxAge property 612
Name property 612
Path property 613
Secure property 613
Value property 613
cookie objects
declaring 594
understanding 595
cookies
ConnectorInfo collection Cookies property 826
Cookie class 595
See Also Cookie class
cookie objects 594
See Also cookie objects
Message class Cookies property 770
Cookies property
ConnectorInfo collection 826
IBInfo class 811
Message class 770
Copy method
AnalyticInstance class 15
Tree class 1656
Tree Structure class 1690
CopyChangedFieldsTo method 1380
CopyCube method 52
CopyCubeCollection method 52
CopyDimension method 53
CopyExplicitDimensionSet method 53
CopyFieldsTo method 1381
copying

Copyright © 1988-2007, Oracle. All rights reserved.

analytic instances 15
analytic type definitions 142
cube collections 52
CopyNode method 1774
CopyObject method 1023
CopyPartRowset Method 740
CopyRowset method
Component Interface class 426
Message class 741
XmlDoc class 1751
CopyRowsetDelta method
Component Interface class 428
Message class 742
CopyRowsetDeltaOriginal method 744
CopySetupRowset method 431
CopySetupRowsetDelta method 433
CopyTo method
AnalyticModelDefn class 54
AnalyticTypeDefn class 142
Row class 1409
Rowset class 1424
CopyToPSFTMessage method 1752
CopyToRowset method 1754
CopyURLLink property 992
CopyUserFunction method 54
Count property
AssignedPagelet collection 1127
Attribute collection 1092
AvailableCategory collection 1130
AvailablePagelet collection 1134
Bindings collection 201
Branch collection 1619
CI collection 860
ClassInfo collection 205
CompIntfPropInfoCollection collection 454
Component Interface collection 426
Content Reference collection 1087
Data collection 450, 865
DynamicCategory collection 1137
Favorite collection 1169
Folder collection 1070
Level collection 1631
Link collection 1112
MethodInfo collection 208
Namespaces collection 203
Node collection 1050
NodeTemplate collection 1059
Pagelet collection 1154
PageletCategory collection 1145
PermissionValue collection 1097
Portal collection 1055
PortalRegistry collection 1045
PropertyInfo collection 211
PropertyInfoCollection class 855
PSMessages collection 1246, 1524
Query collection 1263
Query Metadata collection 1343
QueryCriteria collection 1314
QueryDBRecord collection 1352
QueryDBRecordField collection 1357
QueryExpression collection 1331
QueryField collection 1304
QueryList class 1336
QueryPrompt collection 1363
QueryRecord collection 1298
QueryRecordHierarchy collection 1340
QuerySelect collection 1296
RemoteNode collection 1052
RolePermissionValue collection 1100
Search Language collection 1494
Search Schedule collection 1497
Search Start Options collection 1506
SearchField collection 1480
SearchIndexes collection 1482
SearchRecordField collection 1491
SearchResult collection 1477
SelectedPagelet collection 1164
TabDefinition collection 1122
UserTab collection 1160
Create method
AnalyticModelDefn class 55
AnalyticTypeDefn class 142
Component Interface class 435
Level class 1632
Mobile class 840
PortalRegistry class 1024
Query class 1265
Tree class 1658
Tree Structure class 1691
WorklistEntry class 897
CreateContentRefLink method 1025
CreateCookie method 606
CreatedDateTime property 1851
CreateDocumentElement method 1756
CreateDocumentType method 1757
CreateDtTm property 1277
CreateElement method 307
CreateKeyInfoCollection property 441

Copyright © 1988-2007, Oracle. All rights reserved.

CreateLink method 1070
CreateNextSegment method 747
CreateOprId property 1277
CreateOptEngine function
  loading analytic instances 917
  programming for database updates 920
  running optimization transactions 918
  understanding 933
  using optimization PeopleCode in Application Engine programs 916
  using optimization PeopleCode on application servers 915
CreateOptInterface function 935, 962
CreateQuarantineFolder method 656
CreateRemote method 1026
CreateRowset method 530
CreateUserId property 1277
CreateXmlDoc function 1746
CreationDate property
  Content Reference class 1075
  Content Reference Links class 1105
  Folder class 1063
  Pagelet class 1147
  PageletCategory class 1140
  TabDefinition class 1116
CRefName property 1165
criteria
  AddHavingCriteria method 1285
  adding 1369
  Criteria property 1290
  "Having" criteria for queries 1248
  HavingCriteria property 1291
  queries 1241
Criteria property 1290
cross-references cxvii
Crypt
  class 475
    See Also Crypt class
  objects 475
Crypt class
  functions, methods, properties, returns 1931
  methods 476
    See Also Crypt class methods
  properties 479
  understanding 475
Crypt class methods
  FirstStep 476
  GoToStep 476
  LoadLibrary 477
  NextStep 477
  Open 478
  SetParameter 478
  UpdateData 479
Crypt class properties 479
Crypt objects 475
cryptography 475
cube
  accessing 60
    causes 79
    circular dependencies 80
    dimension aggregates 81
    dimension names 82
    effects 81
    names 62
    rules 82
  adding to analytic model 48
  attaching dimensions 78
  attaching to cube collections 86
  calculating 24
  calculating aggregates 84
  class methods 78
  class properties 84
  commenting 84
  copying 52
  count for collections 96
  counting dimensions 85
  deleting 56
  detaching dimensions 79
  detaching from cube collections 87
  naming 85
  setting
    rules 83
  specifying format type 85
  using dimensions 84
  value dimension names 85
  virtual 85
cube class
  AddIndex method 110
  creating 110
  GenerateRule method 111
  GetIndexes method 111
  methods 110
  Name property 112
  property 112
cube collection
  accessing 61
    current layout 35
    dimension names 88
    dimension sort settings 35

Copyright © 1988-2007, Oracle. All rights reserved.

names 88
adding to analytic model 49
aggregate record name 96
analytic grids 133
attaching cubes 86
class methods 86
class properties 96
collapsing nodes 30
commenting 96
copying 52
counting cubes 96
counting dimensions 96
deleting 57
detaching cubes 87
DrillIntoNode 31
DrillOutOfNode 31
expanding nodes 32
filtering 90, 93
getting a slice 36
getting field mapping 89
getting names 61
getting row count 36
hierarchy display 40
messages property 42
naming 96
persist aggregate 90
populating from rowset 37
populating rowsets 33
record name 97
removing filters 41
removing sorting 41
renaming 68
returning user function 34
setting aggregate mapping 91
setting dimension order 37
setting field mapping 93
setting layout 135
sorting dimensions 89, 92
specifying slice records 39
specifying sort order 38
using cubes 95
using dimensions 95
cube definition
aggregate mapping 87
CubeCollection class 30
CubeCollection class property 42
CubeCollectionDefn class
methods 86
properties 96
CubeCollectionDefn class methods 86
CubeCollectionDefn class properties 96
CubeCount property 96
CubeDefn class
methods 78
properties 84
CurrencyFormat property 1529
CurrencySymbol property 1529
CurrentItem method 444
CurrentItemNumber property 450
CurrentRecord property 551
CurrentSegment property 771
cursors
ResetCursor method 302
ReuseCursor property 1580
reusing 1567
SetCursorPos method 498
Customer Connection website cxiv
CustomizePageLink property 993
Cut method
Leaf class 1619
Node class 1635

D

data
chart data methods 345
See Also Chart class data methods
chart data properties 353
See Also Chart class data properties
collection 419
See Also Data collection
Content Reference class Data property 1075
Content Reference Links class Data property 1105
Data Item class 450
types 222
See Also data types
Data collection
list of methods, properties, returns 1963, 2021
methods 443, 861
See Also Data collection methods
objects 861
properties 449, 865
traversing a Component Interface 419
understanding 835
Data collection methods
CurrentItem 444
DeleteItem 444, 863
GetEffectiveItem 446

Copyright © 1988-2007, Oracle. All rights reserved.

GetEffectiveItemNum 446
GetListViewAttrs 861
InsertItem 447, 864
Item 448, 862
ItemByKeys 448
Data collection objects 861
data hints 347
Data Item class 450
Data property
Content Reference class 1075
Content Reference Links class 1105
data types
declaring array objects 222
declaring BPEL 242
declaring Business Interlink objects 260
declaring chart objects 343
declaring Component Interface objects 417
declaring Cookie objects 594
declaring Exception objects 484
declaring field objects 492
declaring file objects 524
declaring Grid/GridColumn objects 577
declaring iScript objects 594
declaring Java objects 628
declaring MCFIMInfo objects 724
declaring message objects 731
declaring mobile objects 838
declaring notifications 880
declaring page objects 992
declaring PortalRegistry objects 1018
declaring PostReport objects 1186
declaring ProcessRequest objects 1194
declaring query objects 1248
declaring record objects 1379
declaring Request objects 594
declaring Response objects 594
declaring row objects 1408
declaring rowset objects 1422
declaring RowsetCache objects 1455
declaring session objects 1514
declaring SOAPDoc objects 1546
declaring SQL objects 1571
declaring SyncServer objects 1594
declaring Tree objects 1614
declaring XML Publisher objects 1808
declaring XmlDoc objects 1750
declaring XmlNode objects 1750
mail classes 630
mapping PeopleCode and Java 624
passing parameters 163
universal queue classes 1709
DataAreaCollapsed property
Field class 500
Rowset class 1445
database cache, *See* RowsetCache class
DatabaseName property
report class properties 1851
databases
forcing a commit on updates 915, 916, 920
programming for updates 920
queries 1239
DataEndDateTime property 395
DataSourceDefn class
constructor 1810
methods 1813
properties 1815
DataSourceDefn class properties
Name 1816
Type 1817
DataSourceDefn methods
Get method 1813
GetData 1814
GetSampleData 1814
GetSchema 1815
GetTemplateList 1835
DataSourceDefn properties
Active 1815
Description 1815
IsPublic 1816
LastUpdatedBy 1816
LastUpdateDTTM 1816
ObjectOwnerID 1816
RegisteredBy 1816
RegisteredDTTM 1817
DataStartDateTime property 395
DataStartRow property 353
DataWidth property 353
DateFormat property 1529
dates
effective dates 421
See Also effective dates
using with ReadRowset method 539
DateSeparator property 1530
DB2 UDB for OS/390 and z/OS
considerations for extra spaces 1564
sorting records 1428

Copyright © 1988-2007, Oracle. All rights reserved.

DBPatternMatch method 1384
DBRecordName property 1446
DBSpecificCalls property 1533
DeactivateModel method 980
debugging
mobile PeopleCode 835, 836
setting trace options 1514
using the Java Debugging Environment (JDB) 627
Decimal property
ColumnNumber class 1307
QueryDBRecordField class 1357
QueryExpression class 1332
DecimalPosition Field class 501
DecimalSymbol property 1530
DecodeText method 680
DecodeWord method 681
DefaultCharSet property 697
DefaultColumn property 1147
DefaultMessageVersion property 771
DefaultPortalName property 1047
DefaultStyleSheetName property 611
DefaultTemplate property
NodeTemplate class 1056
PortalRegistry class 1041
DefaultText property 487
DefaultValue property 856
Delete method
Agent class 1717
analytic type definition 143
AnalyticInstance class 16
AnalyticModelDefn class 55
Content Reference Links class 1101
File class 531
Leaf class 1620
Node class 1635
PortalRegistry class 1027
Query class 1266
Record class 1385
RowsetCache class 1455
Tree class 1659
Tree Structure class 1691
DeleteAll method 1522
DeleteAttachment method
IBInfo class 804
DeleteByName method 1636
DeleteByRange method 1620
DeleteConnectorProperties method 822
DeleteCriteria method 1288
DeleteCube method 56
DeleteCubeCollection method 57
DeleteDimension method 57
DeleteEmail method 677
DeleteEnabled property
Row class 1414
Rowset class 1446
DeleteExplicitDimensionSet method 58
DeleteExpression method 1288
DeleteField method 1288
DeleteHavingCriteria method 1289
DeleteHomepage method 1028
DeleteItem method
AssignedPagelet collection 1124
Attribute collection 1089
Content Reference collection 1084
Data collection 444, 863
DynamicCategory collection 1135
Favorite collection 1166
Folder collection 1067
NodeTemplate collection 1057
Pagelet collection 1151
PageletCategory collection 1142
PermissionValue collection 1094
PSMessages collection 1523
RolePermissionValue collection 1098
Search Language collection 1492
Search Schedule collection 1496
Search Start Options collection 1504
SearchRecordField collection 1488
SelectedPagelet collection 1162
TabDefinition collection 1120
UserTab collection 1158
DeleteModel method 143
DeleteOptProbInst function 920, 936
DeleteOrganizer method 58
DeleteOrphanedSegments method 787
DeletePrompt method 1266
DeleteQuerySelect method 1293
DeleteQueryStringArg method 822
DeleteRecord method 144, 1289
DeleteRow method 1426
DeleteSegment method 748
DeleteUserFunction method 59
delivery status notifications, *See* email
Description property
AnalyticModelDefn class 72
AnalyticTypeDefn class 148
AnalyticTypeRecordDefn class 150
ColumnNumber class 1306
Content Reference class 1075

Copyright © 1988-2007, Oracle. All rights reserved.

Content Reference Links class 1105
DataSourceDefn class 1815
Folder class 1063
Level class 1633
MCFBodyPart class 650
MCFOutboundEmail class 697
Node class 1047, 1647
NotificationAddress class 890
Pagelet class 1148
PageletCategory class 1140
PortalRegistry class 1041
Query class 1277
QueryDBRecord class 1353
QueryDBRecordField class 1358
QueryRecord class 1300
QueryRecordHierarchy class 1340
ReportDefn class 1830, 1851
RowsetCache class 1458
TabDefinition class 1116
TemplateDefn class 1836
TranslationFile class 1842
Tree class 1677
Tree Structure class 1693
DestinationNode property 812
DestinationPrinter property 1830
DetachCube method 87
DetachDimension method
cube class 79
ExplicitDimensionSet class 76
DetachPart method 99
DetachTree methods 24
DetailComponent property 1694
DetailedStatus property 961
DetailField property 1694
DetailMenu property 1694
DetailMenuBar property 1695
DetailMenuItem property 1695
DetailMsgs property 960
DetailMultiNavigate property 1695
DetailPage property 1695
DetailRecord property 1696
DetailViewLabel property 867
DigitGroupingSymbol property 1530
dimension
accessing
aggregate sequence 75
explicit set names 64
names 63, 82
adding to analytic model 49
analytic model 62
attaching to cubes 78
class properties 75
commenting 75
copying 53
deleting 57
detaching from cubes 79
filter, setting 38
names for cube collection 88
naming 75
order, setting 37
renaming 68
setting aggregates 83
sort setting 35
sorting
CubeCollection class 38
CubeCollectionDefn class 89, 92
using 84, 95
using total values 75
Dimension property
Array class 238
MemberReference class 119
DimensionCount property
CubeCollectionDefn class 96
CubeDefn class 85
DimensionDefn class
properties 75
dimensions
attaching 76
detaching 76
Disabled property 1277
Disconnect method 1518
Discovery (API) 194
DisplayFormat property 501
DisplayLevelNumber property
Leaf class 1626
Node class 1647
DisplayOnly property
Field class 502
Page class 993
PeerDefaultAttributes class 874
PropertyAttrs class 870
DisplayOutput method 1817
DisplayZero property 503
DisplayZeroChanged property 503
Disposition property
MCFBodyPart class 651
MCFOutboundEmail class 697
Distinct property 1290
DistributeDataByRules function 1591
distribution lists, determining 1185

Copyright © 1988-2007, Oracle. All rights reserved.

DistributionChannel property 1836
do_replicate_flag property 904
documentation, *See* comments
  printed cxiv
  related cxiv
  updates cxiv
Documentation property
  ClassInfo class 207
  MethodInfo class 210
  PropertyInfo class 212
DocumentElement property 1764
Domain property 612
DomainLimit property 1500
DoNotPubToNodeName property 772
DoSave function 877
DoSaveNow function 915, 916
downcasting 179
DrillIntoNode method 31
DrillOutOfNode method 31
dttmCreated property 886
DttmReceived property 673
DttmSaved property 673
DttmSent property 673
DumpMsgToLog method 981
DumpToFile method 668
DuplicateLeaves property 1678
dynamic output
  Business Interlink objects 255
  GetFieldCount method 285
  GetFieldType method 286
  GetFieldValue method 287
  MoveFirst method 299
  MoveNext method 300
Dynamic property 1626
DynamicCategories property 1116
DynamicCategory collection
  list of methods, properties, returns 2049
  methods 1134
    See Also DynamicCategory collection methods
  properties 1137
  understanding 1134
DynamicCategory collection methods
  DeleteItem 1135
  First 1135
  InsertItem 1136
  ItemByName 1136
  Next 1136

E

EditError property 504
EditHistoryItems property 441
EditPageContentProvider property 1148
EditPageQueryString property 1148
EditType property 1364
EffDt property
  Rowset class 1447
  Tree class 1678
effective dates
  editing records 749
  inserting data in component interfaces 465
  inserting data in component interfaces via Visual Basic 468
  inserting rows 1437
  using the Record class SetEditTable method 1387
  using with a Component Interface 421
EffectiveDate property
  TemplateFile class 1840
effects 81
EffSeq property 1448
EFTProcessor class
  constructor 1864
  methods 1868
EFTProcessor methods
  GenerateOutput 1868
EFTProcessor properties
  ConfFile 1870
  ConfProp 1871
EFTProessor method
  GenerateXSL 1869
email
  delivery status notifications (DSN) 632
  double-downloading 636
  errors/status 635
  instant messaging 723
    See Also instant messaging
  mail classes 629
    See Also mail classes
  MIME 635
  outbound 631
  priority 633
  retrieving 635
  return receipts 632
  rowset structure 635
  setting attributes 1201
  setting languages 640

Copyright © 1988-2007, Oracle. All rights reserved.

See Also languages
EmailAttachLog property 1216
EmailId property 890
EmailReplyTo property 887
EmailSubject property 1216
EmailText property 1216
EmailWebReport property 1217
Enabled property
Field class 504
GridColumn class 581
EnableMap method 1885
EncodeText method 682
EncodeWord method 683
encryption profiles 475
engines
application 916
See Also Application Engine
Enqueue method 1734
EnqueuedTaskList property 1731
EnqueueTime property 1737
Enterprise Portal cxxi
entities
AddEntityReference method 1771
adding, inserting, getting, finding 1749
InsertEntityReference method 1790
understanding 1745
EnvelopeNode property 1557
environment variables
PS_CLASSPATH 616
using relative paths 534
ErrorCount property
MCFGetMail class 666
ParseResult Collection property 1474
ErrorDescription property 697
ErrorDetails property 698
ErrorPending property 1519
errors
displaying error messages 1513
email 642
Exception class 481
See Also Exception class
forcing commits on pending database updates 915, 916
handling chart errors 343
handling for all APIs 1512
handling for iScripts 593
handling for Java functions 627
handling for messages 738
handling for mobile PeopleCode 837
handling for PortalRegistry classes 1004
handling for Query classes 1246
handling for RowsetCache 1454
handling for Search classes 1464
handling for subscription PeopleCode 1512
handling for Tree classes 1612
handling for XML Publisher 1808
handling for XmlNode class 1749
loading analytic instances 917
logging API errors 1513
logging for APIs called from Application Engine 1513
logging system errors 1513
processing file layout errors 528
retrieving email 635
specifying severity for request messages 925
using the DumpMsgToLog method 981
using the GetTraceLevel method 953
using the SetTraceLevel method 957
EscalatedTaskList property 1731
EscalationTime property 1737
events, PeopleCode, *See* PeopleCode events
events, synchronization server, *See* synchronization server events
Exception class
functions, methods, properties, returns 1929, 1931
methods 485
See Also Exception class methods
properties 487
See Also Exception class properties
try-catch sections 483
understanding 180, 481
Exception class methods
GetSubstitution method 485
Output method 485
SetSubstitution method 486
ToString method 486
Exception class properties
Context property 487
DefaultText property 487
MessageNumber 488
MessageSetNumber 488
MessageSeverity property 488
StackTrace property 488
SubstitutionCount property 489
exceptions

Copyright © 1988-2007, Oracle. All rights reserved.

Exception class 180
See Also Exception class
handling for Java functions 627
MCFGetMail class methods 642
message class 772
objects 484
understanding 481
using 482
using transfer functions 483
using try-catch statements 483
ExcludeProperty property 1600
ExecAppName property 1278
ExecCount property 1345
ExecLogging property 1278
Execute method
Business Interlink class 273
SearchQuery class 1468
SQL class 1572
ExecuteEdits method
Message class 748
Record class 1386, 1399
Exists method 1660
Expand method 1636
ExpandNode method 32
ExpImageName property 1647
ExpirationDate property 1188
ExpireDate property 1851
ExpireMeta property 601
ExplainText property 1525
explicit dimension sets
adding to analytic model 50
analytic model 63
copying 53
deleting 58
renaming 69
ExplicitDimensionSet
AttachDimension method 76
DetachDimension method 76
GetDimensionNames method 77
Name property 77
ExpNum property
ColumnNumber class 1307
QueryExpression class 1332
ExportLocation property 1601
Expr1Expression property 1319
Expr1Field property 1319
Expr1Type property 1319
Expr2Constant1 property 1320
Expr2Constant2 property 1320
Expr2Expression1 property 1320
Expr2Expression2 property 1321
Expr2Field1 property 1321
Expr2Field2 property 1321
Expr2List property 1322
Expr2Subquery property 1322
Expr2Type property 1322
expression blocks
adding 112
getting 113
Expression property 105
ExpressionBlock class
creating 112
Expressions property 1290
external functions, declaring 168
external interfaces, declaring public 156
ExternalMessageID property 812
ExternalOperationName property 812
ExternalUserName property 812
ExternalUserPassword property 813
ExtractFieldNames method 1877
ExtractFieldValues method 1877
extractMap method 1886
ExtraOptions property 1483

F

FaultCode property 1557
FaultCodeS property 1557
faults
AddFault method 1550
FaultCode property 1557
FaultCodeS property 1557
FaultString property 1558
SOAPDoc objects 1545
FaultString property 1558
Favorite class
properties 1165
See Also Favorite class properties
understanding 1165
Favorite class properties
CRefName 1165
Label 1165
QualifiedURL 1165
SequenceNumber 1165
URL 1166
Favorite collection
methods 1166
See Also Favorite collection methods
properties 1169
understanding 1166
Favorite collection methods

Copyright © 1988-2007, Oracle. All rights reserved.

DeleteItem 1166
First 1167
InsertItem 1167
ItemByLabel 1168
Next 1168
Save 1169
Favorite collection property 1169
favorites
class 1165
See Also Favorite class
collection 1166
See Also Favorite collection
Favorites property 1042
objects 1165
Favorites class 2056
Favorites collection 2056
Favorites property 1042
Fetch method 1574
FetchIntoRecord method 275
FetchIntoRowset method 277
FetchNextRow method 280
Field class
declaring 492
functions, methods, properties, returns 1932
methods 492
See Also Field class methods
properties 500
See Also Field class properties
shortcut considerations 491
understanding 491
Field class methods
AddDropDownItem method 493
ClearDropDownList method 494
GetAuxFlag method 494
GetLongLabel method 495
GetRelated method 496
GetShortLabel 497
SearchClear 498
SetCursorPos 498
SetDefault 498, 499
Field class properties
DataAreaCollapsed 500
DecimalPosition 501
DisplayFormat 501
DisplayOnly 502
DisplayZero 503
DisplayZeroChanged 503
EditError 504
Enabled 504
FieldLength 504
FormatLength 505
FormattedValue 505
IsAltKey 506
IsAuditFieldAdd 506
IsAuditFieldChg 506
IsAuditFieldDel 506
IsAutoUpdate 506
IsChanged 507
IsDateRangeEdit 507
IsDescKey 507
IsDuplKey 507
IsEditTable 507
IsEditXlat 508
IsFromSearchField 508
IsInBuf 508
IsKey 508
IsListItem 509
IsNotUsed 509
IsRequired 509
IsSearchItem 509
IsSystem 510
IsThroughSearchField 510
IsUseDefaultLabel 510
IsYesNo 510
Label 510
LabelImage 511
LongTranslateValue 512
MessageNumber 512
MessageSetNumber 513
Name 513
OriginalValue 513
ParentRecord 514
PromptTableName 514
SearchDefault 514, 1394
SearchEdit 515
SetComponentChanged 515
ShortTranslateValue 516
ShowRequiredFieldCue 517
SmartZero 517
SqlText 517
StoredFormat 518
Style 518
SystemIDFieldName 519
TimeStampFieldName 520
Type 520
Value 521
Visible 522
field mapping
getting 89

Copyright © 1988-2007, Oracle. All rights reserved.

setting 93
field objects
declaring/scope 492
understanding 491
FieldCount property 1402
FieldDecimal property 1364
FieldFormat property 1364
FieldLength property
Field class 504
QueryPrompt class 1365
fieldname property 1402
FieldName property
QueryPrompt class 1365
SearchRecordField class 1491
fields
Field class 491
See Also Field class
objects 491
processing numeric fields via ExecuteEdits 749
query definitions 1240
QueryDBRecordFields 1240
understanding 491
using ExecuteEdits for numeric fields 1387
using fixed length files with numeric fields 549
Fields property 1486
FieldType property 1365
File class
access interruptions 525
functions, methods, properties, returns 1936
methods 525
See Also File class methods
processing errors 528
properties 551
See Also File class properties
understanding 523
File class methods
Close 530
CreateRowset 530
Delete 531
GetPosition 525, 532
GetString 533
Open 534
ReadLine 537
ReadRowset 538
SetFileId 540
SetFileLayout 542
SetPosition 525, 543
SetRecTerminator 544
WriteLine 545
WriteRaw 546
WriteRecord 546
WriteRowset 548
WriteString 550
File class properties
CurrentRecord 551
IgnoreInvalidId 551
IsError 552
IsNewFileId 552
IsOpen 553
Name 553
UseSpaceForNull 554
file definition tags 547, 549
file dependant processing
ProcessRequest class FileName property 1217
understanding 1202
using PrcsApi class 1228
File Layout definitions
accessing 527
Application Engine example 566
examples 557
file rowset example 565
reading multiple 569
reading/writing hierarchical data 526
ReadRecord example 560
ReadRowset example 565
understanding 523
using multiple 569
using XML 547, 549
WriteRecord example 558
WriteRowset example 561
writing multiple 571
file objects
declaring/scope 524
security 525
Filename property
MCFBodyPart class 651
MCFOutboundEmail class 698
FileName property 1217
TemplateFile class 1840
TranslationFile class 1842
FileNames property 887
FilePath property
MCFBodyPart class 651
MCFOutboundEmail class 698
FilePathType property

Copyright © 1988-2007, Oracle. All rights reserved.

MCFBodyPart class 652
MCFOutboundEmail class 698
files
definition tags 547
File class 523
See Also File class
file definition tags 549
file dependant processing 1202
See Also file dependant processing
FileName property 1217
FileNames property 887
FileTitles property 887
getAllFileNames method 1230
layout 523
See Also File Layout definitions
objects 524
plain text 526
using file rowsets 565
using fixed length files with numeric fields 549
FileTitles property 887
FileURL property 1851
Fill method 1428
FillAppend method 1430
FillForm method 1878
FillRowset method 943
Filter property 1486
filtering
SearchRecord Options class Filter property 1486
synchronization server OnGetPropertyFilter event 1593
using with synchronization 1594
filters
accessing 90
setting 93
FinalDestinationNode property 813
Find method
Array class 223
Component Interface class 436
Mobile class 841
FindCompIntfcs method 423
FindCRefByName method 1028
FindCRefByURL method 1029
FindCRefForURL method 1030
FindCRefLinkByName method 1031
FindExpression method 1267
FindFolderByName method 1032
FindKeyInfoCollection property 441
FindLeaf method 1661
FindNode method
Tree class 1661
XmlNode class 1776
FindNodes method 1777
FindPgltByName method 1033
FindPortalRegistries method 1020
FindQueries method 1254
FindQueriesDateRange method 1255
FindQueryDBRecords method 1253
FindRoot method 1662
FindRowNum method 981
First method
AssignedPagelet collection 1125
Attribute collection 1090
AvailableCategory collection 1129
AvailablePagelet collection 1133
CompIntfPropInfoCollection collection 453
Component Interface collection 424
Content Reference collection 1085
DynamicCategory collection 1135
Favorite collection 1167
Folder collection 1067
Link collection 1111
Node collection 1049
NodeTemplate collection 1057
Pagelet collection 1152
PageletCategory collection 1143
ParseResult collection 1473
PermissionValue collection 1095
Portal collection 1054
PortalRegistry collection 1044
PSMessages collection 1523
Query collection 1261
Query Metadata collection 1341
QueryCriteria collection 1312
QueryDBRecord collection 1350
QueryDBRecordField collection 1354
QueryExpression collection 1329
QueryField collection 1302
QueryList class 1335
QueryPrompt collection 1361
QueryRecord collection 1296
QueryRecordHierarchy collection 1338
QuerySelect collection 1294
RemoteNode collection 1051
RolePermissionValue collection 1098
Search Language collection 1493
Search Schedule collection 1496

Copyright © 1988-2007, Oracle. All rights reserved.

Search Start Options collection 1504
SearchField collection 1479
SearchIndexes collection 1481
SearchRecordField collection 1489
SearchResult collection 1476
SelectedPagelet collection 1162
TabDefinition collection 1120
UserTab collection 1158
First property 1631
FirstChildLeaf property 1648
FirstChildNode property 1648
FirstStep method 476
Flag property
ColumnNumber class 1307
QueryDBRecordField class 1358
flat table methods 255
Flattening 221
Flush method 1432
FlushRow method 1433
Folder class
adding folders 1172
list of methods, properties, returns 2036
methods 1060
properties 1061
See Also Folder class properties
understanding 1059
Folder class methods 1060
Folder class properties
Attributes 1061
Author 1061
AuthorAccess 1061
Authorized 1061
CascadedPermissions 1062
CascadedRolePermissions 1062
ContentRefs 1062
CreationDate 1063
Description 1063
Folders 1063
IsMobile 1063
IsVisible 1063
Label 1064
Name 1064
OwnerId 1064
ParentName 1064
Path 1065
Permissions 1065
Product 1065
PublicAccess 1066
RolePermissions 1065
SequenceNumber 1066
ValidFrom 1066
ValidTo 1066
Folder collection
list of methods, properties, returns 2038
methods 1067
See Also Folder collection methods
properties 1070
understanding 1066
Folder collection methods
DeleteItem 1067
First 1067
InsertItem 1068
ItemByName 1068
Next 1069
Folder collection properties 1070
folder navigation
FolderNavObject property 1042
IsFolderNavigation property 1042
Folder property 1278
FolderName property 1830
report class 1851
FolderNavObject property 1042
folders
adding 1172
collection 1066
See Also Folder collection
deleting 1017
FindFolderByName method 1032
Folder class 1059
See Also Folder class
Folder collection 1066
Folder property 1278
objects 1059
PortalFolder property 1221
ReportFolder property 1190
RootFolder property 1043
SetItemFolder method 1213
understanding 998
Folders property 1063
FontName property 1883
FontSize property 1884
FOProcessor class
constructor 1864
methods 1871
FOProcessor methods
GenerateMergedOutput 1871
GenerateMergedOutputFO 1872
GenerateOutput 1873
GenerateOutputFO 1874
FOProcessor properties

Copyright © 1988-2007, Oracle. All rights reserved.

ConfFile 1875
ConfProp 1876
Locale 1876
Format property
ColumnNumber class 1307
CompIntfPropInfoCollection objects 454
QueryDBRecordField class 1359
FormatLength property 505
FormatResultString method 1267
FormattedValue property 505
FormatType property 85
FormProcessor class
constructor 1865
methods 1877
properties 1879
FormProcessor methods
ExtractFieldNames 1877
ExtractFieldValues 1877
FillForm 1878
FormProcessor properties
ConfFile 1879
ConfProp 1880
Locale 1880
FOUtility class
constructor 1865
methods 1881
property 1882
FOUtility methods
GenerateFO 1881
MergeFOs 1881
FOUtility property
Prop 1882
freeze column mode 131
From property
MCFEmail class 654
MCFOutboundEmail 699
FSOpts property 1484
FullURI property 601
Func class 620
FunctionCall class
AddArgument method 117
creating 113
GenerateRule method 117
GetArguments method 118
methods 117
Name property 118
properties 118
Type property 118
functions, *See* PeopleCode functions
CommitWork 915, 916
CreateOptInterface 935, 962
DeleteOptProbInst 920, 936
DoSaveNow 915, 916
GetOptEngine 918, 919, 937
GetOptProbInstList 938
InsertOptProbInst 920, 939
IsValidOptProbInst 941
MessageBox 921
sending optimization status 926
WinMessage 921

G

Gantt chart
creating 340
DateTimeAxis methods 397
example 330, 403
methods 380
properties 394
specifying Date Time Axis formats 341
understanding 338
using Gantt Glyphs 342
garbage collection
Java programs 626
general XML Publisher constructors
QueryDS 1858
RowsetDS 1858
Generate functions 591
Generate method 203
GenerateFO method 1881
GenerateMergedOutput method 1871
GenerateMergedOutputFO method 1872
GenerateOutput method
EFTProcessor class 1868
FOProcessor class 1873
GenerateOutputFO 1874
GenerateRule method
assignment class 104
comparison class 106
constant class 109
cube class 111
FunctionCall class 117
MemberReference class 119
Operation class 120
RuleDefn class 102
variable class 122
GenerateXSL method
EFTProcessor class 1869
RTFProcessor class 1892
GenerateXSLXliff method 1893

Copyright © 1988-2007, Oracle. All rights reserved.

generic base classes 184
GenFormattedXmlString method 1758
GenXMLPartString method 751
GenXmlString method
  XmlDoc class 1759
  XmlNode class 1778
GenXMLString method
  Incoming Business Interlink class 308
  Message class 751
Get keyword 176
Get language constructs 187
Get method
  analytic type definition 144
  AnalyticModelDefn class 60
  Array class 224
  Component Interface class 437
  DataSourceDefn class 1813
  Mobile class 842
  ReportDefn class 1818
  RowsetCache class 1456
  TemplateDefn class 1834
GetAbsoluteContentURL method 1033
GetActiveTemplateFile method 1834
GetActiveTemplatesByDistributionChannel method 1818
GetAdditionalUserInfo method 725
GetAggregateMapping method 87
getAllFileNames method 1230
GetAnalyticModel method 17
GetAndExpandTemplate method 891
GetArchData method 787
GetArchIBInfoData method 788
GetArguments method 118
GetASyncBPELProcessInstanceUrl method 245
GetAttachmentContentID method
  IBInfo class 804
GetAttachmentProperty method
  IBInfo class 805
GetAttachments method 670
GetAttributeName method
  Incoming Business Interlink class 309
  XmlNode class 1778
GetAttributeValue method
  Incoming Business Interlink class 310
  XmlNode class 1779
GetAuxFlag method 494
GetBodyPart method 687
GetBPELConsoleUrl method 249
GetBPELDomain method 249
GetBPELProcessBrowserUrl method 246
GetCauses method 79
GetCDataValue method 1779
GetCDataValues method 1780
GetCellProperties method 25
GetChildNode method 1781
GetCircularDeps method 80
GetColumn method
  AnalyticGrid class 133
  Grid class 578
GetCompIntfc method 424, 839
GetConnectorPropertiesName method 823
GetConnectorPropertiesType method 823
GetConnectorPropertiesValue method 823
GetContentBody method 596
GetContentString method 752
GetContentType method 687
GetCookie method 606
GetCookieNames method
  Request class 597
  Response class 606
GetCookieValue method 597
GetCount method
  Business Interlink class 281
  MCFGetMail class 657
  MCFMultiPart class 687
GetCube method 60
GetCubeCollection
  analytic grids 133
GetCubeCollection method
  AnalyticModel class 25
  cube collection 61
GetCubeCollectionNames method 61
GetCubeNames method
  AnalyticModelDefn class 62
  CubeCollectionDefn class 88
GetCurrEffRow method 1434
GetData method 33, 1814
GetDataSource method 1818
GetDatasourceDefnList method 1843
GetDate method 945
GetDateArray method 945
GetDateTime method 946
GetDateTimeArray method 947
GetDefaultHPTabOID method 1034
GetDefaultTempate method 1819
GetDimension method 62
GetDimensionAggregate method 81

Copyright © 1988-2007, Oracle. All rights reserved.

GetDimensionNames method
AnalyticModelDefn method 63
CubeCollectionDefn class 88
CubeDefn class 82
ExplicitDimensionSet class 77
GetDimFilter method 34
GetDimSort method
CubeCollection class 35
CubeCollectionDefn class 89
GetDoc method 282
GetDummyRows property 442
GetEffectiveItem method
description 446
using 421
GetEffectiveItemNum method 446
GetEffects method 81
GetElement method 1781
GetElements method 1782
GetElementsByTagName method
XmlDoc class 1759
XmlNode class 1783
GetElementsByTagNameNS method 1784
GetEmailCount method 657
GetErrorImageName method 725
GetExplicitDimensionSet method 63
GetExplicitDimensionSetNames method 64
GetField method
QueryRecord class 1299
Record class 1388
GetFieldCount method 285
GetFieldMapping method 89
GetFieldType method 286
GetFieldValue method 287
GetFile method
TemplateFile class 1838
TranslationFile class 1841
GetFilter method 90
GetFrom method 670
GetHeader method
MCFBodyPart class 645
MCFOutboundEmail class 690
Request class 597
Response class 607
GetHeaderCount method
MCFBodyPart class 646
MCFOutboundEmail class 691
GetHeaderName method
MCFBodyPart class 646
MCFOutboundEmail class 691
GetHeaderNames method
MCFBodyPart class 647
MCFOutboundEmail class 692
Request class 598
Response class 607
GetHeaderValues method
MCFBodyPart class 647
MCFOutboundEmail class 692
GetHelpURL method 598
GetHistoryItems property 442
GetIBInfoData method 788
GetImageURL method 607
GetIndexes method 111
GetInputDocs method 288
GetJavaScriptURL method 608
GetKeyInfoCollection property 442
GetLaunchURL method 727
GetLayout method 35
GetListViewAttrs method 861
GetLocalNode method 1020
GetLogicalQueue method 1741
GetLongLabel method 495
GetMapFile method 1839
GetMembers method 26
GetMessage method 789
GetMessageErrors method 790
GetMessageId method 249
GetModel method 144
GetModelNames method 145
GetMsgSchema method 791
GetName method 842
GetNewEffRow method 1434
GetNextDoc method 289
GetNextEffRow method 1409
GetNode method
IBUtil class 250
Incoming Business Interlink class 311
GetNodes method 1021
GetNumber method 948
GetNumberArray method 949
GetNumberOfConnectorProperties method 824
GetNumberOfQueryStringArgs method 824
GetNumberOfRoutings method 250
GetOffLineImageName method 726
GetOnlineImageName method 726
GetOperationType method 246
GetOptEngine function 918, 919, 937

Copyright © 1988-2007, Oracle. All rights reserved.

GetOptProbInstList function 938
GetOrganizer method 64
GetOrganizerNames method 65
GetOutDestFormatString method 1819
GetOutputDocs method 291
GetParameter method 598
GetParameterNames method 599
GetParameterValues method 599
GetParent method 843
GetParmDate method 963
GetParmDateArray method 964
GetParmDateTime method 964
GetParmDateTimeArray method 964
GetParmInt method 966
GetParmIntArray method 966
GetParmName method 1554
GetParmNumber method 965
GetParmNumberArray method 965
GetParmString method 967
GetParmStringArray method 967
GetParmTime method 968
GetParmTimeArray method 968
GetParmValue method 1555
GetPartAliasName method 752
GetPartName method 753
GetPartNames method 100
GetPartRowset method 753
GetParts method 670
GetPartVersion method 754
GetPartXMLDoc method 754
GetPeerDefaultAttributesByName method 843
GetPersistAggregate method 90
GetPortalRegistry method 1021
GetPosition method 525, 532
GetPreviousDoc method 292
GetPriorEffRow method 1410
GetProperty method 1891
GetPropertyAttrsByName method 844
GetPropertyByName method
  Component Interface class 437
  Mobile class 845
GetPropertyInfoByName method 845
GetPSQueryPromptRecord method 1820
GetQualifiedURL method 1034
GetQuery method 1255
GetQuerySecurityProfile method 1256
GetQueryString method 755
GetQueryStringArgName method 824
GetQueryStringArgValue method 825
GetRecord method
  AnalyticModelDefn class 145
  Row class 1410
GetRecordNames method 146
GetRelated method 496
GetRemoteNodes method 1022
GetReportDefnList method 1844
GetReportList method 1853
GetResponseStatus method 898
GetRow method 1435
GetRowCount method 36
GetRowset method
  Message class 755
  Row class 1412
GetRule method
  CubeDefn class 82
  UserFunctionDefn class 97
GetRuleExpressions method 113
GetSampleData method 1814
GetSchema method 1815
GetSearchIndexes method 1466
GetSearchQuery method
  PortalRegistry class 1467
  Session class 1466
GetSegment method 757
GetSelectedField method 149
GetSender method 671
GetShortLabel method 497
GetSlice method 36
GetSolution method 982
GetSolutionDetail method 984
GetStatus method
  Business Interlink class 294
  IBUtil class 251
  MCFIMInfo class 727
GetString file class method 533
GetString method 950
GetStringArray method 950
GetStyleSheetURL method 608
GetSubstitution method 485
GetSyncBPELProcessInstanceUrl method 246
GetSyncIBInfoData method 791
GetSyncLogData method 792
GetTemplate method 1821
GetTemplateDefnList method 1846
GetTemplateFile method 1835
GetTemplateFileList method 1835
GetTemplateList method 1821
GetTime method 952

Copyright © 1988-2007, Oracle. All rights reserved.

GetTimeArray method 952
GetTimeDiff method 1741
getting
RuleExpression 113
GetTopParent method 846
GetTraceLevel method 953
GetTransactionId method 251
GetTranslationFile method 1839
GetTranslationFileList method 1840
GetTree method 1617
GetTree methods 27
GetTreeStructure method 1618
GetUnknownImageName method 726
GetUnparsedHeaders method 648
GetUserFunction method 65
GetUserFunctionNames method 66
GetValue method 295
GetXMLData method
QueryDS class 1859
RowsetDS class 1863
GetXmlDoc method 757
GetXMLSampleData method 1860
GetXSDSchema method
QueryDS class 1861
RowsetDS class 1863
global
declaring Business Interlink objects 260
variables in application classes 178
See Also global variables
global variables
availability with PortalRegistry classes 1019
going out of scope with SQL objects 1570
using in application classes 178
GlobList property
Search FS Options class 1502
Search HTTP Options class 1500
GlobListType property
Search FS Options class 1502
Search HTTP Options class 1500
glossary 2101
GoToStep method 476
GrantPermissionForComponent method 1035
GrantPermissionForScript method 1036
Greeting property 1155
Grid class
functions, methods, properties, returns 1938
methods 578
properties 580
shortcut considerations 576
understanding 575
using in PeopleCode 576
grid columns
class 580
See Also GridColumn class
gridcolumn property 580
objects 577
See Also GridColumn objects
grid objects
declaring/scope 577
understanding 576
GridColumn class
list of properties, returns 1939
properties 581
See Also GridColumn class properties
understanding 580
GridColumn class properties
Enabled 581
Label 581
Name 582
Visible 582
GridColumn objects
declaring/scope 577
understanding 580
gridcolumn property 580
GridLines property 355
grids 130
See Also analytic grids
columns 577
See Also grid columns
Grid class 575
See Also Grid class
GridLines property 355
objects 576
See Also grid objects
understanding 575
GUID property 772

H

HasChildLeaves property 1648
HasChildNodes property 1648
HasChildren property 1648
HasDefaultValue property 856
HasDetailRanges property 1678
HasLegend property 371

Copyright © 1988-2007, Oracle. All rights reserved.

HasLockedBranches property 1679
HasNextSib property
  Leaf class 1626
  Node class 1649
HasPrevSib property
  Leaf class 1626
  Node class 1649
HavingCriteria property 1291
HeaderNode property 1558
headers
  comments about the class 183
  comments for methods 182
  SOAP header examples 1561
  SOAPDoc objects 1542
HeadingText property
  ColumnNumber class 1308
  QueryPrompt class 1366
HeadingType property
  QueryField class 1308
  QueryPrompt class 1366
HeadingUniqueFieldName property 1308
Height property 356
HelpID property
  Pagelet class 1148
  TabDefinition class 1116
HelpLink property 994
HideAllRows method 1436
hints, data 347
HitCount property 1470
Homepage property 1042
homepages
  accessing the Homepage tab 1113
  DeleteHomepage method 1028
  deleting templates 1017
  Homepage property 1042
  understanding 1000
  UserHomepage class 1154
    See Also UserHomepage
HostNodeName property 1053
HtmlText property
  Content Reference class 1075
  TabDefinition class 1116
HTTP
  HTTPMethod property 602
  HTTPOpts property 1484
  Search HTTP Options 1499
    See Also Search HTTP Options
HTTPMethod property 602
HTTPOpts property 1484
HTTPSessionId property 813

I

IBConnectorInfo property 810
IBException class
  instantiating 772
IBException property 772
IBInfo
  class 802
    See Also IBInfo class
  objects 802
IBInfo class
  adding attachments 802
  clearing attachments 803
  content ID 804
  deleting attachments 804
  getting attachment properties 805
  list of methods, properties, returns 1957
  methods 802
    See Also IBInfo class methods
  properties 810
    See Also IBInfo class properties
  setting attachment properties 809
  understanding 802
IBInfo class methods
  AddAttachment 802
  ClearAttachments 803
  DeleteAttachment 804
  GetAttachmentContentID 804
  GetAttachmentProperty 805
  LoadConnectorProp 807
  LoadConnectorPropFromNode 807
  LoadConnectorPropFromRouting 808
  LoadConnectorPropFromTrx 808
  SetAttachmentProperty 809
IBInfo class properties
  AppServerDomain 810
  ConnectorOverride 811
  ConversationID 811
  Cookies 811
  DestinationNode 812
  ExternalMessageID 812
  ExternalOperationName 812
  ExternalUserName 812
  ExternalUserPassword 813
  FinalDestinationNode 813
  HTTPSessionId 813
  IBConnectorInfo 810
  InReplyToID 813
  MessageChannel 813
  MessageName 814

Copyright © 1988-2007, Oracle. All rights reserved.

MessageQueue 814
MessageType 814
MessageVersion 814
NodeDN 815
NonRepudiationID 815
NumberOfAttachments 815
OperationType 815
OperationVersion 815
OrigNode 816
OrigProcess 816
OrigTimeStamp 816
OrigUser 816
PublicationID 816
RequestingNodeDescription 817
RequestingNodeName 817
ResponseAsAttachment 817
SegmentsUnOrder 817
SourceNode 817
SyncServiceTimeout 818
TransactionID 818
UserName 818
VisitedNodes 819
WSA_MessageID 819
WSA_ReplyTo 819
IBInfo objects 802
IBInfo property 773
IBNode property
MCFGetMail class 666
MCFInboundEmail class 674
IBUtil class
methods 248
understanding 248
IBUtil class methods
GetBPELConsoleUrl 249
GetBPELDomain 249
GetmessageId 249
GetNode 250
GetNumberOfRoutings 250
GetStatus 251
GetTransactionId 251
IBUtil constructor 244
ID property 21
ReportDefn class 1830
IgnoreInvalidId property 551
ImageFile property 1896
ImageFileLowerLeftX property 1896
ImageFileLowerLeftY property 1896
ImageFileUpperRightX property 1896
ImageFileUpperRightY property 1897
ImageMap property 354
ImageName property 1626
Import function 188
Importance property
MCFEmail class 654
MCFOutboundEmail 699
importing
application class names 170
BPEL classes 242
class names in general 185
Import function 188
mail classes 630
MCFMailStore class 676
Notification class 885
Notification classes 880
PrcsApi class 1229
universal queue classes 1709
InActive property 135
InBoundPublish method 758, 793
Incoming Business Interlink class
methods 303
See Also Incoming Business Interlink class methods
properties 259, 313
See Also Incoming Business Interlink class properties; Incoming Business Interlink properties
Incoming Business Interlink class methods
AddAttribute 303
AddComment 304
AddProcessInstruction 305
AddText 306
CreateElement 307
GenXMLString 308
GetAttributeName 309
GetAttributeValue 310
GetNode 311
ParseXMLString 312
using 259
Incoming Business Interlink class properties
AttributeCount 313
ChildNodeCount 313
NodeName 314
NodeType 315
NodeValue 316
using 259
IncrementalView property 1486
Index property 1798
Indexes property 1470
indexing
BuildSearchIndex method 1467

Copyright © 1988-2007, Oracle. All rights reserved.

GetSearchIndexes method 1466
Index property 1798
Indexes property 1470
IndexName property 1471
search indexes 1459
See Also search indexes
IndexName property 1471
IndirectionMethod property 1696
InError property 871
Init method 969
InputRowset method 255, 297
InReplyToID property 813
Insert method 1390
InsertCDataSection method 1785
InsertChildLeaf method 1637
InsertChildNode method 1637
InsertChildRecord method 1638
InsertComment method 1786
InsertDynChildLeaf method 1639
InsertDynSib method 1620
InsertElement method 1787
InsertElementNS method 1789
InsertEnabled property 1448
InsertEntityReference method 1790
InsertItem method
AssignedPagelet collection 1125
Attribute collection 1090
Component Interface class 422
Content Reference collection 1085
Data collection 447, 864
DynamicCategory collection 1136
Favorite collection 1167
Folder collection 1068
NodeTemplate collection 1058
Pagelet collection 1152
PageletCategory collection 1143
PermissionValue collection 1095
RolePermissionValue collection 1099
Search Language collection 1493
Search Schedule collection 1496
Search Start Options collection 1505
SearchRecordField collection 1489
SelectedPagelet collection 1163
TabDefinition collection 1121
UserTab collection 1159
InsertNode method 1791
InsertOptProbInst function 920, 939
InsertProcessInstruction method 1792
InsertRoot method 1663
InsertRow method 1437
InsertSib method
Leaf class 1621
Node class 1640
InsertSibRecord method 1640
InsertText method 1794
INSTALLATION table 521
instance variables
creating public properties 174
declaring private 160
declaring private in application classes 178
overriding properties 175
passing as function arguments 169
running constructors 166
instanceid property 904
InstanceID property
WSWorklistEntry class 909
instant messaging
adding users 724
checking user status 725
MCFIMInfo class 723
See Also MCFIMInfo class
removing users 728
supported networks 723
Instruction property 894
instselecteddttm property 904
inststatus property 904
insttimeoutddttm property 905
instworkeddttm property 905
IntBroker
class 783
See Also IntBroker class
objects 783
IntBroker class
list of methods, properties 1956
methods 784
See Also IntBroker class methods
understanding 783
IntBroker class methods
Cancel 784
ConnectorRequest 785
ConnectorRequestUrl 786
DeleteOrphanedSegments 787
GetArchData 787
GetArchIBInfoData 788
GetIBInfodata 788
GetMessage 789
GetMessageErrors 790
GetMsgSchema 791
GetSyncIBInfoData 791

Copyright © 1988-2007, Oracle. All rights reserved.

GetSyncLogData 792
InBoundPublish 793
IsOperationActive 794
MsgSchemaExists 794
Publish 795
Resubmit 796
SetErrorMessage 797
SetStatus 798
SyncRequest method 799
Update 801
UpdateXmlDoc 801
IntBroker objects 783
Inteface language construct 189
integration
API Repository 194
Business Interlink class 253
Component Interface classes 411
Integration Broker 642
Integration Broker 642
Integration Broker BPEL classes, *See* BPEL classes
InteractiveMode property 443
interfaces
component 411
See Also Component Interface
declaring public external 156
implementing 165
PeopleCode 164
understanding 165
InternalURL property 1731
Internet scripts, *See* iScripts
InvalidAddresses property 699
IsActive property
Message class 773
PhysicalQueue class 1731
IsAltKey property 506
IsAttachment property
MCFBodyPart class 652
SeachRecordField class 1491
IsAuditFieldAdd property 506
IsAuditFieldChg property 506
IsAuditFieldDel property 506
IsAuthenticationReqd property
MCFOutboundEmail class 700
SMTPSession class 709
IsAutoUpdate property 506
IsBranched property
Node class 1649
Tree class 1679
IsChanged property
Field class 507
Leaf class 1627
Node class 1649
Record class 1402
Row class 1415
Tree class 1679
IsCollection property
CompIntfPropInfoCollection objects 455
PropertyInfo object 857
IsCreatedViaWebService 905
iScript classes
Cookie 595
See Also Cookie class
creating web libraries 587
Request 594
See Also Request class
Response 595
See Also Response class
returns 1939
scope 593
system variables 1939
understanding 585
URLs vs. URIs 587
iScripts
adding security 589
avoiding Java applets/browser plug-ins 590
classes 585
See Also iScript classes
creating 587
creating charts 407
generating charts 334
handling errors 593
objects 594
understanding 585
using styles/stylesheets 590
viewing 590
when to use 589
IsCut property
Leaf class 1627
Node class 1649
IsDateRangeEdit property 507
IsDefault property 1047
IsDeleted property
Leaf class 1627
Node class 1649
Record class 1403
Row class 1415
IsDelta property 774

Copyright © 1988-2007, Oracle. All rights reserved.

IsDescKey property 507
IsDomainRestricted property 1507
IsDrillable property 356
IsDuplKey property 507
IsEditError property
Message class 774
Record class 1403
Row class 1416
Rowset class 1449
IsEditTable property 507
IsEditXlat property 508
IsEmailServerAvailable method 684
IsEmpty property 775
IsError property 552
IsFolderNavigation property 1042
IsFromSearchField property 508
IsHideActionBar property 1116
IsHideMinimize property 1148
IsHostRestricted property 1507
IsInBuf property 508
IsInserted property
Leaf class 1627
Node class 1650
IsKey property 508
IsLayoutLocked property 1117
IsListItem property 509
IsLocal property
Message class 775
Node class 1047
Portal class 1053
IsMinimized property 1161
IsMobile property
Content Reference class 1075
Content Reference Links class 1105
Folder class 1063
IsModelActive OptInterface class methods 985
IsNew method 846
IsNew property 1416
IsNewAndNeverSaved method 847
IsNewFileId property 552
IsNotUsed property 509
IsNull property
XmlDoc class 1765
XmlNode class 1798
IsOkToSendPartial property 700
IsOpen property
AESection class 10
File class 553
SQL class 1579
Tree class 1680
IsOperationActive method 794
IsOperationActive property 776
IsParts property 776
IsPartsStructured property 776
IsPeerReference property 857
IsPlainImage property 357
IsPrompt property 1337
IsPublic property 1816
IsQueryTree property 1680
IsReadOnly property 455
ReportDefn class 1831
TemplateDefn class 1836
IsReferenceUnresolved property 1601
IsRefToParent property 857
IsRenameable property 1117
IsRequest property 777
IsRequired property 509
IsReturnReceiptReqd property 700
IsRoot property 1650
IsSameAs method 847
IsSearchItem property 509
IsSet method
ListViewAttrs class 866
PeerDefaultAttributes class 873
PropertyAttrs class 869
IsSimpleApplicationProperty property 858
IsSourceNodeExternal property 777
IsStructure property 777
IsSubTemplate property 1837
IsSystem property 510
IsSystemProperty property 858
IsThroughSearchField property 510
IsUseDefaultLabel property 510
IsValid property
AnalyticModelDefn class 73
Tree class 1680
IsValidOptProbInst function 941
IsVerityField property 1492
IsVersionChanged property 1680
IsVirtual property 85
IsVisible property
Content Reference class 1076
Content Reference Links class 1106
Folder class 1063
IsWholeTree property 1681
IsWordIndex property 1492
IsXlatExpression property 1332
IsYesNo property 510

Copyright © 1988-2007, Oracle. All rights reserved.

Item method
Bindings collection 202
CI collection 860
ClassInfo collection 206
CompIntfPropInfoCollection collection 453
Component Interface collection 425
Data collection 448, 862
Level collection 1630
MethodInfo collection 208
Namespaces collection 203
PortalRegistry collection 1045
PropertyInfo collection 211
PropertyInfoCollection class 854
PSMessages collection 1523
Query collection 1261
Query Metadata collection 1342
QueryCriteria collection 1313
QueryDBRecord collection 1350
QueryDBRecordField collection 1355
QueryExpression collection 1329
QueryField collection 1302
QueryList class 1335
QueryPrompt collection 1361
QueryRecord collection 1297
QueryRecordHierarchy collection 1338
QuerySelect collection 1294
SearchResult collection 1476
ItemByAlias method 1297
ItemByExpNum method 1303
ItemByKeys method 448
ItemByLabel method 1168
ItemByName method
AssignedPagelet collection 1126
Attribute collection 1091
AvailableCategory collection 1129
AvailablePagelet collection 1133
ClassInfo collection 206
Content Reference collection 1086
DynamicCategory collection 1136
Folder collection 1068
MethodInfo collection 209
Namespaces collection 204
Node collection 1049
NodeTemplate collection 1058
Pagelet collection 1153
PageletCategory collection 1144
PermissionValue collection 1096
Portal collection 1054
PropertyInfo collection 212
Query collection 1262
Query Metadata collection 1342
QueryCriteria collection 1313
QueryDBRecord collection 1351
QueryDBRecordField collection 1355
QueryExpression collection 1330
QueryPrompt collection 1362
QueryRecordHierarchy collection 1338
RemoteNode collection 1051
RolePermissionValue collection 1099
Search Language collection 1494
Search Schedule collection 1497
Search Start Options collection 1505
SearchField collection 1479
SearchIndexes collection 1481
SearchRecordField collection 1490
SelectedPagelet collection 1163
TabDefinition collection 1121
UserTab collection 1159
ItemByNameAndAlias method 1303
ItemBySelNum method 1295
ItemNum property 450

J

JAR files 616
Java
applets 590
calling application classes 623
class 615
See Also Java class
creating objects 619
See Also Java objects
debugging 627
functions 615
See Also Java functions
JAR files 616
Java Debugging Environment (JDB) 627
Java runtime engine (JRE) 616
managing states 618
mapping PeopleCode types 624
version/platform support 616
Java applets 590
Java class
arrays 626
calling PeopleCode 620
copying data 626
Func 620

Copyright © 1988-2007, Oracle. All rights reserved.

garbage collector 626
getting system classes 620
handling errors 627
list of functions, methods, properties, returns 1943
Name 621
naming 616
setting up the system for 616
SysCon 620
SysVar 620
understanding 615
using PeopleCode functions 625
Java Debugging Environment (JDB) 627
Java functions
error handling 627
understanding 615
Java objects
creating 619
declaring/scope 628
Java runtime engine (JRE) 616
JDB 627
JobName property 1217
jobs
defining 1194
displaying items 1204
displaying status 1205
JobName property 1217
jobsets 1194
scheduling with item changes 1227
scheduling without item changes 1227
jobsets
displaying items 1204
displaying status 1205
scheduling 1211
understanding 1194
Join method 225
JoinAlias property 1300
JoinFieldName property 1300
joins
AllowAnyJoin property 1346
JoinAlias property 1300
JoinFieldName property 1300
JoinType property 1300
MaxJoins property 1349
JoinType property 1300
JRE 616

K

Key property
CompIntfPropInfoCollection objects 455
SearchResult class 1478
KeyBranchName property 1681
KeyEffDt property 1681
KeyName property
Tree class 1681
Tree Structure class 1698
keys
CompIntfPropInfoCollection Key property 455
Get/Set keywords 176
SearchResult class Key property 1478
setting for a Component Interface 414
KeySetId property 1682
KeyUserKeyValue property 1682
KnowledgeBase property 1471

L

Label property
AnalyticGrid class 136
AttributeValue class 1087
Content Reference class 1076
Content Reference Links class 1106
Favorite class 1165
Field class 510
Folder class 1064
Grid class 580
GridColumn class 581
ListViewAttrs class 867
Pagelet class 1148
PageletCategory class 1140
PeerDefaultAttributes class 874
PropertyAttrs class 871
TabDefinition class 1117
UserTab class 1157
LabelImage property 511
LabelLong property 456
labels
chart considerations 333
LabelShort property 456
LangCount property 1366
language constructs, application class, *See* application class language constructs
Language property
Agent class 1721
MCFInboundEmail class 674
NotificationAddress class 890
NotificationTemplate class 894
SearchQuery class 1472

Copyright © 1988-2007, Oracle. All rights reserved.

Task class 1737
language_cd property 887
LanguageCd property
ProcessRequest class 1218
Search Language class 1495
LanguageCD property
RegionalSettings class 1530
LanguageCode property 1842
TemplateDefn class 1837
languages
Agent class Language property 1721
application class language constructs 186
See Also application class language constructs
MapLanguageCd property 1495
Notification class language_cd property 887
NotificationAddress class Language property 890
NotificationTemplate class Language property 894
ProcessRequest class LanguageCd property 1218
RegionalSettings LanguageCD property 1530
related languages for a Component Interface 418
Search Language class LanguageCd property 1495
searches 1492
See Also Search Language
SearchIndex class Language property 1484
SearchQuery class Language property 1472
setting for email 640
setting for queries 1246
Task class Language property 1737
Languages property 1484
Last property 1631
LastChildLeaf property 1650
LastChildNode property 1650
LastExecDtTm property 1346
LastSQLErrorCode property 1278
LastSyncDateTime property 1601
LastSyncRowCount property 1602
LastUpdatedBy property
DataSourceDefn class 1816
LastUpdateDTTM property
DataSourceDefn class 1816
ReportDefn class 1831
TemplateDefn class 1837
LastUpdateOprID property
ReportDefn class 1831
TemplateDefn class 1837
LastUpdDttm property 1279
LastUpdOprId property 1279
LaunchASyncBPELProcess method 247
LaunchSyncBPELProcess method 248
LayoutBehavior property
AssignedPagelet class 1123
AvailablePagelet class 1131
TabDefinition class 1117
leaf
ChildLeafCount property 1647
class 1619
See Also Leaf class
FindLeaf method 1661
FirstChildLeaf property 1648
InsertChildLeaf method 1637
InsertDynChildLeaf method 1639
LastChildLeaf property 1650
LeafCount property 1682
LeafExists method 1663
LeafImageName property 1682
LeafOnClipboard property 1682
objects 1619
Leaf class
list of methods, properties, returns 2075
methods 1619
See Also Leaf class methods
properties 1625
See Also Leaf class properties
understanding 1619
Leaf class methods
Cut 1619
Delete 1620
DeleteByRange 1620
InsertDynSib 1620
InsertSib 1621
MoveAsChild 1622
MoveAsChildByName 1623
MoveAsSib 1623
MoveAsSibByRange 1624
PasteSib 1625
UpdateRanges 1625
Leaf class properties
DisplayLevelNumber 1626
Dynamic 1626

Copyright © 1988-2007, Oracle. All rights reserved.

HasNextSib 1626
HasPrevSib 1626
ImageName 1626
IsChanged 1627
IsCut 1627
IsDeleted 1627
IsInserted 1627
NextSib 1627
Parent 1628
PrevSib 1628
RangeFrom 1628
RangeTo 1628
TreeBranchName 1629
TreeEffDt 1629
TreeName 1629
TreeSetId 1629
TreeUserKeyValue 1629
Leaf objects 1619
LeafCount property 1682
LeafExists method 1663
LeafImageName property 1682
LeafOnClipboard property 1682
LegendMaxEntries property 371
LegendPosition property 371
legends, chart
Chart class methods 369
Chart class properties 370
LegendStyle property 371
Len property 239
Length property
ColumnNumber class 1308
CompIntfPropInfoCollection objects 456
QueryDBRecordField class 1359
QueryExpression class 1332
Level class
list of properties, returns 2077
methods 1632
properties 1632
See Also Level class properties
understanding 1632
Level class methods 1632
Level class properties
AllValuesAudit 1632
Description 1633
Name 1633
Number 1633
TreeBranchName 1633
TreeEffDt 1633
TreeName 1633
TreeSetId 1634
TreeUserKeyValue 1634
Level collection
list of methods, properties, returns 2076
methods 1630
See Also Level collection methods
properties 1631
understanding 1629
Level collection methods
Add 1630
Item 1630
Remove 1631
Level collection properties 1631
Level objects 1632
Level property
QueryRecordHierarchy class 1340
Rowset class 1449
LevelComponent property 1698
LevelCount property 1683
LevelMenu property 1698
LevelMenuBar property 1698
LevelMenuItem property 1698
LevelNumber property 1651
LevelPage property 1699
levels
BranchLevel property 1677
class 1632
See Also Level class
collection 1629
See Also Level collection
LevelComponent property 1698
LevelCount property 1683
LevelMenu property 1698
LevelMenuBar property 1698
LevelMenuItem property 1698
LevelNumber property 1651
LevelPage property 1699
Levels property 1683
LevelUse property 1683
objects 1632
ParentLevel property 1686
QueryRecordHierarchy class Level property 1340
Rowset class Level property 1449
Levels property 1683
LevelUse property 1683
licensing
navigation/registration limitations cxxi
Verity/AllHTTPS property 1499
lights-out mode

Copyright © 1988-2007, Oracle. All rights reserved.

showing application messages 931
understanding 921
LimitUnapproved property 1348
LineType property 357
Link class
list of properties, returns 2045
properties 1112
understanding 1112
Link class properties 1112
Link collection
list of methods, properties, returns 2045
methods 1111
properties 1112
understanding 1111
Link collection methods
First 1111
Next 1112
Link collection properties 1112
link objects 1112
LinkDepth property 1500
links
class 1112
See Also Link class
collection 1111
See Also Links collection
content reference 1101
See Also content reference links
creating 1070
Links property 1076
Links property 1076
LinksObjectName property 1113
LinksObjectType property 1113
LinksPortalName property 1113
ListViewAttrs class
list of methods, properties, returns 1964
methods 866
objects 866
properties 867
See Also ListViewAttrs class properties
understanding 835, 866
ListViewAttrs class methods 866
ListViewAttrs class properties
DetailViewLabel 867
Label 867
ListViewLabel 868
Visible 868
ListViewLabel property 868
Load method 17
LoadConnectorProp method 807
LoadConnectorPropFromNode method 807
LoadConnectorPropFromRouting method 808
LoadConnectorPropFromTrx method 808
LoadData method 134
LoadIBContent method 1760
LoadLibrary method 477
LoadXMLPartString method 758
LoadXMLString method 759
local variables
referencing with PortalRegistry objects 1019
supporting for mobile PeopleCode 838
using in catch clauses 483
Locale property
FOProcessor class 1876
FormProcessor class 1880
PDFMapTool class 1887
PDFMerger class 1889
RTFProcessor class 1895
LocalName property 1798
Location property 1484
LockOwner property 1684
LockStatus property 1684
LockTree method 1664
logging
errors for Query classes 1246
errors for Search classes 1464
errors for Tree classes 1612
PortalRegistry classes errors 1004
subscription PeopleCode errors 1512
Logical property 1325
logical queues
class 1727
See Also LogicalQueue class
GetLogicalQueue method 1741
LogicalQueue constructor 1712
MCFFactory class 1728
PhysicalQueue class LogicalQueueID property 1732
LogicalQueue class
list of properties, returns 1992
properties 1727
understanding 1727
LogicalQueue constructor 1712
LogicalQueue property 1728
LogicalQueueID property
LogicalQueue class 1727
PhysicalQueue class 1732

Copyright © 1988-2007, Oracle. All rights reserved.

LogoutURL property 602
LongDescription property 1279
AnalyticModelDefn class 73
LongName property
ColumnNumber class 1309
QueryDBRecordField class 1360
LongTranslateValue property 512
LookupTableName property 1360
LParenLvl property 1325
LTrim property 1579

M

mail
classes 629
See Also mail classes
creating objects 631
servers 635
mail classes
constructors 643
creating mail objects 631
examples 711
Authenticating Email While Sending 721
Creating Email and Overriding SMTP Settings 712
Creating Email Attachments Specifying a URL 719
Creating Email with Attachments 718
Creating HTML Email 714
Creating HTML Email with Images 716
Creating Multi-Part Email With Both Text and HTML Parts 715
Creating Multiple Emails 720
Creating Text Email 711
importing 630
list of functions, methods and properties 1943
MCFBodyPart class 644
See Also MCFBodyPart class
MCFEmail class 653
See Also MCFEmail class
MCFGetMail class 635
See Also MCFGetMail class
MCFMailStore 676
See Also MCFMailStore class
MCFMailUtil 680
See Also MCFMailUtil class
MCFMultiPart 686
See Also MCFMultiPart class
MCFOutboundEmail 688
See Also MCFOutboundEmail class
scope/data types 630
understanding 629
using 631
mail objects, creating 631
mail servers
deleting email 663
double-downloading email 636
retrieving batch email 642
retrieving email 635
understanding errors 642
MailServer property 666
MainTitle property 367
MainTitleOrient property 368
MainTitleStyle property 368
ManagerInfo property 1533
MapFileName property 1840
MapLanguageCd property 1495
Market property 894
MaxAge property 612
MaxDelta property 73
MaxInTreeCriteria property 1349
MaxIterations property 73
MaxJoins property 1349
MaxRowsToFetch property 1349
MaxUnapprovedRows property 1349
MCF mail classes, *See* mail classes
MCFBodyPart class
constructor 643
methods 644
See Also MCFBodyPart Class methods
properties 650
See Also MCFBodyPart Class properties
understanding 644
MCFBodyPart class methods
GetHeader 645
GetHeaderCount 646
GetHeaderName 646
GetHeaderNames 647
GetHeaderValues 647
GetUnparsedHeaders 648
MCFBodyPart Class methods
AddHeader 644
SetAttachmentContent 648
MCFBodyPart class properties
AttachmentURL 650
Charset 650

Copyright © 1988-2007, Oracle. All rights reserved.

ContentType 650
Description 650
Disposition 651
Filename 651
FilePath 651
FilePathType 652
IsAttachment 652
MultiPart 652
Text 653
MCFEmail class
properties 653
See Also MCFEmail class properties
understanding 653
MCFEmail class properties
BCC 653
BounceTo 653
CC 654
From 654
Importance 654
Priority 654
Recipients 655
RefIDs 655
ReplyIDs 655
ReplyTo 655
Sender 656
Sensitivity 656
Subject 656
Text 656
MCFFactory
class 1707
See Also MCFFactory class
objects 1707
MCFFactory class
hierarchy 1707
list of properties, returns 1992
properties 1728
understanding 1728
MCFFactory class properties 1728
MCFFactory constructor 1713
MCFFactory objects 1707
MCFGetMail class
constructor 643
error codes 642
list of methods, properties, returns 1946
methods 656
See Also MCFGetMail class methods
properties 665
See Also MCFGetMail class properties
understanding 635
MCFGetMail class methods
CreateQuarantineFolder 656
errors 642
GetCount 657
GetEmailCount 657
ReadAllEmailHeadersWithAttach 658
ReadEmails 659
ReadEmailsWithAttach 660
ReadEmailsWithUID 661
ReadEmailWithAttach 661
ReadHeaders 662
RemoveEmail 663
RemoveEmails 664
SetMCFEmail 664
MCFGetMail class properties
AttachmentRoot 665
ContentTypes 666
ErrorCount 666
IBNode 666
MailServer 666
Password 667
QuarantineCount 667
QuarantineFolder 667
Status 668
UserID 668
MCFIMInfo
class 723
See Also MCFIMInfo class
objects 724
MCFIMInfo class
methods 724
See Also MCFIMInfo class methods
understanding 723
using 723
MCFIMInfo class methods
AddUser 724
CheckAll 725
GetAdditionalUserInfo 725
GetErrorImageName 725
GetLaunchURL 727
GetOffLineImageName 726
GetOnlineImageName 726
GetStatus 727
GetUnknownImageName 726
RemoveUser 728
MCFIMInfo objects 724
MCFInboundEmail class
methods 668
See Also MCFInboundEmail class methods
MCFInboundEmail class methods

Copyright © 1988-2007, Oracle. All rights reserved.

DumpToFile 668
GetAttachments 670
GetFrom 670
GetParts 670
GetSender 671
ReadFromDatabase 671
SaveToDatabase 672
MCFInboundEmail class properties
AttachList 672
AttachSizes 672
DttmReceived 673
DttmSaved 673
DttmSent 673
IBNode 674
Language 674
MessageID 674
NotifyCC 674
NotifyTo 674
OffsetReceived 675
OffsetSent 675
Server 675
Size 675
Status 675
UID 676
User 676
MCFMailStore class
constructor 644
importing 676
list of methods, properties, returns 1948
methods 676
See Also MCFMailStore class methods
properties 709
See Also MCFMailStore class properties
understanding 676
MCFMailStore class methods
AuthorizeEmailAttach 676
DeleteEmail 677
RetrieveEmail 678
StoreEmail 679
MCFMailUtil class
methods 680
See Also MCFMailUtil class methods
understanding 680
MCFMailUtil class methods
DecodeText 680
DecodeWord 681
EncodeText 682
EncodeWord 683
IsEmailServerAvailable 684
ValidateAddress 685
MCFMultiPart class
methods 686
See Also MCFMultiPart class methods
properties 688
See Also MCFMultiPart class properties
understanding 686
MCFMultiPart class methods
AddBodyPart 686
GetBodyPart 687
GetContentType 687
GetCount 687
MCFMultiPart class properties
SubType 688
MCFOutboundEmail class
constructor 643
methods 688
See Also MCFOutboundEmail class methods
properties 694
See Also MCFOutboundEmail class methods
understanding 688
MCFOutboundEmail class methods
AddAttachment 688
AddHeader 689
GetHeader 690
GetHeaderCount 691
GetHeaderName 691
GetHeaderNames 692
GetHeaderValues 692
Send 693
SetSMTPParam 693
MCFOutboundEmail class properties
BackupSMTPServer 695
BackupSMTPUserName 695
BackupSMTPUserPassword 695
BCC 695
BounceTo 695
CC 696
Charset 696
ContentLanguage 696
ContentType 696
DefaultCharSet 697
Description 697
Disposition 697
ErrorDescription 697
ErrorDetails 698
Filename 698

Copyright © 1988-2007, Oracle. All rights reserved.

FilePath 698
FilePathType 698
From 699
Importance 699
InvalidAddresses 699
IsAuthenticationReqd 700
IsOkToSendPartial 700
IsReturnReceiptReqd 700
MessageNumber 700
MessageSetNumber 701
MultiPart 701
Priority 701
Recipients 702
RefIDs 702
ReplyIDs 702
ReplyTo 702
ResultOfSend 702
Sender 703
Sensitivity 703
SetSMTPParam 694
SMTPPort 703
SMTPServer 703
SMTPUserName 704
SMTPUserPassword 704
StatusNotifyOptions 704
StatusNotifyReturn 704
Subject 705
Text 705
TimeToWaitForResult 705
UsedDefaultConfig 705
UsedPrimaryServer 706
ValidSentAddresses 706
ValidUnsentAddresses 706
Member property 119
MemberReference class
creating 118
Dimension property 119
Member property 119
method 119
properties 119
members
adding 21
getting 26
removing 29
%Menu conditions 423
menus
DetailMenu property 1694
DetailMenuBar property 1695
DetailMenuItem property 1695
GUI/online processing 422
LevelMenu property 1698
LevelMenuBar property 1698
LevelMenuItem property 1698
NodeMenu property 1700
NodeMenuBar property 1700
NodeMenuItem property 1701
SourceMenu property 889
using %Menu conditions 423
MergeFOs method 1881
MergePDFs method 1888
Message class
list of functions, methods, properties, returns 1952
methods 739
See Also Message class methods
properties 769
See Also Message class properties
Message class methods
Clone method 739
CopyPartRowset 740
CopyRowset method 741
CopyRowsetDelta method 742
CopyRowsetDeltaOriginal 744
CreateNextSegment 747
DeleteSegment 748
ExecuteEdits method 748
GenXMLPartString 751
GenXMLString 751
GetContentString 752
GetPartAliasName 752
GetPartName 753
GetPartRowset 753
GetPartVersion 754
GetPartXMLDoc 754
GetQueryString 755
GetRowset 755
GetSegment 757
GetXmlDoc 757
InBoundPublish 758
LoadXMLPartString 758
LoadXMLString 759
Publish method 760
SetEditTable method 749, 762
SetQueryString method 763
SetStatus method 764
SetXmlDoc method 765
SyncRequest method 766
Update method 767
UpdateSegment 768
Message class properties

Copyright © 1988-2007, Oracle. All rights reserved.

ActionName 769
AliasName 769
ChannelName 769
Cookies 770
CurrentSegment 771
DefaultMessageVersion 771
DoNotPubToNodeName 772
GUID 772
IBException 772
IBInfo 773
IsActive 773
IsDelta 774
IsEditError 774
IsEmpty 775
IsLocal 775
IsOperationActive 776
IsParts 776
IsPartsStructured 776
IsRequest 777
IsSourceNodeExternal 777
IsStructure 777
MessageDetail 777
Name 778
NRId 778
OperationName 778
OperationVersion 778
ParentTransactionId 778
PartCount 779
PubID 779
PubNodeName 779
QueueName 780
QueueSeqIf 780
ResponseStatus 780
SegmentCount 780
SegmentsByDatabase 780
Size 781
SubName 782
SubQueueName 782
SubscriptionProcessId 782
TransactionId 783
Version 783
Message classes
understanding 729
message objects
declaring/scope 731
populating 731
Message property
Notification class 887
ParseResult class 1475
message schema
accessing 791
verifying existence 794
message segments
accessing 757
count 780
creating 747
current 771
deleting 748
unordering 817
updating 768
writing to database 780
MessageBox function 921
MessageChannel property 813
MessageDetail property 777
MessageID property 674
MessageName property 814
MessageNumber property
Exception class 488
Field class 512
MCFOutboundEmail class 700
PSMessages class 1525
MessageQueue property 814
messages
copying parts 740
determining parts 776
determining structured parts 776
generating XML part string 751
getting part alias names 752
getting part name 753
getting part rowset 753
getting part version 754
getting XML 754
part count 779
populating parts 758
Messages property
AnalyticInstance class 21
AnalyticModel class 29
AnalyticModelDefn class 74
CubeCollection class 42
MessageSetNumber property 701
Exception class 488
Field class 513
PSMessages class 1525
MessageSeverity property 488
MessageType property
IBInfo class 814
PSMessages class 1525
MessageVersion property 814
messaging

Copyright © 1988-2007, Oracle. All rights reserved.

accessing/creating XML messages 1746
CopyToPSFTMessage method 1752
Exception class MessageNumber property 488
Exception class MessageSetNumber property 488
ExternalMessageID property 812
handlers 730
handling errors 738
Message classes 729
See Also Message classes
MessageChannel property 813
MessageName property 814
MessageNumber property 512
MessageQueue property 814
MessageSetNumber property 513
MessageSeverity property 488
MessageType property 814
MessageVersion property 814
Notification class Message property 887
objects 731
See Also message objects
OPT_CALL message 921
ParseResult class Message property 1475
PSMessages 1004
See Also PSMessages
PSMessages property 1519
PSMessagesMode property 1520
publishing/subscribing to partial records 735
request messages 922
See Also request messages
response messages 926
See Also response messages
rowset-based and nonrowset-based messages 729
sending detailed messages 926
SOAP 1540
structured and unstructured messages 729
subscribing to character fields 735
messaging PeopleCode
deprecated methods and properties 2092
metadata 1453
See Also RowsetCache class
Query class Metadata property 1279
Query Metadata 1344
See Also Query Metadata
using query metadata 1243
metadata classes
analytic model 123
Metadata property 1243, 1279
MetaSQL property 1279
Method function 190
MethodInfo class
list of properties, returns 2017
properties 209
MethodInfo collection
list of properties, returns 2017
methods 208
MethodName property 1559
MethodNode property 1559
methods
AESection class 5
See Also AESection class methods
Agent class 1717
AgentPhysQueueTasks class 1723
AnalyticGrid class 132
AnalyticInstance class 14
AnalyticModel class 21
AnalyticModelDefn class 48
AnalyticTypeDefn class 140
AnalyticTypeRecordDefn class 149
API instantiation 1518
Array class 223
See Also Array class methods
AssignedPagelet collection 1124
See Also AssignedPagelet collection methods
assignment class 104
Attribute collection 1089
See Also Attribute collection methods
AvailableCategory collection 1129
See Also AvailableCategory collection methods
AvailablePagelet collection 1132
See Also AvailablePagelet collection methods
Bindings 202
Bindings collection 202
BPELUtil class 245
Business Interlink 254
See Also Business Interlink methods
Chart class axes 362
See Also Chart class axes methods
Chart class color 361

Copyright © 1988-2007, Oracle. All rights reserved.

See Also Chart class color methods
Chart class data methods 345
See Also Chart class data methods
Chart class Gantt 380
See Also Chart class Gantt methods
Chart class Gantt DateTimeAxis 397
See Also Chart class Gantt DateTimeAxis methods
Chart class legend 369
Chart class overlay 372
See Also Chart class overlay methods
Chart class Scatter 374
See Also Chart class Scatter methods
CI collection 860
ClassInfo collection 206
comparison class 106
CompIntfPropInfoCollection collection 453
See Also CompIntfPropInfoCollection collection methods
Component Interface class 426
See Also Component Interface class methods
Component Interface classes 411
See Also Component Interface classes methods
Component Interface collection 424
See Also Component Interface collection methods
ConnectorInfo collection 819
See Also ConnectorInfo collection methods
constant class 109
constructors 166
See Also constructors
Content Reference class 1070
See Also Content Reference class methods
Content Reference collection 1084
See Also Content Reference collection methods
Content Reference Links class 1101
Crypt class 476
cube class 110
CubeCollection class 30
CubeCollectionDefn class 86
CubeDefn class 78
Data collection 443, 861
See Also Data collection methods
declaring abstract 164
declaring private 160, 185
declaring protected 157
defining for application classes 162
deprecated 2091, 2092
DynamicCategory collection 1134
See Also DynamicCategory collection methods
Exception class 485
See Also Exception class methods
Favorite collection 1166
See Also Favorite collection methods
Field class 492
See Also Field class methods
File class 525
See Also File class methods
Folder class 1060
Folder collection 1067
See Also Folder collection methods
FunctionCall class 117
Grid class 578
header comments 182
IBInfo class 802
See Also IBInfo class methods
IBUtil class 248
Incoming Business Interlink class 303
See Also Incoming Business Interlink class methods
IntBroker class 784
See Also IntBroker class methods
interfaces 165
Leaf class 1619
See Also Leaf class methods
Level class 1632
Level collection 1630
See Also Level collection methods
Link collection 1111
ListViewAttrs class 866
mapping functions to 2085
MCFGetMail class 642
See Also MCFGetMail class methods
MCFIMInfo class 724
See Also MCFIMInfo class methods
MCFMailStore 676
See Also MCFMailStore class methods
MCFMailUtil 680
See Also MCFMailUtil class methods
MCFMultiPart 686
See Also MCFMultiPart class methods
MCFOutboundEmail 688

Copyright © 1988-2007, Oracle. All rights reserved.

See Also MCFOutboundEmail class methods
MemberReference class 119
Message class 739
See Also Message class methods
Method function 190
MethodInfo collection 208
Mobile class 840
See Also Mobile class methods
Namespaces class 205
Namespaces collection 203
naming 170
Node class 1634
See Also Node class methods
Node collection 1049
See Also Node collection methods
NodeTemplate collection 1057
See Also NodeTemplate collection methods
Notification class 886
NotificationTemplate class 891
See Also NotificationTemplate class methods
Operation class 120
OptBase callback 962
OptBase class 962
See Also OptBase class methods
OptEngine class 915
See Also OptEngine class methods
OptInterface class 978
See Also OptInterface class methods
OrganizerDefn class 99
Pagelet class 1145
Pagelet collection 1151
See Also Pagelet collection methods
PageletCategory class 1138
PageletCategory collection 1142
See Also PageletCategory collection methods
ParseResult collection 1473
PeerDefaultAttributes class 873
PermissionValue collection 1094
See Also PermissionValue collection methods
PhysicalQueue class 1729
Portal class 1053
Portal collection 1054
See Also Portal collection methods
PortalRegistry class 1022
See Also PortalRegistry class methods
PortalRegistry collection 1044
See Also PortalRegistry collection methods
PostReport class 1186
See Also PostReport class methods
PrcsApi class 1230
See Also PrcsApi class methods
ProcessRequest class 1203
See Also ProcessRequest class methods
PropertyAttrs class 869
PropertyInfo collection 211
PropertyInfoCollection class 854
PSMessages collection 1522
See Also PSMessages collection methods
Query class 1263
See Also Query class methods
Query collection 1260
See Also Query collection methods
Query Metadata collection 1341
See Also Query Metadata collection methods
QueryCriteria class 1315
See Also QueryCriteria class methods
QueryCriteria collection 1312
See Also QueryCriteria collection methods
QueryDBRecord class 1352
See Also QueryDBRecord class methods
QueryDBRecord collection 1350
See Also QueryDBRecord collection methods
QueryDBRecordField collection 1354
See Also QueryDBRecordField collection methods
QueryExpression collection 1329
See Also QueryExpression collection methods
QueryField class 1305
See Also QueryField class methods
QueryField collection 1302
See Also QueryField collection methods
QueryList class 1334
See Also QueryList class methods
QueryPrompt collection 1361
See Also QueryPrompt collection methods
QueryRecord class 1299

Copyright © 1988-2007, Oracle. All rights reserved.

QueryRecord collection 1296
See Also QueryRecord collection methods
QueryRecordHierarchy collection 1337
See Also QueryRecordHierarchy collection methods
QuerySelect class 1284
See Also QuerySelect methods
QuerySelect collection 1293
See Also QuerySelect collection methods
Record class 1378
See Also Record class methods
RemoteNode collection 1051
See Also RemoteNode collection methods
Request class 596
See Also Request class methods
Response class methods 605
See Also Response class methods
RolePermissionValue collection 1098
See Also RolePermissionValue collection methods
Row class 1408
See Also Row class methods
Rowset class 1423
See Also Rowset class methods
RuleDefn class 102
Search Language collection 1492
See Also Search Language collection methods
Search Schedule collection 1495
See Also Search Schedule collection methods
Search Start Options collection 1504
See Also Search Start Options collection methods
SearchField collection 1479
See Also SearchField collection methods
SearchIndex class 1483
SearchIndexes collection 1481
See Also SearchIndexes collection methods
SearchQuery class 1468
See Also SearchQuery class methods
SearchRecordField collection 1488
See Also SearchRecordField collection methods
SearchResult collection 1476
SelectedPagelet collection 1162
See Also SelectedPagelet collection methods
self-referencing 171
sending optimization status 926
Session class 423, 839
See Also Session class methods
SMTPSession 706
See Also SMTPSession class methods
SOAPDoc class 1546
See Also SOAPDoc class methods
SOAPDoc object Method section 1543
SQL class 1572
See Also SQL class methods
super-referencing 178
SyncServer class 1595
See Also SyncServer class methods
TabDefinition class 1113
TabDefinition collection 1120
See Also TabDefinition collection methods
Task class 1733
See Also Task class methods
TaskList class 1738
Tree class 1654
See Also Tree class methods
Tree Structure class 1690
See Also Tree Structure class methods
understanding 174
understanding API Repository 194
UserHomepage class 1155
UserTab collection 1158
See Also UserTab collection methods
using for collections 177
See Also PeopleCode collections
Util class 1740
variable class 122
Worklist class 896
WorklistEntry class 896
See Also WorklistEntry class methods
XmlDoc class 1751
See Also XmlDoc class methods
XmlNode class 1765
See Also XmlNode class methods
Methods property 207
MIMEList property
Search FS Options class 1503
Search HTTP Options class 1501
MIMEListType property

Copyright © 1988-2007, Oracle. All rights reserved.

Search FS Options class 1503
Search HTTP Options class 1501
mobile
classes 827
See Also Mobile classes
device 828
events 831
See Also mobile events
Mobile Agent 827
See Also Mobile Agent
objects 831
See Also mobile objects
pages 827
See Also mobile pages
Mobile Agent
detail views 829
list views 829
synchronization 1585, 1588
synchronization considerations 1605
understanding 827
mobile applications
system ID field name 519
timestamp field name 520
Mobile class
list of functions, methods, properties, returns, variables 1960
methods 840
See Also Mobile class methods
properties 854
understanding 835, 840
Mobile class methods
Create 840
Find 841
Get 842
GetName 842
GetParent 843
GetPeerDefaultAttributesByName 843
GetPropertyAttrsByName 844
GetPropertyByName 845
GetPropertyInfoByName 845
GetTopParent 846
IsNew 846
IsNewAndNeverSaved 847
IsSameAs 847
ReadBlobFromFile 848
Save 849
SetModified 850
SetPropertyByName 851
ValidateEnum 851
ValuesDifferFromLastSync 852
WasSaved 852
WriteBlobToFile 853
Mobile classes
hierarchy 831
list of functions, methods, properties, returns, variables 1960
ListViewAttrs 835
See Also ListViewAttrs class
Mobile 835
See Also Mobile class
PeerDefaultAttributes 835
See Also PeerDefaultAttributes class
PropertyAttrs 835
See Also PropertyAttrs class
PropertyInfo 835
See Also PropertyInfo class
PropertyInfoCollection 835
See Also PropertyInfoCollection class
understanding 827
Mobile collections
CI collection 835
See Also CI collection
Data collection 835
See Also Data collection
mobile device 828
mobile events
OnConflict 1589
OnGetDefinition 1606
OnGetProperty 1593
OnGetPropertyFilter 1593
OnSelect 1590
OnSetProperty 1593
OnValidate 1592
OnValidateSet 1605
understanding 831, 1589
using attachment events 1593
mobile objects
declaring/scope 838
hierarchy 831
understanding 831, 840
mobile pages
understanding 827
views 829
mobile PeopleCode
debugging 835, 836
functions 876
handling errors 837
PeopleCode classes 875
Session class methods 839
supported system variables 876

Copyright © 1988-2007, Oracle. All rights reserved.

supporting variables 838
tracing 837
models
analytic type definitions, deleting 143
MoveAsChild method
Leaf class 1622
Node class 1641
MoveAsChildByName method
Leaf class 1623
Node class 1641
MoveAsSib method
Leaf class 1623
Node class 1642
MoveAsSibByName method 1643
MoveAsSibByRange method 1624
MoveFirst method 299
MoveNext method 300
MoveToDoc method 301
MSFFactory
example 1742
MsgSchemaExists method 794
MultiChannel Framework functions
Broadcast 1725
MultiChannel Framework mail classes, *See* mail classes
MultiPart property
MCFBodyPart class 652
MCFOutboundEmail class 701
Multipurpose Internet Mail Extensions (MIME) 635

N

Name class 621
Name property
Agent class 1721
AnalyticModelDefn class 74
AnalyticTypeDefn class 148
AnalyticTypeModelDefn class 148
AnalyticTypeRecordDefn class 151
AttributeValue class 1088
Bindings 202
ClassInfo 207
ColumnNumber class 1309
CompIntfPropInfoCollection objects 456
Content Reference class 1076
Content Reference Links class 1106
Cookie class 612
cube class 112
CubeCollectionDefn class 96
CubeDefn class 85
DataSourceDefn class 1816
DimensionDefn class 75
ExplicitDimensionSet class 77
Field class 513
File class 553
Folder class 1064
FunctionCall class 118
GridColumn class 582
Level class 1633
Message class 778
MethodInfo 210
Namespaces 205
Node class 1048, 1651
NodeTemplate class 1056
OrganizerDefn class 101
Page class 994
Pagelet class 1149
PageletCategory class 1140
PermissionValue class 1093
Portal class 1054
PortalRegistry class 1042
PropertyInfo 212
PropertyInfo object 858
Query class 1280
Query Metadata class 1344
QueryCriteria class 1325
QueryDBRecord class 1353
QueryDBRecordField class 1360
QueryExpression class 1332
QueryPrompt class 1366
QueryRecord class 1301
QueryRecordHierarchy class 1341
Record class 1404
Rowset class 1450
SearchField class 1480
SearchIndex class 1484
TabDefinition class 1118
Tree class 1684
Tree Structure class 1699
UserFunctionDefn class 98
variable class 123
namespaces
API Repository 194
class 204
See Also Namespaces class
collection 203
See Also Namespaces collection
defining XML namespaces 1747
Namespaces property 201

Copyright © 1988-2007, Oracle. All rights reserved.

Namespaces class
list of properties, returns 2016
methods 205
properties 204
Namespaces class properties
Classes 204
Name 205
Namespaces collection
list of properties, returns 2016
methods 203
properties 203
Namespaces collection methods
Item 203
ItemByName 204
Namespaces collection properties 203
Namespaces methods 205
Namespaces property 201
NamespaceURI property 1798
naming
content references 1017
Java classes and packages 616
Negation property 1326
NetworkServices property 1534
NewWindowLink property 994
Next method
Array class 227
AssignedPagelet collection 1127
Attribute collection 1092
AvailableCategory collection 1130
AvailablePagelet collection 1133
CompIntfPropInfoCollection collection 454
Component Interface collection 425
Content Reference collection 1086
DynamicCategory collection 1136
Favorite collection 1168
Folder collection 1069
Link collection 1112
Node collection 1050
NodeTemplate collection 1059
Pagelet collection 1154
PageletCategory collection 1144
ParseResult collection 1474
PermissionValue collection 1096
Portal collection 1055
PortalRegistry collection 1045
PSMessages collection 1524
Query collection 1262
Query Metadata collection 1343
QueryCriteria collection 1314
QueryDBRecord collection 1351
QueryDBRecordField collection 1356
QueryExpression collection 1330
QueryField collection 1303
QueryList class 1336
QueryPrompt collection 1362
QueryRecord collection 1298
QueryRecordHierarchy collection 1339
QuerySelect collection 1295
RemoteNode collection 1052
RolePermissionValue collection 1100
Search Language collection 1494
Search Schedule collection 1497
Search Start Options collection 1506
SearchField collection 1480
SearchIndexes collection 1482
SearchRecordField collection 1490
SearchResult collection 1477
SelectedPagelet collection 1164
TabDefinition collection 1122
UserTab collection 1160
Next property 1631
NextSib property
Leaf class 1627
Node class 1651
NextSibling property 1799
NextStep method 477
NickName property 1721
Node class
list of methods, properties, returns 2077
list of properties, returns 2033
methods 1634
See Also Node class methods
properties 1046, 1646
See Also Node class properties
understanding 1046, 1634
Node class methods
Branch 1634
Cut 1635
Delete 1635
DeleteByName 1636
Expand 1636
InsertChildLeaf 1637
InsertChildNode 1637
InsertChildRecord 1638
InsertDynChildLeaf 1639
InsertSib 1640
InsertSibRecord 1640
MoveAsChild 1641

Copyright © 1988-2007, Oracle. All rights reserved.

MoveAsChildByName 1641
MoveAsSib 1642
MoveAsSibByName 1643
PasteChild 1643
PasteSib 1644
RefreshDescription 1644
Rename 1645
SwitchLevel 1645
Unbranch 1645
Node class properties
ActiveNode 1046
AllChildCount 1646
AllChildNodeCount 1646
AppsRelease 1046
ChildLeafCount 1647
ChildNodeCount 1647
ColImageName 1647
ContentURI 1047
DefaultPortalName 1047
Description 1047, 1647
DisplayLevelNumber 1647
ExpImageName 1647
FirstChildLeaf 1648
FirstChildNode 1648
HasChildLeaves 1648
HasChildNodes 1648
HasChildren 1648
HasNextSib 1649
HasPrevSib 1649
IsBranched 1649
IsChanged 1649
IsCut 1649
IsDefault 1047
IsDeleted 1649
IsInserted 1650
IsLocal 1047
IsRoot 1650
LastChildLeaf 1650
LastChildNode 1650
LevelNumber 1651
Name 1048, 1651
NextSib 1651
NodePassword 1048
NodeType 1048
Parent 1651
PortalURI 1048
PrevSib 1652
State 1652
ToolsRelease 1049
TreeBranchName 1652
TreeEffDt 1653
TreeName 1653
TreeSetId 1653
TreeUserKeyValue 1653
Type 1653
Node collection
list of methods, properties, returns 2034
methods 1049
See Also Node collection methods
properties 1050
understanding 1049
Node collection methods
First 1049
ItemByName 1049
Next 1050
node objects 1634
node templates, *See* NodeTemplate
NodeColImageName property 1685
NodeComponent property 1700
NodeCount property 1685
NodeDN property 815
NodeExists method 1668
NodeExpImageName property 1685
NodeField property 1700
NodeMenu property 1700
NodeMenuBar property 1700
NodeMenuItem property 1701
NodeMultiNavigate property 1701
NodeName property
Business Interlink class 314
XmlNode class 1799
NodeOnClipboard property 1685
NodePage property 1701
NodePassword property 1048
NodePath property 1799
NodeRecord property 1701
nodes
BodyNode property 1557
class 1046
See Also Node class
collection 1049
See Also Node collection
DestinationNode property 812
EnvelopeNode property 1557
FinalDestinationNode property 813
GetLocalNode method 1020
GetNodes method 1021
GetRemoteNodes method 1022
HeaderNode property 1558
HostNodeName property 1053

Copyright © 1988-2007, Oracle. All rights reserved.

MethodNode property 1559
naming content references 1017
NodeColImageName property 1685
NodeComponent property 1700
NodeCount property 1685
NodeDN property 815
NodeExpImageName property 1685
NodeField property 1700
NodeMenu property 1700
NodeMenuBar property 1700
NodeMenuItem property 1701
NodeMultiNavigate property 1701
NodeName property 314
NodeOnClipboard property 1685
NodePage property 1701
NodeRecord property 1701
NodeType property 315
NodeUserKeyField property 1702
NodeValue property 316
objects 1634
OrigNode property 816
Portal collection 1054
See Also Portal collection
RemoteNode collection 1051
See Also RemoteNode collection
RequestingNodeDescription property 817
RequestingNodeName property 817
SourceNode property 817
templates 1056
See Also NodeTemplate
understanding 999
using the URL property 1016
VisitedNodes property 819
XmlNode 1745
See Also XmlNode
NodeTemplate
class 1056
See Also NodeTemplate class
collection 1057
See Also NodeTemplate collection
NodeTemplates property 1043
NodeTemplate class
list of properties, returns 2036
properties 1056
understanding 1056
NodeTemplate class properties 1056
NodeTemplate collection
list of methods, properties, returns 2036
methods 1057
See Also NodeTemplate collection methods
properties 1059
understanding 1057
NodeTemplate collection methods
DeleteItem 1057
First 1057
InsertItem 1058
ItemByName 1058
Next 1059
NodeTemplate collection properties 1059
NodeTemplates property 1043
NodeType property
Incoming Business Interlink class 315
Node class 1048
XmlNode class 1799
NodeUserKeyField property 1702
NodeValue property
Incoming Business Interlink class 316
XmlNode class 1800
NonRepudiationID property 815
nonrowset-based messages
understanding 729
NonSSBs property 1534
notes cxvii
notification
PeopleCode 738
Notification class
importing 885
list of methods, properties, returns 1966
methods 886
properties 886
See Also Notification class properties
understanding 885
Notification class constructors
Notification 881
NotificationAddress 882
NotificationTemplate 883
Worklist 883
WorklistEntry 884
WSWorklistEntry 885
Notification class properties
ContentType property 886
dttmCreated property 886
EmailReplyTo property 887
FileNames property 887
FileTitles property 887
language_cd property 887
Message property 887
NotifyBCC property 888

Copyright © 1988-2007, Oracle. All rights reserved.

NotifyCC property 888
NotifyFrom property 888
NotifyGuid property 888
NotifyTo property 888
SourceComponent property 889
SourceMarket property 889
SourceMenu property 889
Subject property 889
Template property 889
Notification classes
constructors 881
See Also Notification class constructors
creating notification objects 881
examples 910
importing 880
list of constructors, methods, properties, returns 1965
Notification 885
See Also Notification class
NotificationAddress 890
See Also NotificationAddress class
NotificationTemplate 891
See Also NotificationTemplate class
scope 879
understanding 879
Worklist 895
See Also Worklist class
WorklistEntry 896
See Also WorklistEntry class
Notification constructor 881
notification objects, creating 881
notification PeopleCode 738
NotificationAddress class
list of properties, returns 1967
properties 890
understanding 890
NotificationAddress class properties 890
NotificationAddress constructor 882
notifications
classes 879
See Also Notification classes
declaring 880
NotificationTemplate class
list of methods, properties, returns 1967
methods 891
See Also NotificationTemplate class methods
properties 893
See Also NotificationTemplate class properties
understanding 891
NotificationTemplate class methods
GetAndExpandTemplate 891
SetupCompVarsAndRcpts 892
SetupGenericVars 893
NotificationTemplate class properties
ComponentId 894
Instruction 894
Language 894
Market 894
Priority 894
Responses 894
Subject 895
TemplateId 895
TemplateType 895
Text 895
NotificationTemplate constructor 883
Notify method 1598
NotifyBCC property 888
NotifyCC property 674, 888
NotifyFrom property 888
NotifyGuid property 888
NotifyTextMsgNum property 1218
NotifyTextMsgSet property 1219
NotifyTo property 674, 888
notifyToWindow method 1231
NRId property 778
Number property 1633
NumberOfAttachments property 815

O

object cleanup
Java programs 626
ObjectOwnerID property
DataSourceDefn class 1816
ReportDefn class 1831
TemplateDefn class 1837
objects, Java, *See* Java objects
objects, PeopleCode, *See* PeopleCode objects
OffsetReceived property 675
OffsetSent property 675
OLLineType property 374
OLType property 374
On Error method
WSWorklistEntry class 908
OnConflict event 1589
OnGetDefinition event 1606
OnGetProperty event 1593
OnGetPropertyFilter event 1593

Copyright © 1988-2007, Oracle. All rights reserved.

OnNotify method
WSWorklistEntry class 909
OnRequestSend method
AsyncFFSend class 244
OnSelect event 1590
OnSetProperty event 1593
OnValidate event 1592
OnValidateSet event 1605
Open method
AESection class 6
Crypt class 478
File class 534
PortalRegistry class 1037
Query class 1268
SQL class 1576
Tree class 1665
Tree Structure class 1692
OpenWholeTree method 1667
Operand1 property
comparison class 107
Operation class 121
Operand2 property
comparison class 107
Operation class 121
Operation class
creating 120
method 120
Operand1 property 121
Operand2 property 121
properties 121
Type property 121
OperationName property 778
OperationType property 815
OperationVersion property 778, 815
Operator property
description 1326
Using 1241
OPIs
developing PeopleCode to use 962
invoking 919
oprid property 905
Oprid property 891
OPT_CALL message
sending messages 932
showing messages for lights-out mode 931
understanding lights-out mode 921
OPT_CALL program 931
OptBase callback methods 962
OptBase class methods
callback 962
GetParmDate 963
GetParmDateArray 964
GetParmDateTime 964
GetParmDateTimeArray 964
GetParmInt 966
GetParmIntArray 966
GetParmNumber 965
GetParmNumberArray 965
GetParmString 967
GetParmStringArray 967
GetParmTime 968
GetParmTimeArray 968
Init 969
OptDeleteCallback 969
OptInsertCallback 970
OptPostUpdateCallback 970
OptPreUpdateCallback 971
OptRefreshCallback 972
SetOutputParmDate 972
SetOutputParmDateArray 973
SetOutputParmDateTime 973
SetOutputParmDateTimeArray 974
SetOutputParmInt 975
SetOutputParmIntArray 976
SetOutputParmNumber 974
SetOutputParmNumberArray 975
SetOutputParmString 976
SetOutputParmStringArray 977
SetOutputParmTime 977
SetOutputParmTimeArray 978
understanding 962
OptDeleteCallback method 962, 969
OptEngine class
methods 942
See Also OptEngine class methods
properties 960
See Also OptEngine class properties
OptEngine class methods
CheckOptEngineStatus 942
FillRowset 943
GetDate 945
GetDateArray 945
GetDateTime 946
GetDateTimeArray 947
GetNumber 948
GetNumberArray 949
GetString 950
GetStringArray 950
GetTime 952

Copyright © 1988-2007, Oracle. All rights reserved.

GetTimeArray 952
GetTraceLevel 953
RunAsynch 954
See Also RunAsynch method
RunSynch 956
See Also RunSynch method
SetTraceLevel 957
ShutDown 959
using optimization PeopleCode in Application Engine programs 916
using optimization PeopleCode on application servers 915
OptEngine class properties
DetailedStatus 961
DetailMsgs 960
OptEngine objects 918
optimization
PeopleCode 915
See Also optimization PeopleCode
Optimization classes
list of functions, methods, properties, returns 1971
optimization engines
creating 923
loading 927
loading analytic instances 917
OptEngine class 942
See Also OptEngine class
running transactions in asynchronous mode 954
running transactions in synchronous mode 956
shutting down 919, 926, 959
status messages, logging 981
status, checking 942
status, viewing 923
using the CreateOptEngine function 933
using the GetOptEngine function 937
optimization models
activating 979
active 985
changing variable types 988
deactivating 980
overriding bounding values 986
restoring bounding values 985
retrieving solution details 984
retrieving solutions 982
solving 989
optimization PeopleCode
analytic instances, creating 916
analytic instances, deleting 919
analytic instances, loading 917
invoking OPIs 919
OptBase application class 962
See Also OptBase class methods
programming for database updates 920
request messages 922
See Also request messages
response messages 926
See Also response messages
running optimization transactions 918
shutting down optimization engines 919
understanding the functions 933
using in Application Engine 917
using in Application Engine programs 916
using lights-out mode 921
using on application servers 915, 917
optimization transactions
processing parameters 931
running 918, 923, 929
running in asynchronous mode 954
running in synchronous mode 956
running on application servers 918
running on the Application Engine 919
OptInsertCallback method 962, 970
OptInterface class methods
ActivateModel 979
ActivateObjective 979
DeactivateModel 980
DumpMsgToLog 981
FindRowNum 981
GetSolution 982
GetSolutionDetail 984
IsModelActive 985
RestoreBounds 985
SetVariableBounds 986
SetVariableType 988
Solve 989
understanding 978
OPTIONS table 521
OptPostUpdateCallback method 963, 970
OptPreUpdateCallback method 963, 971
OptRefreshCallback method 963, 972
Oracle
sorting records 1428
using hints in expressions 1333
OrderByDirection property 1309

Copyright © 1988-2007, Oracle. All rights reserved.

OrderByNumber property 1309
organizer
accessing 64
accessing names 65
adding to analytic model 50
deleting 58
renaming 70
OrganizerDefn class
AttachPart method 99
Comments property 101
DetachPart method 99
GetPartNames method 100
methods 99
Name property 101
parts
attaching 99
detaching 99
getting names 100
using 100
properties 101
UsesPart method 100
OriginalTime property 1738
OriginalValue property 513
originatorid property 905
OrigNode property 816
OrigProcess property 816
OrigTimeStamp property 816
OrigUser property 816
outbound email 631
OutDest property 1219
OutDestFormat property
PostReport class 1188
ProcessRequest class 1220
OutDestination property 1831
OutDestinationType property 1831
OutDestType property 1220
outer joins query 1375
Output method 485
OutputEditable property 1832
OutputField property 1333
OutputUNICODE property 1534
overflow
email errors 642
OverflowedTaskList property 1732
OverFlowTime property 1738
OverflowedTaskList property 1732
OverFlowTime property 1738
overlay, chart 372, 374
OwnerId property
Content Reference class 1077
Content Reference Links class 1106
Folder class 1064
Pagelet class 1149
PageletCategory class 1140
PortalRegistry class 1043
TabDefinition class 1118
OwnerID property
AnalyticTypeDefn class 148

P

packages
application 153
See Also application packages
naming in Java 616
Page class
list of functions, methods, properties, returns 1974
properties 992
See Also Page class properties
shortcut considerations 991
understanding 991
Page class properties
CopyURLLink 992
CustomizePageLink 993
DisplayOnly 993
HelpLink 994
Name 994
NewWindowLink 994
Visible 995
page objects
declaring/scope 992
understanding 991
PageIndex property 1897
Pagelet class
list of methods, properties, returns 2052
methods 1145
properties 1146
See Also Pagelet class properties
understanding 1145
Pagelet class properties
Attributes 1146
Author 1146
AuthorAccess 1146
Authorized 1147
CascadedPermissions 1147
ContentProvider 1147
CreationDate 1147
DefaultColumn 1147
Description 1148
EditPageContentProvider 1148

Copyright © 1988-2007, Oracle. All rights reserved.

EditPageQueryString 1148
HelpID 1148
IsHideMinimize 1148
Label 1148
Name 1149
OwnerId 1149
ParentName 1149
Permissions 1149
Product 1149
PublicAccess 1150
QualifiedURL 1150
SequenceNumber 1150
URL 1150
URLType 1151
Pagelet collection
list of methods, properties, returns 2053
methods 1151
See Also Pagelet collection methods
properties 1154
understanding 1151
Pagelet collection methods
DeleteItem 1151
First 1152
InsertItem 1152
ItemByName 1153
Next 1154
pagelet objects
AssignedPagelet objects 1123
AvailablePagelet 1130
PageletCategory 1137
understanding 1145
PageletCategories property 1043
PageletCategory
class 1137
See Also PageletCategory class
collection 1142
See Also PageletCategory collection
PageletCategory class
list of methods, properties, returns 2050
methods 1138
properties 1138
See Also PageletCategory class properties
understanding 1137
PageletCategory class properties
Attributes 1138
Author 1139
AuthorAccess 1139
Authorized 1139
CascadedPermissions 1139
CreationDate 1140
Description 1140
Label 1140
Name 1140
OwnerId 1140
Pagelets 1141
Permissions 1141
Product 1141
PublicAccess 1141
SequenceNumber 1142
PageletCategory collection 1142
list of methods, properties, returns 2051
methods 1142
See Also PageletCategory collection methods
properties 1145
understanding 1142
PageletCategory collection methods
DeleteItem 1142
First 1143
InsertItem 1143
ItemByName 1144
Next 1144
PageletLabel property 1132
PageletName property
AssignedPagelet class 1124
AvailablePagelet class 1132
SelectedPagelet class 1161
pagelets 1145
See Also pagelet objects
AvailablePagelet collection 1132
See Also AvailablePagelet collection
class 1145
See Also Pagelet class
collection 1151
See Also Pagelet collection
PageletCategory class 1137
See Also PageletCategory class
PageletCategory collection 1142
See Also PageletCategory collection
PageletName property 1124
SelectedPagelet class PageletName property 1161
SelectedPagelet collection 1162
See Also SelectedPagelet collection
SelectedPagelet objects 1160
using FindPgltByName method 1033
Pagelets property 1141
PageNumber class
constructor 1866

Copyright © 1988-2007, Oracle. All rights reserved.

properties 1883
PageNumber class properties
BackgroundFile 1883
FontName 1883
FontSize 1884
PositionX 1884
PositionY 1884
StartFromPageNum 1884
StartNum 1885
PageNumber property 1889
pages
Page class 991
See Also Page class
page objects 991
See Also page objects
parameters
passing in application class methods 162
passing with object data types 163
using the out specification 163
Parent property
Leaf class 1628
Node class 1651
ParentFlag property 1341
ParentLevel property 1686
ParentName property
Content Reference class 1077
Content Reference Links class 1107
Folder class 1064
Pagelet class 1149
Tree class 1686
ParentNode property 1800
ParentRecord property 514
ParentRow property
Record class 1404
Rowset class 1450
ParentRowset property
Row class 1416
Rowset class 1451
ParentSelectNum property 1291
ParentTransactionId property 778
ParmCount property 1559
Parse method 1469
ParseResult
class 1475
See Also ParseResult class
collection 1473
See Also ParseResult collection
objects 1475
ParseResult class
list of properties, returns 2023
properties 1475
understanding 1475
ParseResult class properties 1475
ParseResult collection
list of methods, properties, returns 2023
methods 1473
properties 1474
understanding 1473
ParseResult collection methods 1473
ParseResult collection properties 1474
ParseResult objects 1475
ParseXmlFromURL method 1762
ParseXmlString method 1763
ParseXMLString method 312
PartCount property 779
parts
attaching 99
detaching 99
getting names 100
using 100
Password property 667
PasteChild method 1643
PasteSib method
Leaf class 1625
Node class 1644
Path property
Content Reference class 1077
Content Reference Links class 1107
Cookie class 613
Folder class 1065
PathInfo property
ConnectorInfo collection 826
Request class 602
paths, relative 534
PCExtFcnCalls property 1534
PCFcnReturnValues property 1534
PCFetchedValues property 1535
PCIntFcnCalls property 1535
PCListProgram property 1535
PCParameterValues property 1535
PCProgramStatements property 1535
PCStack property 1535
PCStartOfPrograms property 1536
PCTraceProgram property 1536
PCVariableAssignments property 1536
PDFFont property 1280
PDFMapTool class
constructor 1866
PDFMapTool methods

Copyright © 1988-2007, Oracle. All rights reserved.

EnableMap 1885
extractMap 1886
processForm 1887
PDFMapTool property
Locale 1887
PDFMerger class
constructor 1866
method 1888
properties 1889
PDFMerger method
MergePDFs 1888
PDFMerger properties
ConfProp 1889
Locale 1889
PageNumber 1889
Watermark 1890
PeerDefaultAttributes class
list of methods, properties, returns 1965
methods 873
properties 874
understanding 835, 873
PeopleBooks
ordering cxiv
PeopleCode
accessing PeopleCode classes using Java 621
See Also PeopleCode objects
accessing the API Repository 195
calling via Java 620
functions 917
See Also functions
generating for file layout automatically 527
mapping Java types/classes 624
mobile 831
See Also mobile PeopleCode
notification 738
optimization 915
See Also optimization PeopleCode
quick reference for syntax 2096
typographic conventions for this guide cxxi
PeopleCode classes
accessing via Java 621
AESection 1
See Also AESection class
Agent class 1717
See Also Agent class
AgentPhysQueueProps class 1722
See Also AgentPhysQueueProps class
AgentPhysQueueTasks class 1723
See Also AgentPhysQueueTasks class
API Repository classes 201
See Also API Repository classes
Array 215, 223
See Also Array class
AssignedPagelet 1123
See Also AssignedPagelet class
AttributeValue 1087
See Also AttributeValue class
AvailableCategory 1128
See Also AvailableCategory class
AvailablePagelet 1130
See Also AvailablePagelet class
Bindings 202
See Also Bindings class
BPEL classes 241
See Also BPEL classes
Broadcast class 1725
See Also Broadcast class
Business Interlink 253
See Also Business Interlink class
Chart 319
See Also Chart class
ClassInfo 207
See Also ClassInfo class
Component Interface 426
See Also Component Interface class
Component Interface classes 411
See Also Component Interface classes
Content Reference 1070
See Also Content Reference class
Content Reference Links 1101
See Also Content Reference Links class
Cookie 595
See Also Cookie class
Crypt 475
See Also Crypt class
Data Item 450
See Also Data Item
Exception 180
See Also Exception class
Favorite 1165
See Also Favorite class
Field 491
See Also Field class
File 523
See Also File class
Folder 1059

Copyright © 1988-2007, Oracle. All rights reserved.

See Also Folder class
Func Java 620
Grid 575
See Also Grid class
GridColumn 580
See Also GridColumn class
header comments 183
IBInfo 802
See Also IBInfo class
Incoming Business Interlink 303
See Also Incoming Business Interlink class
IntBroker 783
See Also IntBroker class
iScript classes 585
See Also iScript classes
Java 615
See Also Java class
Leaf 1619
See Also Leaf class
Level 1632
See Also Level class
Link 1112
See Also Link class
LogicalQueue class 1727
See Also LogicalQueue class
mail classes 629
See Also mail classes
MCFFactory 1707
See Also MCFFactory class
MCFFactory class 1728
See Also MCFFactory class
MCFIMInfo 723
See Also MCFIMInfo class
Message 729
See Also Message classes
MethodInfo 209
See Also MethodInfo class
Mobile 827
See Also Mobile classes
Name Java 621
Namespaces 204
See Also Namespaces class
Node 1046, 1634
See Also Node class
NodeTemplate 1056
See Also NodeTemplate class
Notification classes 879
See Also Notification classes
Page 991
See Also Page class
Pagelet 1145
See Also Pagelet class
PageletCategory 1137
See Also PageletCategory class
ParseResult 1475
See Also ParseResult class
PermissionValue 1002, 1092
See Also PermissionValue class; PermissionValue objects
PhysicalQueue class 1728
See Also PhysicalQueue class
Portal 1052
See Also Portal class
PortalRegistry 997, 1000, 1022
See Also PortalRegistry class; PortalRegistry classes
PostReport 1185
See Also PostReport class
PrcsApi class 1228
ProcessRequest 1193
See Also ProcessRequest class
PropertyInfo 212
See Also PropertyInfo class
PSMessages 1525
See Also PSMessages class
Query classes 1233
See Also Query classes
Query Metadata 1344
See Also Query Metadata class
QueryCriteria 1314
See Also QueryCriteria class
QueryDBRecord 1352
See Also QueryDBRecord class
QueryDBRecordField 1357
See Also QueryDBRecordField class
QueryExpression 1331
See Also QueryExpression class
QueryList 1334
See Also QueryList class
QueryListValue 1337
See Also QueryListValue class
QueryPrompt 1363
See Also QueryPrompt class
QueryRecord 1299
See Also QueryRecord class
QueryRecordHierarchy 1340
See Also QueryRecordHierarchy class
QuerySecurityProfile 1346
See Also QuerySecurityProfile class

Copyright © 1988-2007, Oracle. All rights reserved.

QuerySelect 1283
See Also QuerySelect class
QueryStatistics 1345
See Also QueryStatistics class
quick reference 1901
Record 1377, 1564
See Also Record class
Regional Settings 1528
See Also Regional Settings class
Repository 201
See Also Repository class
Request 594
See Also Request class
Response 595
See Also Response class
RolePermissionValue 1002
See Also RolePermissionValue objects
Row 1407
See Also Row class
Rowset 1421
See Also Rowset class
RowsetCache class 1453
See Also RowsetCache class
Search classes 1459
See Also Search classes
Search FS Options 1502
See Also Search FS Options class
Search HTTP Options 1499
See Also Search HTTP Options class
Search Language 1495
See Also Search Language class
Search Schedule 1498
See Also Search Schedule class
Search Start Options 1506
SearchField 1480
See Also SearchField class
SearchIndex 1482
See Also SearchIndex class
SearchQuery 1468
See Also SearchQuery class
SearchRecord Options 1485
See Also SearchRecord Options class
SearchRecordField 1491
See Also SearchRecordField class
SearchResult 1477
See Also SearchResult class
SelectedPagelet 1160
See Also SelectedPagelet class
Session 423, 1511
See Also Session class
SOAPDoc 1539
See Also SOAPDoc class
SQL 1563
See Also SQL class
SyncServer 1585
See Also SyncServer class
SysCon Java 620
SysVar Java 620
TabDefinition 1113
See Also TabDefinition class
Task class 1733
See Also Task class
TaskList class 1738
See Also TaskList class
Trace Setting 1532
See Also Trace Setting class properties
Tree 1653
See Also Tree class
Tree Structure 1690
See Also Tree Structure class
universal queue classes 1707
See Also universal queue classes
UserHomepage 1154
See Also UserHomepage class
UserTab 1156
See Also UserTab class
using security with PortalRegistry 1001
Util class 1740
See Also Util class
XML Publisher classes 1801
See Also XML Publisher classes
XmlDoc 1745
See Also XmlDoc class
XmlNode 1765
See Also XmlNode class
PeopleCode collections
accessing PortalRegistry objects 1001
API Repository collections 201
See Also API Repository collections
AssignedPagelet 1124
See Also AssignedPagelet collection
Attribute 1089
See Also Attribute collection
AvailableCategory 1128
See Also AvailableCategory collection
AvailablePagelet 1132
See Also AvailablePagelet collection
Bindings 201
See Also Bindings collection
Bindings collection 201

Copyright © 1988-2007, Oracle. All rights reserved.

See Also Bindings collection
Branch 1618
See Also Branch collection
ClassInfo 205
See Also ClassInfo collection
CompIntfPropInfoCollection 451
See Also CompIntfPropInfoCollection collection
Component Interface 424
See Also Component Interface collection
ConnectorInfo 819
See Also ConnectorInfo collection
Content Reference 1084
See Also Content Reference collection
Data 419, 443, 861
See Also Data collection
declaring 1515
DynamicCategory 1134
See Also DynamicCategory collection
Favorite 1166
See Also Favorite collection
Folder 1066
See Also Folder collection
Level 1629
See Also Level collection
Links 1111
See Also Links collection
MethodInfo 208
See Also MethodInfo collection
Namespaces 203
See Also Namespaces collection
Node 1049
See Also Node collection
NodeTemplate 1057
See Also NodeTemplate collection
Pagelet 1151
See Also Pagelet collection
PageletCategory 1142
See Also PageletCategory collection
ParseResult 1473
See Also ParseResult collection
PermissionValue 1002, 1094
See Also PermissionValue collection
Portal 1054
See Also Portal collection
PortalRegistry 1044
See Also PortalRegistry collection
PortalRegistry collections 1044
See Also PortalRegistry collections
PropertyInfo 210
See Also PropertyInfo collection
PSMessages 1522
See Also PSMessages collection
Query collections 1260
See Also Query collections
Query Metadata 1341
See Also Query Metadata collection
QueryCriteria 1312
See Also QueryCriteria collection
QueryDBRecord 1349
See Also QueryDBRecord collection
QueryDBRecordField 1354
See Also QueryDBRecordField collection
QueryExpression 1328
See Also QueryExpression collection
QueryField 1301
See Also QueryField collection
QueryPrompt 1361
See Also QueryPrompt collection
QueryRecord 1296
See Also QueryRecord collection
QueryRecordHierarchy 1337
See Also QueryRecordHierarchy collection
QuerySelect 1293
See Also QuerySelect collection
RemoteNode 1051
See Also RemoteNode collection
RolePermissionValue 1002, 1097
See Also RolePermissionValue collection
Search collections 1460
See Also Search collections
Search Language 1492
See Also Search Language collection
Search Schedule 1495
See Also Search Schedule collection
Search Start Options 1504
See Also Search Start Options collection
SearchField collection 1479
See Also SearchField collection
SearchIndexes 1481
See Also SearchIndexes collection
SearchRecordField 1488
See Also SearchRecordField collection
SearchResult 1460, 1476
See Also SearchResult collection

Copyright © 1988-2007, Oracle. All rights reserved.

SelectedPagelet 1162
See Also SelectedPagelet collection
TabDefinition 1119
See Also TabDefinition collection
Tree collections 1612
See Also Tree collections
UserTab 1158
See Also UserTab collection
using methods/properties 177
PeopleCode events
GUI/online processing 422
mobile 831
See Also mobile events
processing Search dialog boxes 422
using the WorklistEntry class Save method 899
using the WorklistEntry class SaveWithCustomData method 900
PeopleCode functions
accessing messages 837
application class 186
See Also application class functions
binding meta-SQL 1565
calling from Java programs 625
Create 166
CreateXmlDoc 1746
declaring external 168
deprecated 2090
DistributeDataByRules 1591
DoSave 877
function libraries 587
Generate 591
GUI/online processing 422
Import 188
instantiating record objects 1377
instantiating rowset objects 1421
iScripts 585
See Also iScripts
Java 615
See Also Java functions
mapping deprecated/invalid terms to new terms 2088
mapping old names to new names 2088
mapping to methods and properties 2085
Method 190
methods 194
See Also methods
mobile PeopleCode 876
ScheduleProcess 1193
structuring via application classes 154
using flattening and promotion 221
using transfer functions 483
PeopleCode interfaces 164
PeopleCode objects
accessing 1001
AESection 2
See Also AESection objects
ApiObject 1514
arrays 215
See Also array class
AssignedPagelet 1123
Attribute collection 1089
AttributeValue 1087
AvailableCategory 1128
AvailablePagelet 1130
BPEL 242
Business Interlink 253
See Also Business Interlink objects
chart 343
collections 177
See Also PeopleCode collections
CompIntfPropInfoCollection 451
Component Interface 411
See Also Component Interface objects
ConnectorInfo collection 819
content reference 1070
See Also content reference objects
content reference links 1101
creating mail objects 631
Data collection 861
declaring cookie objects 594
declaring iScript objects 594
declaring Request objects 594
declaring Response objects 594
favorite 1165
field 491
folder 1059
Folder collection 1066
grid 576
See Also grid objects
IBInfo 802
instantiating API objects 1518
IntBroker 783
leaf 1619
Level 1632
links 1112
ListViewAttrs class 866
MCFFactory 1707
mobile 831

Copyright © 1988-2007, Oracle. All rights reserved.

See Also mobile objects
node 1634
nodes 1046
NodeTemplate 1056
notification 881
page 991
See Also page objects
PageletCategory 1137
PageletCategory collection 1142
pagelets 1145
See Also pagelet objects
ParseResult 1475
PermissionValue 1092
See Also PermissionValue class
portal 1000
PostReport 1186
PrcsApi 1228
ProcessRequest 1194
PropertyAttrs class 869
PropertyInfo class 856
PropertyInfoCollection 854
PSMessages 1512
queries 1233
See Also query objects
query metadata 1344
QueryCriteria 1240, 1314
See Also QueryCriteria objects
QueryDBRecord 1352
QueryDBRecordField 1357
QueryExpression 1331
QueryField 1304
QueryRecord 1299
QueryRecordHierarchy 1337
QuerySelect 1283
records 1377
See Also record objects
regional settings 1513
request 593
See Also request objects
response 593
See Also response objects
rows 1407
See Also row objects
rowsets 1421
See Also rowset objects
search 1465
Search FS Options 1502
Search HTTP Options 1499
search indexes 1459
Search Language 1495
search queries 1468
search record options 1485
search schedule 1498
Search Start 1506
SearchField 1480
SearchIndex 1459
SearchRecordField 1491
SelectedPagelet 1160
Session 1511
See Also session objects
SOAPDoc 1542
See Also SOAPDoc objects
SQL 1563
See Also SQL objects
SyncServer 1587
See Also SyncServer objects
TabDefinition 1113
TemplateObject property 1081
TemplateObject property (Content Reference Links class) 1109
See Also TemplateObject property
TraceSettings 1514
tree structure 1690
trees 1609
See Also tree objects
universal queue 1710
UserHomepage 1154
UserTab 1156
using with Java 621
XmlDoc 1745
See Also XmlDoc objects
XMLDoc 1545
See Also XMLDoc objects
XmlNode 1745
See Also XmlNode objects
PeopleCode programs
cleaning up in Java 626
PeopleCode, typographical conventions cxvi
PeopleSoft API Respository, *See* API Repository
PeopleSoft Application Engine, *See* Application Engine
PeopleSoft Enterprise Portal cxxi
PeopleSoft Mobile Agent 827
See Also Mobile Agent
PeopleSoft MultiChannel Framework mail classes, *See* mail classes
PeopleSoft Query Manager 1233
PeopleSoft Tree Manager 1609

Copyright © 1988-2007, Oracle. All rights reserved.

PeopleTools, licensing cxxi
performance issues
accessing a Component Interface structure 451
BulkExecute method 268
BulkMode property 1563
IsPlainImage property 357
PerformanceMethod property 1686
PerformanceSelector property 1687
PerformanceSelectorOption property 1687
replicating business rules 184
RequestedFields property 1472
reusing a cursor 1567
RowsetCache class 1453
setting trace levels 957
using closed trees 1610
using Get/Set keywords/methods 176
using OnValidate event 1592
using OnValidateSet event 1605
using synchronization server events 1588
Visible property 582
PerformanceMethod property 1686
PerformanceSelector property 1687
PerformanceSelectorOption property 1687
permission lists
PermissionListDelete method 1037
PermissionListSaveAs method 1038
using PermissionValue 1002
PermissionListDelete method 1037
PermissionListSaveAs method 1038
permissions
CascadedPermissions folder property 1062
CascadedPermissions PortalRegistry property 1002
CascadedPermissions property 1074, 1104, 1139, 1147
CascadedRolePermissions folder property 1062
CascadedRolePermissions property 1074, 1104
GrantPermissionForComponent method 1035
GrantPermissionForScript method 1036
Permissions folder class property 1065
Permissions PortalRegistry property 1002
Permissions property 1077, 1107, 1141, 1149
PermissionValue 1002
See Also PermissionValue
RevokePermissionForComponent method 1038
RevokePermissionForScript method 1039
RolePermissions Folder class property 1065
RolePermissions property 1079, 1108
setting via PermissionValue objects 1177
Permissions property
Content Reference class 1077
Content Reference Links class 1107
Folder class 1065
Pagelet class 1149
PageletCategory class 1141
understanding 1002
PermissionValue
class 1092
See Also PermissionValue class
collection 1094
See Also PermissionValue collection
objects 1002
See Also PermissionValue objects
using 1002
PermissionValue class
list of, properties, returns 2042
properties 1093
See Also PermissionValue class properties
understanding 1092
PermissionValue class properties
Cascade 1093
Name 1093
PermType 1093
PermissionValue collection
list of methods, properties, returns 2042
methods 1094
See Also PermissionValue collection methods
properties 1097
understanding 1094
PermissionValue collection methods
DeleteItem 1094
First 1095

Copyright © 1988-2007, Oracle. All rights reserved.

InsertItem 1095
ItemByName 1096
Next 1096
PermissionValue objects
setting permissions 1177
PermType property 1093
persist aggregate
getting 90
setting 94
physical queues
constructors 1713
PhysicalQueue class 1728
See Also PhysicalQueue class
PhysicalQueueID property 1721, 1722, 1725, 1727, 1738, 1739
TotalPhysicalQueues property 1722
understanding 1728
viewing properties 1722
viewing tasks 1723
PhysicalQueue class
list of methods, properties, returns 1993
methods 1729
properties 1730
See Also PhysicalQueue class properties
understanding 1728
PhysicalQueue class methods 1729
PhysicalQueue class properties
AcceptedTaskList 1730
Agent 1730
AssignedTaskList 1730
BrowserURL 1730
EnqueuedTaskList 1731
EscalatedTaskList 1731
InternalURL 1731
IsActive 1731
LogicalQueueID 1732
OverflowedTaskList 1732
PhysicalQueueID 1732
RENURLID 1733
TotalAgents 1733
PhysicalQueue constructor 1713
PhysicalQueueID property 1721
AgentPhysQueueTasks class 1725
LogicalQueue class 1727
PhysicalQueue class 1732
PhysicalQueueID class 1722
Task class 1738
TaskList class 1739
pluggable cryptography 475
Pop method 228
Port property 710
Portal class
list of methods, properties, returns 2035
methods 1053
properties 1053
understanding 1052
Portal class methods 1053
Portal class properties 1053
Portal collection
list of methods, properties, returns 2035
methods 1054
See Also Portal collection methods
properties 1055
understanding 1054
Portal collection methods
First 1054
ItemByName 1054
Next 1055
Portal collection properties 1055
portal component content references 1014
portal objects 1000
portal registry
classes 997
See Also PortalRegistry classes
FindPortalRegistries method 1020
GetPortalRegistry method 1021
PortalRegistry classes 997
See Also PortalRegistry
understanding 997
using security for objects 1001
PortalFolder property 1221
PortalRegistry
classes 997, 1000
See Also PortalRegistry classes
collection 1044
See Also PortalRegistry collection
PortalRegistry class
changing properties 1169
list of methods, properties, returns 2030
methods 1022
See Also PortalRegistry class methods
properties 1041
See Also PortalRegistry class properties
understanding 1022
PortalRegistry class methods
BuildSearchIndex 1467
Close 1023
CopyObject 1023

Copyright © 1988-2007, Oracle. All rights reserved.

Create 1024
CreateContentRefLink 1025
CreateRemote 1026
Delete 1027
DeleteHomepage 1028
FindCRefByName 1028
FindCRefByURL 1029
FindCRefForURL 1030
FindCRefLinkByName 1031
FindFolderByName 1032
FindPgltByName 1033
GetAbsoluteContentURL 1033
GetDefaultHPTabOID 1034
GetQualifiedURL 1034
GetSearchQuery 1467
GrantPermissionForComponent 1035
GrantPermissionForScript 1036
Open 1037
PermissionListDelete 1037
PermissionListSaveAs 1038
RevokePermissionForComponent 1038
RevokePermissionForScript 1039
Save 1040
PortalRegistry class properties
DefaultTemplate 1041
Description 1041
Favorites 1042
FolderNavObject 1042
Homepage 1042
IsFolderNavigation 1042
Name 1042
NodeTemplates 1043
OwnerId 1043
PageletCategories 1043
Portals 1043
RootFolder 1043
TabDefinitions 1044
TemplateObject 1044
PortalRegistry classes
AssignedPagelet 1123
See Also AssignedPagelet class
AttributeValue 1087
See Also AttributeValue class
AvailableCategory 1128
See Also AvailableCategory class
AvailablePagelet 1130
See Also AvailablePagelet class
Content Reference 1070
See Also Content Reference class
Content Reference Links 1101
See Also Content Reference Links class
examples 1169
Favorite 1165
See Also Favorite class
Folder 1059
See Also Folder class
handling errors 1004
Link 1112
See Also Link class
list of methods, properties, returns 2030
Node 1046
See Also Node class
NodeTemplate 1056
See Also NodeTemplate class
Pagelet 1145
See Also Pagelet class
PageletCategory 1137
See Also PageletCategory class
PermissionValue 1092
See Also PermissionValue class
Portal class 1052
See Also Portal class
PortalRegistry 1022
See Also PortalRegistry class
saving content 1018
SelectedPagelet 1160
See Also SelectedPagelet class
Session class 1019
See Also Session class
TabDefinition 1113
See Also TabDefinition class
understanding 997, 1000
UserHomepage 1154
See Also UserHomepage class
UserTab 1156
See Also UserTab class
using 1000
using ValidFrom/ValidTo 1004
viewing the class hierarchy 1008
PortalRegistry collection
list of methods, properties, returns 2033
methods 1044
See Also PortalRegistry collection methods
properties 1045
understanding 1044
PortalRegistry collection methods
First 1044
Item 1045

Copyright © 1988-2007, Oracle. All rights reserved.

Next 1045
PortalRegistry collections
AssignedPagelet 1124
See Also AssignedPagelet collection
Attribute 1089
See Also Attribute collection
AvailableCategory 1128
See Also AvailableCategory collection
AvailablePagelet 1132
See Also AvailablePagelet collection
Content Reference 1084
See Also Content Reference collection
DynamicCategory 1134
See Also DynamicCategory collection
Favorite 1166
See Also Favorite collection
Folder 1066
See Also Folder collection
Links 1111
See Also Links collection
Node 1049
See Also Node collection
NodeTemplate collection 1057
See Also NodeTemplate collection
Pagelet 1151
See Also Pagelet collection
PageletCategory 1142
See Also PageletCategory collection
PermissionValue 1094
See Also PermissionValue collection
Portal collection 1054
See Also Portal collection
PortalRegistry 1044
See Also PortalRegistry collection
RemoteNode 1051
See Also RemoteNode collection
RolePermissionValue 1097
See Also RolePermissionValue collection
SelectedPagelet 1162
See Also SelectedPagelet collection
TabDefinition 1119
See Also TabDefinition collection
UserTab 1158
See Also UserTab collection
PortalRegistry objects
declaring 1018
deleting 1017
life-cycle 1005
referencing via C/C++ 1181
referencing via Visual Basic 1180
scope 1019
portals
AbsolutePortalURL property 1072, 1103
class 1052
See Also Portal class
DefaultPortalName property 1047
Enterprise Portal cxxi
LinksPortalName property 1113
objects 1000
portal component 1014
PortalFolder property 1221
PortalURI property 1048
registry 997
See Also portal registry
templates 1013
Portals property 1043
PortalURI property 1048
PositionX property 1884
PositionY property 1884
Post Office Protocol 3 (POP3) 635
PostReport
class 1185
See Also PostReport class
objects 1186
PostReport class
determining distribution lists 1185
examples 1191
list of functions, methods, properties, returns 1975
methods 1186
See Also PostReport class methods
properties 1188
See Also PostReport class properties
understanding 1185
PostReport class methods
AddDistributionOption 1186
Put 1187
PostReport class properties
ExpirationDate 1188
OutDestFormat 1188
ProcessInstance 1189
ProcessName 1189
ProcessType 1189
ReportDescr 1190
ReportFolder 1190
ReportId 1190
ServerName 1191
SourceReportPath 1191

Copyright © 1988-2007, Oracle. All rights reserved.

PostReport objects 1186
PrcsApi
  class 1228
    See Also PrcsApi class
  objects 1228
PrcsApi class
  constructors 1230
  importing 1229
  list of constructors, methods, returns 1979
  methods 1230
    See Also PrcsApi class methods
  understanding 1228
PrcsApi class constructors 1230
PrcsApi class methods
  getAllFileNames 1230
  notifyToWindow 1231
PrcsApi constructor 1230
PrcsApi objects 1228, 1229
Prefix property 1800
prerequisites cxiii
PreviousSibling property 1800
prevoprid property 905
PrevSib property
  Leaf class 1628
  Node class 1652
printed documentation cxiv
PrintJobHTMLRpt method 1204
PrintJobRqstRpt method 1205
PrintOutput method 1821
PrintSchdlHTMLRpt method 1208
Priority property 894
  MCFEmail class 654
  MCFOutboundEmail class 701
private
  declaring instance variables 160, 178
    See Also instance variables
  declaring methods 160, 185
process instances 1189
ProcessedCount property 1472
ProcessEtextReport method 1822
processForm method 1887
ProcessInstance property
  PostReport class 1189
  ProcessRequest class 1221
ProcessInstanceID property 1852
ProcessName property
  PostReport class 1189
  ProcessRequest class 1221
ProcessReport method 1823
ProcessRequest
  class 1193
    See Also ProcessRequest class
  objects 1194
ProcessRequest class
  examples 1226
  format values 1198
  list of functions, methods, properties, returns 1976
  methods 1203
    See Also ProcessRequest class methods
  output type values 1198
  properties 1216
    See Also ProcessRequest class properties
  scheduling a single process 1226
  scheduling jobs with item changes 1227
  scheduling jobs without item changes 1227
  understanding 1193
ProcessRequest class methods
  AddDistributionOption 1203
  PrintJobHTMLRpt 1204
  PrintJobRqstRpt 1205
  PrintSchdlHTMLRpt 1208
  RunJobSetNow 1211
  Schedule 1211
  SetEmailOption 1212
  SetItemFolder 1213
  SetOutputOption 1214
  UpdateRunStatus 1215
ProcessRequest class properties
  EmailAttachLog 1216
  EmailSubject 1216
  EmailText 1216
  EmailWebReport 1217
  FileName 1217
  JobName 1217
  LanguageCd 1218
  NotifyTextMsgNum 1218
  NotifyTextMsgSet 1219
  OutDest 1219
  OutDestFormat 1220
  OutDestType 1220
  PortalFolder 1221
  ProcessInstance 1221
  ProcessName 1221
  ProcessType 1222
  RunControlID 1223

Copyright © 1988-2007, Oracle. All rights reserved.

RunDateTime 1223
RunLocation 1223
RunRecurrence 1224
RunStatus 1224
Status 1225
TimeZone 1226
ProcessRequest objects 1194
ProcessType property
PostReport class 1189
ProcessRequest class 1222
Product property
Content Reference class 1078
Content Reference Links class 1107
Folder class 1065
Pagelet class 1149
PageletCategory class 1141
TabDefinition class 1118
programs, Application Engine, *See* Application Engine programs
Promotion 221
Prompt property 457
PromptRecord property 1281
PromptRecordFieldName property 1366
Prompts property 1280
PromptTable property 1367
PromptTableName property 514
Prop property 1882
properties 121
AESection class 10
See Also AESection class properties
Agent class 1719
See Also Agent class properties
AgentPhysQueueProps class 1722
AgentPhysQueueTasks class 1724
analytic grid class 135
analytic type definition 147
AnalyticInstance class 21
AnalyticModel class 29
AnalyticModelDefn class 72
AnalyticTypeModelDefn 148
Array class 238
AssignedPagelet class 1123
See Also AssignedPagelet class properties
AssignedPagelet collection 1127
assignment class 105
Attribute collection 1092
AttributeValue class 1087
See Also AttributeValue class properties
AvailableCategory class 1128
AvailableCategory collection 1130
AvailablePagelet class 1131
See Also AvailablePagelet class properties
AvailablePagelet collection 1134
Bindings 202
Bindings collection 201
Business Interlink class 316
Cascade property 1002
CascadedPermissions property 1002
Chart class appearance 354
See Also Chart class appearance properties
Chart class axes 364
See Also Chart class axes properties
Chart class data 353
See Also Chart class data properties
Chart class legend 370
See Also Chart class legend properties
Chart class overlay 374
See Also Chart class overlay properties
Chart class Scatter 376
See Also Chart class Scatter chart property
Chart class title 367
See Also Chart class title properties
Chart class TrueXY line 377
See Also Chart class TrueXY line chart properties
CI collection 860
ClassInfo 207
ClassInfo collection 205
comparison class 107
CompIntfPropInfoCollection collection 454
Component Interface class 440
See Also Component Interface class properties
Component Interface classes 411
See Also Component Interface class properties
Component Interface collection 425
ConnectorInfo collection 825
constant class 109
Content Reference class 1072
See Also Content Reference class properties
Content Reference collection 1087
Content Reference Links class 1102

Copyright © 1988-2007, Oracle. All rights reserved.

See Also Content Reference Links class properties
content references 1013
See Also content reference properties
Cookie class 612
See Also Cookie class properties
Crypt class 479
cube class 112
cube collections 42
CubeCollectionDefn class 96
CubeDefn class 84
Data collection 449, 865
declaring abstract 164
declaring protected 157
deprecated 2091, 2092
DimensionDefn class 75
DynamicCategory collection 1137
Exception class 487
See Also Exception class properties
Favorite class 1165
See Also Favorite class properties
Favorite collection 1169
Field class 500
See Also Field class properties
File class 551
See Also File class properties
Folder class 1061
See Also Folder class properties
Folder collection 1070
FunctionCall class 118
Grid class 580
GridColumn class 581
See Also GridColumn class properties
IBInfo class 810
See Also IBInfo class properties
Incoming Business Interlink class 259, 313
See Also Incoming Business Interlink class properties; Incoming Business Interlink properties
Leaf class 1625
See Also Leaf class properties
Level class 1632
See Also Level class properties
Level collection 1631
Link class 1112
Link collection 1112
ListViewAttrs class 867
See Also ListViewAttrs class properties
LogicalQueue class 1727
mapping functions to 2085
MCFFactory class 1728
MCFMailStore 709
See Also MCFMailStore class properties
MCFMultiPart 688
See Also MCFMultiPart class properties
MCFOutboundEmail 694
See Also MCFOutboundEmail class properties
MemberReference class 119
Message class 769
See Also Message class properties
MethodInfo class 209
MethodInfo collection 208
Mobile class 854
Namespaces class 204
Namespaces collection 203
naming 170
Node class 1046, 1646
See Also Node class properties
Node collection 1050
NodeTemplate class 1056
NodeTemplate collection 1059
Notification class 886
See Also Notification class properties
NotificationAddress class 890
NotificationTemplate class 893
See Also NotificationTemplate class properties
OrganizerDefn class 101
overriding 175, 178
Page class 992
See Also Page class properties
Pagelet class 1146
See Also Pagelet class properties
Pagelet collection 1154
PageletCategory class 1138
See Also PageletCategory class properties
PageletCategory collection 1145
ParseResult class 1475
ParseResult collection 1474
PeerDefaultAttributes class 874
Permissions property 1002
PermissionValue class 1093
See Also PermissionValue class properties
PermissionValue collection 1097

Copyright © 1988-2007, Oracle. All rights reserved.

PhysicalQueue class 1730
See Also PhysicalQueue class properties
Portal class 1053
Portal collection 1055
PortalRegistry class 1041
See Also PortalRegistry class properties
PortalRegistry collection 1045
PostReport class 1188
See Also PostReport class properties
ProcessRequest 1216
See Also ProcessRequest class properties
PropertyAttrs class 870
See Also PropertyAttrs class properties
PropertyInfo class 212
PropertyInfo collection 210
PropertyInfo objects 856
See Also PropertyInfo objects properties
PropertyInfoCollection class 855
PSMessages class 1525
See Also PSMessages class properties
PSMessages collection 1524
Query class 1275
See Also Query class properties
Query collection 1262
Query Metadata class 1344
See Also Query Metadata class properties
Query Metadata collection 1343
QueryCriteria class 1318
See Also QueryCriteria class properties
QueryCriteria collection 1314
QueryDBRecord class 1353
QueryDBRecord collection 1351
QueryDBRecordField class 1357
See Also QueryDBRecordField class properties
QueryDBRecordField collection 1357
QueryExpression class 1331
See Also QueryExpression class properties
QueryExpression collection 1330
QueryField class 1306
See Also QueryField class properties
QueryField collection 1304
QueryList class 1336
QueryListValue class 1337
QueryPrompt class 1364
See Also QueryPrompt class properties
QueryPrompt collection 1363
QueryRecord class 1299
See Also QueryRecord class properties
QueryRecord collection 1298
QueryRecordHierarchy class 1340
See Also QueryRecordHierarchy class properties
QueryRecordHierarchy collection 1339
QuerySecurityProfile class 1346
See Also QuerySecurityProfile class properties
QuerySelect class 1290
See Also QuerySelect class properties
QuerySelect collection 1296
QueryStatistics class 1345
See Also QueryStatistics class properties
Record class 1402
See Also Record class properties
Regional Settings class 1528
See Also Regional Settings class properties
RemoteNode collection 1052
Repository class 201
Request class 599
See Also Request class properties
Response class properties 611
RolePermissionValue collection 1100
Row class 1413
See Also Row class properties
Rowset class 1444
See Also Rowset class properties
RuleDefn class 103
Search FS Options class 1502
See Also Search FS Options class properties
Search HTTP Options class 1499
See Also Search HTTP Options class properties
Search Language class 1495
Search Language collection 1494
Search Schedule class 1498
See Also Search Schedule class properties
Search Schedule collection 1497
Search Start Options class 1506
Search Start Options collection 1506

Copyright © 1988-2007, Oracle. All rights reserved.

SearchField class 1480
SearchField collection 1480
SearchIndex class 1483
See Also SearchIndex class properties
SearchIndexes collection 1482
SearchQuery class 1470
See Also SearchQuery class properties
SearchRecord Options class 1486
See Also SearchRecord Options class properties
SearchRecordField class 1491
See Also SearchRecordField class properties
SearchRecordField collection 1490
SearchResult class 1477
See Also SearchResult class properties
SearchResult collection 1477
SelectedPagelet class 1161
See Also SelectedPagelet class properties
SelectedPagelet collection 1164
Session class 1519
See Also Session class properties
SOAPDoc class 1556
See Also SOAPDoc class properties
SQL class 1577
See Also SQL class properties
SyncServer class 1599
See Also SyncServer class properties
TabDefinition class 1114
See Also TabDefinition class properties
TabDefinition collection 1122
Task class 1737
See Also Task class properties
TaskList class 1739
See Also TaskList class properties
Tree class 1676
See Also Tree class properties
Tree Structure class 1693
See Also Tree Structure class properties
understanding 174
understanding API Repository 194
UserFunctionDefn class 98
UserHomepage class 1155
UserTab class 1156
See Also UserTab class properties
UserTab collection 1160
using for collections 177
See Also PeopleCode collections
using Get/Set 176
using PortalRegistry object security properties 1001
using read-only 176
variable class 123
WorklistEntry class 903
See Also WorklistEntry class properties
XmlDoc class 1764
XmlNode class 1797
See Also XmlNode class properties
Properties class
constructor 1867
methods 1891
Properties class methods
GetProperty 1891
SetProperty 1891
Properties property 208
properties, OptEngine class, *See* OptEngine class properties
property
Branch collection 1618
Chart class Gantt 394
See Also Chart class Gantt properties
PropertyAttrs class
list of methods, properties, returns 1964
methods 869
objects 869
properties 870
See Also PropertyAttrs class properties
PropertyAttrs class objects 869
PropertyAttrs class properties
DisplayOnly 870
InError 871
Label 871
ShowRequiredCue 871
Visible 872
PropertyInfo class
list of properties, returns 1962, 2018
objects 856
properties 212
understanding 835, 856
PropertyInfo class objects 856
PropertyInfo collection
list of methods, properties, returns 1962
list of properties, returns 2018
methods 211
properties 210
PropertyInfo object properties
DefaultValue property 856
HasDefaultValue property 856
IsCollection property 857

Copyright © 1988-2007, Oracle. All rights reserved.

IsPeerReference property 857
IsRefToParent property 857
IsSimpleApplicationProperty property 858
IsSystemProperty property 858
Name property 858
PropertyInfoCollection property 859
PropertyInfoCollection class
methods 854
understanding 835, 854
PropertyInfoCollection class properties 855
PropertyInfoCollection objects 854
PropertyInfoCollection property
CompIntfPropInfoCollection objects 457
Component Interface class 443
Mobile class property 854
PropertyInfo object 859
PropertyValue property 1602
protected
declaring methods 157
declaring properties 157
protected statements 483
Protocol property 602
ProxyHost property 1501
ProxyPort property 1501
PS_CLASSPATH environment variable 616
PS_MESSAGE_LOG table 921
PSMessages
class 1525
See Also PSMessages class
collection 1522
See Also PSMessages collection
objects 1512
PSMessages property 1519
PSMessagesMode property 1520
PSMessages class
list of properties, returns 2013
properties 1525
See Also PSMessages class properties
PSMessages class properties
ExplainText 1525
MessageNumber 1525
MessageSetNumber 1525
MessageType 1525
Source 1526
Text 1528
PSMessages collection
accessing 1522
displaying error messages 1513
list of methods, properties, returns 2012
logging errors for PortalRegistry classes 1004
logging errors for Query classes 1246
logging errors for Search classes 1464
logging errors for Session class 1512
logging errors for Tree classes 1612
methods 1522
See Also PSMessages collection methods
properties 1524
PSMessages collection methods
DeleteAll 1522
DeleteItem 1523
First 1523
Item 1523
Next 1524
PSMessages objects 1512
PSMessages property 1519
PSMessagesMode property 1520
PT_DETMSGS record 926
PT_MCF_MAIL application package 629
PT_OPT_BASE:OptBase application class, *See* OptBase class methods
PT_OPTCALL program 927
PT_OPTDETMSGS record 921
PT_OPTPARMS record
checking optimization engine status 923
getting the trace level 925
running transactions 923
sending optimization status 926
setting the trace level 924
shutting down optimization engines 926
understanding OPT_CALL messages 921
understanding request messages 923
understanding response messages 926
PubID property 779
public
declaring external interfaces 156
properties, creating 174
PublicAccess property
Content Reference class 1078
Content Reference Links class 1108
Folder class 1066
Pagelet class 1150

Copyright © 1988-2007, Oracle. All rights reserved.

PageletCategory class 1141
setting object access 1001
TabDefinition class 1118
publication
Publish method 795
publishing to partial records 735
PublicationID property 816
PublicPrivate property 1281
Publish method
IntBroker class 795
Message class 760
ReportDefn class 1824
PubNodeName property 779
Push method 229
Put method 1187

Q

QualifiedURL property
Content Reference class 1078
Favorite class 1165
Pagelet class 1150
TabDefinition class 1118
UserTab class 1157
QuarantineCount property 667
QuarantineFolder property 667
queries
adding criteria 1369
adding expressions 1241
collections 1260
See Also Query collections
creating 1367
database queries 1239
"Having" criteria 1248
life-cycle 1234
objects 1233
See Also query objects
operator and Expression 2 dependencies 1241
outer joins 1375
Query class 1263
See Also Query class
Query classes 1233
See Also Query classes
Query Manager 1233
Query Metadata class 1344
See Also Query Metadata class
Query Metadata collection 1341
See Also Query Metadata collection
Query Monitor 1243
QueryCriteria class 1314
See Also QueryCriteria class
QueryCriteria collection 1312
See Also QueryCriteria collection
QueryDBRecord class 1352
See Also QueryDBRecord class
QueryDBRecord collection 1349
See Also QueryDBRecord collection
QueryDBRecordField class 1357
See Also QueryDBRecordField class
QueryDBRecordField collection 1354
See Also QueryDBRecordField collection
QueryExpression class 1331
See Also QueryExpression class
QueryExpression collection 1328
See Also QueryExpression collection
QueryField class 1304
See Also QueryRecord class
QueryField collection 1301
See Also QueryRecord collection
QueryList class 1334
See Also QueryList class
QueryListValue class 1337
See Also QueryListValue class
QueryPrompt class 1363
See Also QueryPrompt class
QueryPrompt collection 1361
See Also QueryPrompt collection
QueryRecord class 1299
See Also QueryRecord class
QueryRecord collection 1296
See Also QueryRecord collection
QueryRecordHierarchy class 1340
See Also QueryRecordHierarchy class
QueryRecordHierarchy collection 1337
See Also QueryRecordHierarchy collection
QuerySecurityProfile class 1346
See Also QuerySecurityProfile class
QuerySelect class 1283
See Also QuerySelect class
QuerySelect collection 1293
See Also QuerySelect collection
QueryStatistics class 1345
See Also QueryStatistics class
running 1245
SearchQuery class 1468
See Also SearchQuery class
security 1246

Copyright © 1988-2007, Oracle. All rights reserved.

setting the expression type 1241
setting user languages 1246
using criteria and expressions 1241
using metadata 1243
Query class
list of methods, properties, returns 2057
methods 1263
See Also Query class methods
properties 1275
See Also Query class properties
understanding 1263
Query class methods
AddPrompt 1264
AddQuerySelect 1263
Close 1264
Create 1265
Delete 1266
DeletePrompt 1266
FindExpression 1267
FormatResultString 1267
Open 1268
Rename 1269
RunToFile 1270
RunToRowset 1272
Save 1275
Query class properties
Approved 1276
ApproveDtTm 1276
ApproveOprId 1276
ApproveUserId 1276
CreateDtTm 1277
CreateOprId 1277
CreateUserId 1277
Description 1277
Disabled 1277
ExecAppName 1278
ExecLogging 1278
Folder 1278
LastSQLErrorCode 1278
LastUpdDttm 1279
LastUpdOprId 1279
LongDescription 1279
Metadata 1279
MetaSQL 1279
Name 1280
PDFFont 1280
PromptRecord 1281
Prompts 1280
PublicPrivate 1281
QuerySelect 1281
QueryStatistics 1282
RunTimePrompts 1282
SQL 1282
Type 1282
Query classes
adding criteria 1369
collections in 1233
creating queries 1367
examples 1367
handling errors 1246
hierarchy 1236
list of methods, properties, returns 2056
outer joins 1375
Query class 1263
See Also Query class
query fields 1247
Query Metadata 1344
See Also Query Metadata class
QueryCriteria 1314
See Also QueryCriteria class
QueryDBRecord 1352
See Also QueryDBRecord class
QueryDBRecordField 1357
See Also QueryDBRecordField class
QueryExpression 1331
See Also QueryExpression class
QueryField class 1304
See Also QueryField class
QueryList 1334
See Also QueryList class
QueryListValue 1337
See Also QueryListValue class
QueryOutputFields 1247
QueryPrompt 1363
See Also QueryPrompt class
QueryRecord 1299
See Also QueryRecord class
QueryRecordHierarchy 1340
See Also QueryRecordHierarchy class
QuerySecurityProfile 1346
See Also QuerySecurityProfile class
QuerySelect 1283
See Also QuerySelect class
QuerySelectedFields 1247
QueryStatistics 1345
See Also QueryStatistics class
running queries 1245
security 1246
Session class methods 1249
See Also Session class methods

Copyright © 1988-2007, Oracle. All rights reserved.

understanding 1233, 1239
Query collection
list of methods, properties, returns 2057
methods 1260
See Also Query collection methods
properties 1262
understanding 1260
Query collection methods
First 1261
Item 1261
ItemByName 1262
Next 1262
Query collection properties 1262
Query collections
Query 1260
See Also Query collection
Query Metadata 1341
See Also Query Metadata collection
QueryCriteria 1312
See Also QueryCriteria collection
QueryDBRecord 1349
See Also QueryDBRecord collection
QueryDBRecordField 1354
See Also QueryDBRecordField collection
QueryExpression 1328
See Also QueryExpression collection
QueryField 1301
See Also QueryField collection
QueryPrompt 1361
See Also QueryPrompt collection
QueryRecord 1296
See Also QueryRecord collection
QueryRecordHierarchy 1337
See Also QueryRecordHierarchy collection
QuerySelect 1293
See Also QuerySelect collection
understanding 1233
Query Manager 1233
query metadata 1243
class 1344
See Also Query Metadata class
collection 1341
See Also Query Metadata collection
objects 1344
Query Metadata class
list of properties, returns 2070
properties 1344
See Also Query Metadata class properties
understanding 1344
Query Metadata class properties
Name 1344
Value 1344
Query Metadata collection
list of methods, properties, returns 2069
methods 1341
See Also Query Metadata collection methods
properties 1343
understanding 1341
Query Metadata collection methods
First 1341
Item 1342
ItemByName 1342
Next 1343
query metadata objects 1344
Query Monitor 1243
query objects
declaring/scope 1248
QueryCriteria 1240
QueryDBRecordFields 1240
QueryDBRecords 1240
QueryOutputFields 1240
QueryRecords 1239
QuerySelectedFields 1240
understanding 1233, 1239, 1263
understanding QuerySelect 1239
QueryCriteria
class 1314
See Also QueryCriteria class
collection 1312
See Also QueryCriteria collection
objects 1240, 1314
See Also QueryCriteria objects
QueryCriteria class
list of methods, properties, returns 2065
methods 1315
See Also QueryCriteria class methods
properties 1318
See Also QueryCriteria class properties
understanding 1314
QueryCriteria class methods
AddExpr1Expression 1315
AddExpr1Field 1315
AddExpr2Expression 1316
AddExpr2Field1 1316
AddExpr2Field2 1317

Copyright © 1988-2007, Oracle. All rights reserved.

AddExpr2List 1318
AddExpr2Subquery 1318
QueryCriteria class properties
Expr1Expression 1319
Expr1Field 1319
Expr1Type 1319
Expr2Constant1 1320
Expr2Constant2 1320
Expr2Expression1 1320
Expr2Expression2 1321
Expr2Field1 1321
Expr2Field2 1321
Expr2List 1322
Expr2Subquery 1322
Expr2Type 1322
Logical 1325
LParenLvl 1325
Name 1325
Negation 1326
Operator 1326
R1ExprNum 1327
R1ExprType 1328
R2ExprNum 1328
R2ExprType 1328
RParenLvl 1328
QueryCriteria collection
list of methods, properties, returns 2064
methods 1312
See Also QueryCriteria collection methods
properties 1314
understanding 1312
QueryCriteria collection methods
First 1312
Item 1313
ItemByName 1313
Next 1314
QueryCriteria objects 1240, 1314
QueryDBRecord
class 1352
See Also QueryDBRecord class
collection 1349
See Also QueryDBRecord collection
objects 1352
QueryDBRecord class
list of methods, properties, returns 2072
methods 1352
See Also QueryDBRecord class methods
properties 1353
understanding 1352
QueryDBRecord class methods
QueryDBRecordFieldByIndex 1352
QueryDBRecordFieldByName 1353
QueryDBRecord collection
list of methods, properties, returns 2071
methods 1350
See Also QueryDBRecord collection methods
properties 1351
understanding 1349
QueryDBRecord collection methods
First 1350
Item 1350
ItemByName 1351
Next 1351
QueryDBRecord objects 1352
QueryDBRecordField
class 1357
See Also QueryDBRecordField class
collection 1354
See Also QueryDBRecordField collection
objects 1357
QueryDBRecordField class
list of properties, returns 2073
properties 1357
See Also QueryDBRecordField class properties
understanding 1357
QueryDBRecordField class properties
Decimal 1357
Description 1358
Flag 1358
Format 1359
Length 1359
LongName 1360
LookupTableName 1360
Name 1360
Shortname 1360
Type 1360
QueryDBRecordField collection
list of methods, properties, returns 2072
methods 1354
See Also QueryDBRecordField collection methods
properties 1357
understanding 1354
QueryDBRecordField collection methods
First 1354

Copyright © 1988-2007, Oracle. All rights reserved.

Item 1355
ItemByName 1355
Next 1356
Sort 1356
QueryDBRecordField collection
properties 1357
QueryDBRecordField objects 1357
QueryDBRecordFieldByIndex
method 1352
QueryDBRecordFieldByName
method 1353
QueryDBRecordFields property 1353
QueryDS class
constructor 1858
methods 1859
QueryDS methods
GetXMLData 1859
GetXMLSampleData 1860
GetXSDSchema 1861
QueryExpression
class 1331
See Also QueryExpression class
collection 1328
See Also QueryExpression collection
objects 1331
QueryExpression class
list of properties, returns 2067
properties 1331
See Also QueryExpression class properties
understanding 1331
QueryExpression class properties
Aggregate 1331
BindFlag 1331
Decimal 1332
ExpNum 1332
IsXlatExpression 1332
Length 1332
Name 1332
OutputField 1333
RightExprFlag 1333
SelectedField 1333
Text 1333
Type 1334
QueryExpression collection
list of methods, properties, returns 2066
methods 1329
See Also QueryExpression collection methods
properties 1330
understanding 1328
QueryExpression collection methods
First 1329
Item 1329
ItemByName 1330
Next 1330
QueryExpression objects 1331
QueryField
class 1304
See Also QueryField class
collection 1301
See Also QueryField collection
objects 1304
QueryField class
list of methods, properties, returns 2063
methods 1305
See Also QueryField class methods
properties 1306
See Also QueryField class properties
understanding 1304
QueryField class methods
AddTranslateExpression 1305
AddTranslateField 1305
QueryField class properties
Aggregate 1306
ColumnNumber 1306
Decimal 1307
Description 1306
ExpNum 1307
Flag 1307
Format 1307
HeadingText 1308
HeadingType 1308
HeadingUniqueFieldName 1308
Length 1308
LongName 1309
Name 1309
OrderByDirection 1309
OrderByNumber 1309
QueryRecord 1310
RecordAlias 1310
ShortName 1310
TranslateEffDtLogic 1311
TranslateExpression 1311
TranslateField 1311
TranslateOption 1311
Type 1312
QueryField collection
list of methods, properties, returns 2062
methods 1302

Copyright © 1988-2007, Oracle. All rights reserved.

See Also QueryField collection methods
properties 1304
understanding 1301
QueryField collection methods
First 1302
Item 1302
ItemByExpNum 1303
ItemByNameAndAlias 1303
Next 1303
QueryField objects 1304
QueryFields property 1301
QueryList class
list of methods, properties, returns 2068
methods 1334
See Also QueryList class methods
properties 1336
understanding 1334
QueryList class methods
AddListValue 1334
First 1335
Item 1335
Next 1336
QueryListValue class
list of properties, returns 2068
properties 1337
understanding 1337
QueryOutputFields property 1291
QueryPrompt
class 1363
See Also QueryPrompt class
collection 1361
See Also QueryPrompt collection
QueryPrompt class
list of properties, returns 2074
properties 1364
See Also QueryPrompt class properties
understanding 1363
QueryPrompt class properties
EditType 1364
FieldDecimal 1364
FieldFormat 1364
FieldLength 1365
FieldName 1365
FieldType 1365
HeadingText 1366
HeadingType 1366
LangCount 1366
Name 1366
PromptRecordFieldName 1366
PromptTable 1367
UniquePromptName 1367
UseCount 1367
QueryPrompt collection
list of methods, properties, returns 2073
methods 1361
See Also QueryPrompt collection methods
properties 1363
understanding 1361
QueryPrompt collection methods
First 1361
Item 1361
ItemByName 1362
Next 1362
QueryPrompt collection properties 1363
QueryRecord
class 1299
See Also QueryRecord class
collection 1296
See Also QueryRecord collection
objects 1299
QueryRecord class
list of methods, properties, returns 2062
methods 1299
properties 1299
See Also QueryRecord class properties
understanding 1299
QueryRecord class methods 1299
QueryRecord class properties
Description 1300
JoinAlias 1300
JoinFieldName 1300
JoinType 1300
Name 1301
QueryFields 1301
RecordAlias 1301
QueryRecord collection
list of methods, properties, returns 2061
methods 1296
See Also QueryRecord collection methods
properties 1298
understanding 1296
QueryRecord collection methods
First 1296
Item 1297
ItemByAlias 1297
Next 1298
QueryRecord collection properties 1298

Copyright © 1988-2007, Oracle. All rights reserved.

QueryRecord objects 1299
QueryRecord property 1310
QueryRecordHierarchy
  class 1340
    See Also QueryRecordHierarchy class
  collection 1337
    See Also QueryRecordHierarchy collection
  objects 1337
QueryRecordHierarchy class
  list of properties, returns 2069
  properties 1340
    See Also QueryRecordHierarchy class properties
  understanding 1340
QueryRecordHierarchy class properties
  Description 1340
  Level 1340
  Name 1341
  ParentFlag 1341
QueryRecordHierarchy collection
  list of methods, properties, returns 2068
  methods 1337
    See Also QueryRecordHierarchy collection methods
  properties 1339
  understanding 1337
QueryRecordHierarchy collection methods
  First 1338
  Item 1338
  ItemByName 1338
  Next 1339
QueryRecordHierarchy objects 1337, 1340
QueryRecords property 1291
QuerySecurityProfile class
  list of properties, returns 2070
  properties 1346
    See Also QuerySecurityProfile class properties
  understanding 1346
QuerySecurityProfile class properties
  AllowAnyJoin 1346
  AllowDistinct 1346
  AllowExpressions 1346
  AllowSubqueries 1347
  AllowUnions 1347
  ApprovePrivateQuery 1347
  ApprovePublicQuery 1347
  CanCreatePublic 1347
  CanCreateWorkFlow 1347
  CanModifyQuery 1348
  CanRunQuery 1348
  CanRunToCrystal 1348
  CanRunToExcel 1348
  LimitUnapproved 1348
  MaxInTreeCriteria 1349
  MaxJoins 1349
  MaxRowsToFetch 1349
  MaxUnapprovedRows 1349
QuerySelect
  class 1283
    See Also QuerySelect class
  collection 1293
    See Also QuerySelect collection
  objects 1283
  understanding select statements 1239
QuerySelect class
  list of methods, properties, returns 2059
  methods 1284
    See Also QuerySelect methods
  properties 1290
    See Also QuerySelect class properties
  understanding 1283
QuerySelect class properties
  Criteria 1290
  Distinct 1290
  Expressions 1290
  HavingCriteria 1291
  ParentSelectNum 1291
  QueryOutputFields 1291
  QueryRecords 1291
  QuerySelectedFields 1292
  QuerySelects 1292
  SelectNum 1292
  SelectType 1292
QuerySelect collection
  list of methods, properties, returns 2061
  methods 1293
    See Also QuerySelect collection methods
  properties 1296
  understanding 1293
QuerySelect collection methods
  AddUnion 1293
  DeleteQuerySelect 1293
  First 1294
  Item 1294
  ItemBySelNum 1295
  Next 1295

Copyright © 1988-2007, Oracle. All rights reserved.

QuerySelect methods
AddAllFields 1284
AddCriteria 1284
AddExpression 1285
AddHavingCriteria 1285
AddQueryOutputField 1286
AddQueryRecord 1286
AddQuerySelectedField 1287
DeleteCriteria 1288
DeleteExpression 1288
DeleteField 1288
DeleteHavingCriteria 1289
DeleteRecord 1289
QuerySelect objects 1283
QuerySelect property 1281
QuerySelectedFields property 1292
QuerySelects property 1292
QueryStatistics class
list of properties, returns 2070
properties 1345
See Also QueryStatistics class properties
understanding 1345
QueryStatistics class properties
AvgExecTime 1345
AvgFetchTime 1345
AvgNumRows 1345
ExecCount 1345
LastExecDtTm 1346
QueryStatistics property 1282
QueryString property 602
QueryText property 1472
QueueName property 780
queues, universal, *See* universal queues
QueueSeqId property 780

R

R1ExprNum property 1327
R1ExprType property 1328
R2ExprNum property 1328
R2ExprType property 1328
RangeFrom property 1628
RangeTo property 1628
read-only properties 176
readable property 151
ReadAllEmailHeadersWithAttach method 658
ReadBlobFromFile method 848
ReadEmails method 659
ReadEmailsWithAttach method 660
ReadEmailsWithUID method 661
ReadEmailWithAttach method 661
ReadFromDatabase method 671
ReadHeaders method 662
ReadLine method 537
ReadOnce property 151
ReadRecord method 560
ReadRowset method 538, 565
Reassign method
Worklist class 896
WorklistEntry class 898
Recalculate method 28
Recipients property
MCFEmail class 655
MCFOutboundEmail class 702
recname property 1417
RecName property 1486
RecOpts property 1484
Record class
list of functions, methods, properties, returns 1979
methods 1378
See Also Record class methods
properties 1402
See Also Record class properties
shortcut considerations 1377
understanding 1377
using SQL 1564
Record class methods
CompareFields 1379
CopyChangedFieldsTo 1380
CopyFieldsTo 1381
creating SQL statements 1378
DBPatternMatch 1384
Delete 1385
ExecuteEdits 1386, 1399
GetField 1388
Insert 1390
Save 1391
SearchClear 1394
SelectByKey 1394
SelectByKeyEffDt 1396
SetDefault 1398
SetEditTable 1387, 1398
Update 1400
Record class properties
FieldCount 1402
fieldname 1402
IsChanged 1402
IsDeleted 1403

Copyright © 1988-2007, Oracle. All rights reserved.

IsEditError 1403
Name 1404
ParentRow 1404
RelLangRecName 1404
record objects
declaring/scope 1379
understanding 1377, 1563
RecordAlias property
QueryField class 1310
QueryRecord class 1301
RecordCount property 1417
RecordHierachy property 1354
RecordName property
cube collection 97
SearchRecordField class 1492
records
AdvancedSearchRecords method 1252
callback 962
case sensitivity 1564
editing via ExecuteEdits /SetEditTable 749
FindQueryDBRecords method 1253
objects 1377
See Also record objects
PT_DETMSGS 926
PT_OPTDETMSGS 921
PT_OPTPARMS 921
See Also PT_OPTPARMS record
publishing/subscribing to partial 735
query definitions 1239
QueryDBRecords 1240
Record class 1377, 1564
See Also Record class
sorting 1428
understanding 1377, 1563
understanding work records 333
RedirectURL method 609
referencing
referencing PortalRegistry objects via C/C++ 1181
referencing PortalRegistry objects via Visual Basic 1180
self-referencing 171
superclass referencing 178
using arrays with object references 230
RefIDs property
MCFEmail class 655
MCFOutboundEmail class 702
Refresh method
Agent class 1718
AgentPhysQueueTasks class 1723
chart data 345
PhysicalQueue class 1729
Rowset class 1438
Task class 1735
TaskList class 1739
RefreshDescription method 1644
refreshing
PhysicalQueue class Refresh method 1729
PhysicalQueue class RefreshTaskList method 1729
Task class Refresh method 1735
Task class RefreshStatus method 1736
RefreshQTaskList method 1719
RefreshStatus method 1736
RefreshTaskList method 1729
regional settings
class 1528
See Also Regional Settings class
objects 1513
RegionalSettings property 1521
Regional Settings class
list of properties, returns 2013
properties 1528
See Also Regional Settings class properties
Regional Settings class properties
1159Separator 1531
2359Separator 1532
ClientTimeZone 1528
CurrencyFormat 1529
CurrencySymbol 1529
DateFormat 1529
DateSeparator 1530
DecimalSymbol 1530
DigitGroupingSymbol 1530
LanguageCD 1530
UseLocalTime 1531
regional settings objects 1513
RegionalSettings property 1521
RegisteredBy property
DataSourceDefn class 1816
ReportDefn class 1832
TemplateDefn class 1837
RegisteredDTTM property
DataSourceDefn class 1817
ReportDefn class 1832
TemplateDefn class 1837
registration, content cxxi

Copyright © 1988-2007, Oracle. All rights reserved.

registry, portal, *See* portal registry
related documentation cxiv
relative paths
  ReadBlobFromFile method 848
  using 534
  WriteBlobToFile method 853
RelativeURL property
  Content Reference class 1078
  Content Reference Links class 1108
  Request class 603
RelLangRecName property 1404
RemoteAddr property 603
RemoteFrameworkURL property 826
RemoteHost property 603
RemoteNode collection
  list of methods, properties, returns 2034
  methods 1051
    See Also RemoteNode collection methods
  properties 1052
  understanding 1051
RemoteNode collection methods
  First 1051
  ItemByName 1051
  Next 1052
RemoteUser property 604
Remove method 1631
RemoveAllChildNode method 1795
RemoveChildNode method 1796
RemoveEmail method 663
RemoveEmails method 664
RemoveUser method 728
Rename method
  analytic type definition 146
  AnalyticModelDefn class 67
  Node class 1645
  Query class 1269
  Tree class 1668
  Tree Structure class 1692
RenameCube method 67
RenameCubeCollection method 68
RenameDimension method 68
RenameExplicitDimensionSet method 69
RenameMember method 29
RenameOrganizer method 70
RenameUserFunction method 70
RENURLID property 1733
Replace method 231
ReplyIDs property
  MCFEmail class 655
  MCFOutboundEmail class 702
ReplyTo property
  MCFEmail class 655
  MCFOutboundEmail class 702
report class
  constructor 1848
report class properties 1850
  contentId 1850
  CreatedDateTime 1851
  DatabaseName 1851
  ExpireDate 1851
  FolderName 1851
  ProcessInstanceID 1852
  ReportInstanceID 1852
  ReportName 1852
  ReportURL 1852
report manager definition classes
  constructors 1810
  hierarchy 1804
  search operator values 1809
  understanding 1802
report manager definition constructors
  DataSourceDefn 1810
  ReportDefn 1811
  TemplateDefn 1811
  TemplateFile 1812
  TranslationFile 1813
report manager search classes
  constructors 1848
  report class 1850
  report class constructor 1848
  ReportManager 1852
Report search manager classes
  search operator values 1809
ReportCategoryID property 1838
ReportDefn class
  constructor 1811
  methods 1817
  properties 1828
ReportDefn class methods
  DisplayOutput 1817
  Get 1818
  GetActiveTemplatesByDistributionChannel 18
  GetDataSource 1818
  GetDefaultTemplate 1819
  GetOutDestFormatString 1819
  GetPSQueryPromptRecord 1820
  GetTemplate 1821
  GetTemplateList 1821
  PrintOutput 1821

Copyright © 1988-2007, Oracle. All rights reserved.

ProcessEtextReport 1822
ProcessReport 1823
Publish 1824
SetPSQueryPromptRecord 1825
SetRuntimeDataRowset 1826
SetRuntimeDataXMLDoc 1826
SetRuntimeDataXMLFile 1827
SetRuntimeProperties 1828
ReportDefn class properties
AllowRecipientEntry 1829
AllowViewerEntry 1829
ArchiveAfter 1829
BurstFieldName 1829
CategoryId 1830
Description 1830, 1851
DestinationPrinter 1830
FolderName 1830
ID 1830
IsReadOnly 1831
LastUpdateDTTM 1831
LastUpdateOprID 1831
ObjectOwnerID 1831
OutDestination 1831
OutDestinationType 1831
OutputEditable 1832
RegisteredBy 1832
RegisteredDTTM 1832
SecurityFieldName 1832
SecurityIDType 1832
SecurityJoinTable 1833
Status 1833
TemplateControlFieldName 1833
UseBurstValueAsOutputFileName 1834
ReportDescr property 1190
ReportFolder property 1190
ReportId property 1190
ReportInstanceID property 1852
ReportManager class 1852
constructor 1849
methods 1853
ReportManager methods
AddSearchFieldCriteria 1853
GetReportList 1853
SetBurstFieldCritera 1854
SetCaseSensitive 1854
SetDateCriteria 1855
SetFolderCriteria 1855
SetProcessInstanceCriteria 1856
SetReportIDCriteria 1856
SetUserIdCriteria 1857
ReportName property 1852
reports
ReportFolder property 1190
ReportId property 1190
ReportURL property 1852
Repository class
list of properties, returns 2016
properties 201
See Also Repository class properties
Repository class properties
Bindings 201
Namespaces 201
Repository property 1521
Request class
list of methods, properties, returns 1939
methods 596
See Also Request class methods
mobile PeopleCode 875
properties 599
See Also Request class properties
understanding 594
Request class methods
GetContentBody method 596
GetCookieNames 597
GetCookieValue method 597
GetHeader 597
GetHeaderNames 598
GetHelpURL method 598
GetParameter method 598
GetParameterNames method 599
GetParameterValues method 599
Request class properties
AuthTokenDomain 599
AuthType 600
BrowserPlatform 600
BrowserType 600
BrowserVersion 600
ByPassSignOn 600
ContentURI 600
ExpireMeta 601
FullURI 601
HTTPMethod 602
LogoutURL 602
PathInfo 602
Protocol 602
QueryString 602
RelativeURL 603
RemoteAddr 603
RemoteHost 603
RemoteUser 604

Copyright © 1988-2007, Oracle. All rights reserved.

RequestURI 603
Scheme 604
ServerName 604
ServerPort 605
ServletPaty 605
Timeout 605
request messages
creating 922
editing PeopleCode 927
getting the trace level 925
setting the trace level 924
understanding 923
request objects
accessing 593
declaring 594
understanding 594
%Request system variable 593
RequestedFields property 1472
RequestingNodeDescription property 817
RequestingNodeName property 817
requestmessageid property 906
requests
class 594
See Also Request class
objects 593
See Also request objects
%Request system variable 593
RequestURI property 603
Required property 457
reserved internal functions
Generate method 203
reserved words 2088
ResetCursor method 302
ResolveCircularDeps property 74
Response class
list of methods, properties, returns 1941
methods 605
See Also Response class methods
properties 611
understanding 595
Response class methods
Clear 606
CreateCookie 606
GetCookie 606
GetCookieNames 606
GetHeader 607
GetHeaderNames 607
GetImageURL 607
GetJavaScriptURL 608
GetStyleSheetURL 608
RedirectURL 609
SetContentType 609
SetHeader 609
UseSimpleURL 610
Write 610
WriteLine 611
response messages
building 932
creating 926
editing PeopleCode 931
understanding 926
response objects
accessing 593
declaring 594
understanding 595
%Response system variable 593
ResponseAsAttachment property 817
responses
class 595
See Also Response class
objects 593
See Also response objects
%Response system variable 593
Responses property 894
Responses property 894
ResponseStatus property 780, 906
RestoreBounds method 985
Resubmit method 796
Result property 479
ResultOfSend property 702
RetrieveEmail method 678
return receipts*, See* email
ReuseCursor property 1568, 1580
Reverse method 232
RevokePermissionForComponent method 1038
RevokePermissionForScript method 1039
RightExprFlag property 1333
RolePermissions property
Content Reference class 1079
Content Reference Links class 1108
Folder class 1065
RolePermissionValue
collection 1097
See Also RolePermissionValue collection
RolePermissionValue collection
methods 1098

Copyright © 1988-2007, Oracle. All rights reserved.

See Also RolePermissionValue collection methods
properties 1100
understanding 1097
RolePermissionValue collection methods
DeleteItem 1098
First 1098
InsertItem 1099
ItemByName 1099
Next 1100
RolePermissionValue Collection methods 2043
RootFolder property 1043
RotationAngle property 358
Row class
list of functions, methods, properties, returns 1981
methods 1408
See Also Row class methods
properties 1413
See Also Row class properties
shortcut considerations 1408
understanding 1407
Row class methods
CopyTo 1409
GetNextEffRow 1409
GetPriorEffRow 1410
GetRecord 1410
GetRowset 1412
scrollname 1413
Row class properties
ChildCount 1413
DeleteEnabled 1414
IsChanged 1415
IsDeleted 1415
IsEditError 1416
IsNew 1416
ParentRowset 1416
recname 1417
RecordCount 1417
RowNumber 1417
Selected 1418
Style 1418
Visible 1419
row objects
declaring/scope 1408
understanding 1407
Row property
AssignedPagelet class 1124
AvailablePagelet class 1132
SelectedPagelet class 1161
RowCount property 1451
RowFetch property 1536
RowNumber property 1417
rows
effective-dated 1437
objects 1407
See Also row objects
Row class 1407
See Also Row class
RowsAffected property 1580
Rowset class
list of functions, methods, properties, returns 1982
methods 1423
See Also Rowset class methods
properties 1444
See Also Rowset class properties
shortcut considerations 1422
understanding 1421
Rowset class methods
ClearDeletesChanges 1423
CopyTo 1424
DeleteRow 1426
Fill 1428
FillAppend 1430
Flush 1432
FlushRow 1433
GetCurrEffRow 1434
GetNewEffRow 1434
GetRow 1435
HideAllRows 1436
InsertRow 1437
Refresh 1438
Select 1439
SelectNew 1441
SetDefault 1442
ShowAllRows 1443
Sort 1443
Rowset class properties
ActiveRowCount 1445
ChangeOnInit 1445
DataAreaCollapsed 1445
DBRecordName 1446
DeleteEnabled 1446
EffDt 1447
EffSeq 1448
InsertEnabled 1448
IsEditError 1449
Level 1449

Copyright © 1988-2007, Oracle. All rights reserved.

Name 1450
ParentRow 1450
ParentRowset 1451
RowCount 1451
SetComponentChanged 1451
TopRowNumber 1451
rowset objects
declaring/scope 1422
understanding 1421
rowset-based messages 729
RowsetCache class 1453
data type 1455
error handling 1454
properties 1458
See Also RowsetCache class properties
scope 1455
understanding 1453
using 1453
RowsetCache Class
methods 1455
See Also RowsetCache class methods
RowsetCache class methods
Delete 1455
Get 1456
Save 1457
RowsetCache class properties
Content 1458
Description 1458
RowsetDS class
constructor 1858
methods 1861
RowsetDS methods
AddXMLData 1861
AddXSDSchema 1862
GetXMLData 1863
GetXSDSchema 1863
rowsets
Business Interlink objects 255
component buffer 731
message 731
objects 1421
See Also rowset objects
Rowset class 1421
See Also Rowset class
RowsetCache class 1453
See Also RowsetCache class
structure for email 635
transforming XML 1748
understanding 1421
understanding charts 320
rowsets, file 565
RParenLvl property 1328
RTFProcessor class
constructor 1867
methods 1892
properties 1894
RTFProcessor methods
GenerateXSL 1892
GenerateXSLXliff 1893
RTFProcessor properties
ConfFile 1894
ConfProp 1895
Locale 1895
RuleDefn class
AddRuleExpression method 102
methods 102
properties 103
RuleString property 103
RuleExpressions
classes 103
using constant class 104
rules 82, 83, 97
adding
ExpressionBlock class 112
RuleDefn class 102
generating
assignment class 104
comparison class 106
constant class 109
cube class 111
FunctionCall class 117
MemberReference class 119
Operation class 120
RuleDefn class 102
variable class 122
getting
ExpressionBlock class 113
RuleString property 103
RunAsync method 19
RunAsynch method
running optimization transactions 918
understanding 954
RunCntlId property 1498
RunControlID property 1223
RunDateTime property 1223
RunJobSetNow method 1211
RunLocation property 1223
RunRecurrence property
ProcessRequest class 1224
Search Schedule class 1499

Copyright © 1988-2007, Oracle. All rights reserved.

RunStatus property 1224
RunSync method 19
RunSynch method
running optimization transactions 918
understanding 956
RunTimePrompts property 1282
RunToFile method 1245, 1270
RunToRowset method 1234, 1245, 1272

S

SameTime instant messaging 723
Save method
AESection class 8
analytic type definition 147
AnalyticModelDefn class 71
Component Interface class 439
Content Reference class 1071
Content Reference Links class 1102
Favorite collection 1169
Folder class 1060
Mobile class 849
Pagelet class 1145
PageletCategory class 1138
Portal class 1053
PortalRegistry class 1040
Query class 1275
Record class 1391
RowsetCache class 1457
SearchIndex class 1483
TabDefinition class 1113
Tree class 1669
Tree Structure class 1693
UserHomepage class 1155
WorklistEntry class 899
SaveAs method 1670
SaveAsDraft method 1671
SaveDraft method 1672
SaveToDatabase method 672
SaveWithCustomData method 900
scatter chart 374, 376
ScenarioManaged property 151
Schedule method 1211
Schedules property 1485
schedules, search, *See* search schedules
schema
messages 791, 794
Scheme property 604
Score property 1478
ScoreAsNumber property 1478
ScorePrecision property 1472
scrollname method 1413
Search classes
examples 1507
general purpose search 1507
handling errors 1464
hierarchy 1462
list of methods, properties, returns 2022
ParseResult 1475
See Also ParseResult class
portal search API 1508
Search FS Options 1502
See Also Search FS Options class
Search HTTP Options 1499
See Also Search HTTP Options class
Search Language 1495
See Also Search Language class
Search Schedule 1498
See Also Search Schedule class
Search Start Options 1506
SearchField 1480
See Also SearchField class
SearchIndex 1482
See Also SearchIndex class
SearchQuery 1468
See Also SearchQuery class
SearchRecord Options 1485
See Also SearchRecord Options class
SearchRecordField 1491
See Also SearchRecordField class
SearchResult 1477
See Also SearchResult class
understanding 1459
Search collections
ParseResult 1473
See Also ParseResult collection
Search Language 1492
See Also Search Language collection
Search Schedule 1495
See Also Search Schedule collection
Search Start Options 1504
See Also Search Start Options collection
SearchField 1479
See Also SearchField collection
SearchIndexes 1481
See Also SearchIndexes collection
SearchRecordField 1488
See Also SearchRecordField collection
SearchResult 1460, 1476
See Also SearchResult collection

Copyright © 1988-2007, Oracle. All rights reserved.

Search FS Options
class 1502
See Also Search FS Options class
objects 1502
Search FS Options class
properties 1502
See Also Search FS Options class properties
understanding 1502
Search FS Options class properties
changing values 1502
GlobList 1502
GlobListType 1502
MIMEList 1503
MIMEListType 1503
StartOpts 1503
Search FS Options objects 1502
Search HTTP Options
class 1499
See Also Search HTTP Options class
objects 1499
Search HTTP Options class
properties 1499
See Also Search HTTP Options class properties
understanding 1499
Search HTTP Options class properties
AllowHTTPS 1499
changing values 1499
DomainLimit 1500
GlobList 1500
GlobListType 1500
LinkDepth 1500
MIMEList 1501
MIMEListType 1501
ProxyHost 1501
ProxyPort 1501
StartOpts 1502
Search HTTP Options objects 1499
search indexes
class 1482
See Also SearchIndex class
collection 1481
See Also SearchIndexes collection
objects 1459
understanding 1459, 1482
Search Language
class 1495
See Also Search Language class
collection 1492
See Also Search Language collection objects 1495
Search Language class
properties 1495
understanding 1495
Search Language collection
methods 1492
See Also Search Language collection methods
properties 1494
understanding 1492
Search Language collection methods
DeleteItem 1492
First 1493
InsertItem 1493
ItemByName 1494
Next 1494
Search Language objects 1495
search objects 1465
search operator values 1809
search queries
class 1468
See Also SearchQuery class
objects 1468
PortalRegistry class GetSearchQuery method 1467
SearchQuery objects 1465, 1466, 1468
SearchQueryDBRecords method 1256
Session class GetSearchQuery method 1466
Search Schedule class
properties 1498
See Also Search Schedule class properties
understanding 1498
Search Schedule class properties
BuildType 1498
changing values 1498
RunCntlId 1498
RunRecurrence 1499
ServerName 1499
Search Schedule collection
methods 1495
See Also Search Schedule collection methods
properties 1497
understanding 1495
Search Schedule collection methods
DeleteItem 1496
First 1496

Copyright © 1988-2007, Oracle. All rights reserved.

InsertItem 1496
ItemByName 1497
Next 1497
search schedule objects 1498
search schedules
class 1498
See Also Search Schedule class
collection 1495
See Also Search Schedule collection
objects 1498
Search Start
objects 1506
options 1504
See Also Search Start Options
Search Start objects 1506
Search Start Options
class 1506
collection 1504
See Also Search Start Options collection
Search Start Options class 1506
Search Start Options collection
methods 1504
See Also Search Start Options collection methods
properties 1506
understanding 1504
Search Start Options collection methods
DeleteItem 1504
First 1504
InsertItem 1505
ItemByName 1505
Next 1506
SearchAttribute class 1857
constructor 1850
properties 1857
SearchAttribute properties
attrName 1857
attrValue 1857
compareOp 1858
FileURL 1851
SearchClear method
Field class 498
Record class 1394
SearchDefault property
Field class 514
using with SearchClear property 1394
SearchEdit property 515
searches
AdvancedSearchQueries method 1249
AdvancedSearchRecords method 1252
languages 1492
See Also Search Language
objects 1465
results 1460
See Also SearchResult
schedules 1495
See Also search schedules
Search classes 1459
See Also Search classes
search collection path 1467
search indexes 1459
See Also search indexes
SearchIndex objects 1459
understanding 1459
SearchField
class 1480
See Also SearchField class
collection 1479
See Also SearchField collection
objects 1480
SearchField class
list of properties, returns 2025
properties 1480
understanding 1480
SearchField collection
list of methods, properties, returns 2024
methods 1479
See Also SearchField collection methods
properties 1480
understanding 1479
SearchField collection methods
First 1479
ItemByName 1479
Next 1480
SearchField collection properties 1480
SearchField objects 1480
SearchFields property 1478
SearchFSOptions class
list of properties, returns 2029
SearchHTTPOptions class
list of properties, returns 2029
SearchIndex class
list of, methods, properties, returns 2025
methods 1483
properties 1483
See Also SearchIndex class properties
understanding 1482

Copyright © 1988-2007, Oracle. All rights reserved.

SearchIndex class method
Save 1483
SearchIndex class properties
changing values 1483
ExtraOptions 1483
FSOpts 1484
HTTPOpts 1484
Languages 1484
Location 1484
Name 1484
RecOpts 1484
Schedules 1485
Type 1485
SearchIndex objects 1459
SearchIndexes collection
list of methods, properties, returns 2025
methods 1481
See Also SearchIndexes collection methods
properties 1482
understanding 1481
SearchIndexes collection methods
First 1481
ItemByName 1481
Next 1482
SearchLanguage class 2028
SearchLanguage collection 2027
SearchPrivateQueries method 1258
SearchPublicQueries method 1259
SearchQuery class
list of methods, properties, returns 2022
methods 1468
See Also SearchQuery class methods
properties 1470
See Also SearchQuery class properties
understanding 1468
SearchQuery class methods
Execute 1468
Parse 1469
SearchQuery class properties
HitCount 1470
Indexes 1470
IndexName 1471
KnowledgeBase 1471
Language 1472
ProcessedCount 1472
QueryText 1472
RequestedFields 1472
ScorePrecision 1472
SortSpecifications 1473
SearchQuery objects 1465, 1466, 1468
SearchQueryDBRecords method 1256
SearchRecord Options
class 1485
See Also SearchRecord Options class
objects 1485
SearchRecord Options class
list of properties, returns 2026
properties 1486
See Also SearchRecord Options class properties
understanding 1485
SearchRecord Options class properties
changing values 1486
Fields 1486
Filter 1486
IncrementalView 1486
RecName 1486
VeggieKey 1487
ZoneOptions 1488
SearchRecord Options objects 1485
SearchRecordField
class 1491
See Also SearchRecordField class
collection 1488
See Also SearchRecordField collection
objects 1491
SearchRecordField class
list of properties, returns 2027
properties 1491
See Also SearchRecordField class properties
understanding 1491
SearchRecordField class properties
changing values 1491
FieldName 1491
IsAttachment 1491
IsVerityField 1492
IsWordIndex 1492
RecordName 1492
SearchRecordField collection
list of methods, properties, returns 2026
methods 1488
See Also SearchRecordField collection methods
properties 1490
understanding 1488
SearchRecordField collection methods
DeleteItem 1488
First 1489

Copyright © 1988-2007, Oracle. All rights reserved.

InsertItem 1489
ItemByName 1490
Next 1490
SearchRecordField objects 1491
SearchResult
class 1477
See Also SearchResult class
collection 1460, 1476
See Also SearchResult collection
objects 1477
understanding 1460
SearchResult class
list of properties, returns 2024
properties 1477
See Also SearchResult class properties
understanding 1477
SearchResult class properties
Key 1478
Score 1478
ScoreAsNumber 1478
SearchFields 1478
SearchResult collection
list of methods, properties, returns 2023
methods 1476
properties 1477
understanding 1460, 1476
using 1460
SearchResult objects 1477
SearchSchedule class 2028
SearchSchedule collection 2028
SearchStartOptions class 2030
SearchStartOptions collection 2030
Secure property 613
security
accessing PeopleSoft 1512
adding for iScripts 589
file objects 525
queries 1246
QuerySecurityProfile class 1346
See Also QuerySecurityProfile class
understanding PortalRegistry 999
using for PortalRegistry classes 1001
SecurityFieldName property 1832
SecurityIDType property 1832
SecurityJoinTable property 1833
SegmentCount property 780
segments
deleting 787
SegmentsByDatabase property 780
SegmentsUnOrder property 817
Select method 1439
SelectAll method 1599
SelectByKey method
Record class 1394
WorklistEntry class 901
SelectByKeyEffDt method 1396
Selected property 1418
SelectedField property 1333
SelectedPagelet
class 1160
See Also SelectedPagelet class
collection 1162
See Also SelectedPagelet collection
SelectedPagelet class
list of properties, returns 2055
properties 1161
See Also SelectedPagelet class properties
understanding 1160
SelectedPagelet class properties
CategoryName 1161
Column 1161
IsMinimized 1161
PageletName 1161
Row 1161
SelectedPagelet collection
list of methods, properties, returns 2055
methods 1162
See Also SelectedPagelet collection methods
properties 1164
understanding 1162
SelectedPagelet collection methods
DeleteItem 1162
First 1162
InsertItem 1163
ItemByName 1163
Next 1164
SelectedPagelets property 1157
SelectNew method 1441
SelectNum property 1292
SelectType property 1292
self-referencing 171
Send method
MCFOutboundEmail class 693
Notification class 886
SendAll method 707
Sender property
MCFEmail class 656
MCFOutboundEmail class 703

Copyright © 1988-2007, Oracle. All rights reserved.

Sensitivity property
MCFEmail class 656
MCFOutboundEmail class 703
SequenceNumber property
Content Reference class 1079
Content Reference Links class 1108
Favorite class 1165
Folder class 1066
Pagelet class 1150
PageletCategory class 1142
TabDefinition class 1119
UserTab class 1157
Server property
MCFInboundEmail 675
SMTPSession class 710
ServerName property
PostReport class 1191
Request class 604
Search Schedule class 1499
ServerPort property 605
servers
application 618
See Also application servers
mail 635
See Also mail servers
PostReport class ServerName property 1191
Request class ServerName property 604
Search Schedule class ServerName property 1499
ServerName property 605
synchronization 1585
See Also synchronization server
service operations 730
Servlet property 605
session
class 423
See Also Session class
objects 1511
See Also session objects
Session class
API instantiation methods 1518
list of functions, methods, properties, returns, variables 2010
methods 423
See Also Session class methods
properties 1519
See Also Session class properties
security 1512
understanding 1511
Session class methods
AdvancedSearchQueries 1249
AdvancedSearchRecords 1252
Connect 1516
Disconnect 1518
FindCompIntfcs 423
FindPortalRegistries 1020
FindQueries 1254
FindQueriesDateRange 1255
FindQueryDBRecords 1253
GetCompIntfc 424, 839
GetLocalNode 1020
GetNodes 1021
GetPortalRegistry 1021
GetQuery 1255
GetQuerySecurityProfile 1256
GetRemoteNodes 1022
GetSearchIndexes 1466
GetSearchQuery 1466
GetTree 1617
GetTreeStructure 1618
SearchPrivateQueries 1258
SearchPublicQueries 1259
SearchQueryDBRecords 1256
Session class properties
ErrorPending 1519
PSMessages 1519
PSMessagesMode 1520
RegionalSettings 1521
Repository 1521
SuspendFormatting 1521
TraceSettings 1521
WarningPending 1521
session objects
declaring 1514
handling errors 1512
scope 1515
state considerations 1515
understanding 1511
understanding regional settings 1513
SessionID property 1602
Set keyword 176
Set language construct 190
Set method 233
SetActualEndDate method 380
SetActualStartDate method 380
SetActualTaskBarColor method 381
SetAggregateMapping method 91
SetAnalyticInstance method 134

Copyright © 1988-2007, Oracle. All rights reserved.

SetAttachmentContent method 648
SetAttachmentProperty method 809
SetBurstFieldCriteria method 1854
SetCaseSensitive method 1854
SetChartArea method 381
SetColorArray methods 361
SetComponentChanged property
  Field class 515
  Rowset class 1451
SetContentType method 609
SetCursorPos method 498
SetData method 37, 346
SetDataAnnotations method 375
SetDataColor methods 362
SetDataGlyphScale 375
SetDataHints method 347
SetDataSeries method 350
SetDataURLs method 351
SetDataXAxis method 351
SetDataYAxis method 352
SetDateCriteria method 1855
SetDayFormat method 399
SetDefault method
  Field class 499
  Field class methods 498
  Record class 1398
  Rowset class 1442
SetDimensionAggregate method 83
SetDimensionOrder method 37
SetDimFilter method 38
SetDimSort method
  CubeCollection class 38
  CubeCollectionDefn class 92
SetEditTable method
  Message class 762
  Message class methods 749
  Record class 1398
  Record class methods 1387
SetEmailOption method 1212
SetErrorMessage method 797
SetFieldMapping method 93
SetFileId method 540
SetFileLayout method 542
SetFilter method 93
SetFolderCriteria method 1855
SetHeader method 609
SetHourFormat method 398
SetID property 1688
SetImportMode method 1673
SetItemFolder method 1213
SetLayout method 135
SetLegend method 370
SetMCFEmail method 664
SetMinuteFormat method 398
SetModified method 850
SetMonthFormat method 400
SetOLData method 372
SetOLDataAnnotations 375
SetOLDataGlyphScale 376
SetOLDataSeries method 372
SetOLDataXAxis method 373
SetOLDataYAxis method 373
SetOutputOption method 1214
SetOutputParmDate method 972
SetOutputParmDateArray method 973
SetOutputParmDateTime method 973
SetOutputParmDateTimeArray
 method 974
SetOutputParmInt method 975
SetOutputParmIntArray method 976
SetOutputParmNumber method 974
SetOutputParmNumberArray
 method 975
SetOutputParmString method 976
SetOutputParmStringArray method 977
SetOutputParmTime method 977
SetOutputParmTimeArray method 978
SetParameter method 478
SetPersistAggregate method 94
SetPlannedEndDate method 382
SetPlannedStartDate method 382
SetPlannedTaskBarColor method 383
SetPosition method 525, 543
SetProcessInstanceCriteria method 1856
SetProperty method 1891
SetPropertyByName method
  Component Interface class 439
  Mobile class 851
SetPSQueryPromptRecord method 1825
SetQueryString method 763
SetRecTerminator method 544
SetReportIDCriteria method 1856
SetRule method
  cube class 83
  UserFunctionDefn class 97
SetRuntimeDataRowset method 1826
SetRuntimeDataXMLDoc method 1826
SetRuntimeDataXMLFile method 1827
SetRuntimeProperties method 1828
SetSecondFormat method 397

Copyright © 1988-2007, Oracle. All rights reserved.

SetSelectedField method 149
SetSlice method 39
SetSMTPParam method
  MCFOutboundEmail class 693
  SMTPSession 708
SetSMTPParam property 709
SetSQL method 8
SetStatus method 764, 798
SetSubstitution method 486
SetTableXScrollbar method 383
SetTaskAppData method 384
SetTaskAppDataTitles method 384
SetTaskBarURL method 385
SetTaskData method 385
SetTaskDependencyChildID method 386
SetTaskDependencyData method 386
SetTaskDependencyParentID method 388
SetTaskDependencyType method 387
SetTaskDependencyURL method 388
SetTaskExpanded method 389
SetTaskHints method 389
SetTaskID method 390
SetTaskLabel method 390
SetTaskLevel method 390
SetTaskMilestone method 391
SetTaskName method 392
SetTaskProgress method 392
SetTaskProgressBarColor method 393
SetTemplate method 9
SetTraceLevel method 957
SetupCompVarsAndRcpts method 892
SetupGenericVars method 893
SetUserIdCriteria method 1857
SetVariableBounds method 986
SetVariableType method 988
SetWBSNumbering method 393
SetXAxisLabels methods 362
SetXmlDoc method 765
SetYAxisLabels methods 363
SetYearFormat method 400
Severity property 1475
Shift method 234
shortcut considerations
  Field class 491
  Grid class 576
  Page class 991
  Record class 1377
  Row class 1408
  Rowset class 1422
Shortname property 1360
ShortName property 1310
ShortTranslateValue property 516
ShowAllRows method 1443
ShowCrossHair property 376
ShowGridLines property 136
ShowHierarchy method 40
ShowRequiredCue property 871
ShowRequiredFieldCue property 517
ShowTaskLabels property 395
ShutDown method 959
signon
  authenticating single signon 599
  ByPassSignOn property 600
  PeopleSoft sessions 1512
Simple Object Access Protocol (SOAP), *See* SOAP
single point of data 327
Size property
  MCFInboundEmail class 675
  Message class 781
SkillLevel property 1723
SlicerVisible property 136
SmartZero property 517
SMTPPort property 703
SMTPServer property 703
SMTPSession class
  constructor 644
  methods 706
    See Also SMTPSession class methods
SMTPSession class methods
  SendAll 707
  SetSMTPParam 708
  ValidUnsentAddresses 707
SMTPSession class properties
  BackupServer 709
  BackupUserName 709
  BackupUserPassword 709
  IsAuthenticationReqd 709
  Port 710
  Server 710
  SetSMTPParam 709
  UsedDefaultConfig 710
  UsedPrimaryServer 710
  UserName 711
  UserPassword 711
SMTPUserName property 704
SMTPUserPassword property 704
SOAP
  messages 1540
  SOAPDoc class 1539

Copyright © 1988-2007, Oracle. All rights reserved.

See Also SOAPDoc class
SOAPDoc objects 1542
See Also SOAPDoc objects
understanding 1539
validating documents 1556
SOAP messages 1540
SOAPDoc
class 1539
See Also SOAPDoc class
objects 1542
See Also SOAPDoc objects
SOAPDoc class
creating documents 1560
examples 1560
list of functions, methods, properties, returns 1985
methods 1546
See Also SOAPDoc class methods
properties 1556
See Also SOAPDoc class properties
reading documents 1561
understanding 1539, 1542
using SOAP XML text 1561
SOAPDoc class methods
AddBody 1546
AddEnvelope 1547
AddFault 1550
AddHeader 1551
AddMethod 1552
AddParm 1554
GetParmName 1554
GetParmValue 1555
ValidateSOAPDoc 1545, 1556
SOAPDoc class properties
BodyNode 1557
EnvelopeNode 1557
FaultCode 1557
FaultCodeS 1557
FaultString 1558
HeaderNode 1558
MethodName 1559
MethodNode 1559
ParmCount 1559
understanding 1556
XmlDoc 1560
SOAPDoc objects
accessing sections 1544
body section 1543
creating 1544
declaring/scope 1546
envelope section 1543
fault section 1545
header section 1542
method section 1543
understanding 1542
using 1545
using XmlDoc/XmlNode methods/properties 1750
validating 1545
Solve method 989
solvers
activating 979
deactivating 980
Sort method
Array class 234
QueryDBRecordField collection 1356
Rowset class 1443
sorting
records 1428
Rowset class Sort method 1443
SortSpecifications property 1473
Source property 1526
SourceComponent property 889
SourceMarket property 889
SourceMenu property 889
SourceNode property 817
SourceReportPath property 1191
spidering
FSYS search indexes 1502
HTTP search indexes 1499
Search Start Options collection 1504
using for mobile devices 1591
SQL
binding/executing statements 1565
case sensitivity 1564
class 1563
See Also SQL class
definitions 1563
See Also SQL definitions
fetching from a select 1566
objects 1563
See Also SQL objects
opening/processing 1576
Query classes 1233
See Also Query classes
reusing a cursor 1567
SetSQL method 8
SQL property 1282
SQL class
application engine considerations 1570

Copyright © 1988-2007, Oracle. All rights reserved.

global declaration considerations 1570
list of functions, methods, properties, returns 1987
methods 1572
See Also SQL class methods
properties 1577
See Also SQL class properties
ReuseCursor property 1568
reusing a cursor 1567
understanding 1563
understanding SELECT styles 1567
using “Array of Any” 1566
SQL class methods
Close 1572
Execute 1572
Fetch 1574
Open 1576
SQL class properties
BulkMode 1577
IsOpen 1579
LTrim 1579
ReuseCursor 1580
RowsAffected 1580
Status 1581
TraceName 1581
Value 1582
SQL definitions
creating 1565
understanding 1563
SQL objects
declaring/scope 1571
reusing cursors 1567
understanding 1563
SQL property 1282
SQLStatement property 1536
SQLStatementVariables property 1537
SqlText property 517
SSBs property 1537
StackTrace Exception class property 488
StartFromPageNum property 1884
StartNum property 1885
StartOpts property
Search FS Options class 1503
Search HTTP Options class 1502
State property 1652
statements 483
states
Business Interlink objects 260
managing 618
returning to users 1515
status
building status response messages 932
checking for optimization engines 923, 942
ConflictStatus property 1600
email 635
LockStatus property 1684
MCFGetMail error codes 642
ProcessRequest class Status property 1225
RunStatus property 1224
sending for optimization 926
SQL class Status property 1581
Tree class Status property 1688
writing status messages to optimization engine logs 981
Status property
MCFGetMail class 668
MCFInboundEmail class 675
ProcessRequest class 1225
ReportDefn class 1833
SQL class 1581
TemplateFile class 1841
TranslationFile class 1842
Tree class 1688
StatusNotifyOptions property 704
StatusNotifyReturn property 704
StopAtError property 316
StopOnFirstError property 443
StorageType property 1079
StoredFormat property 518
StoreEmail method 679
Structure property 1688
structured messages 729
StructureName property 1688
Style property
Chart class appearance 358
Field class 518
Row class 1418
style sheets
default classes for charts 332
using for charts 331
styles, iScript 590
StyleSheet property
Chart class appearance 358
TabDefinition class 1119
stylesheets, iScript 590
Subarray method 236
subclasses
downcasting 179

Copyright © 1988-2007, Oracle. All rights reserved.

overriding superclass properties 175
understanding 153, 156
using instance variables 178
using private methods 160
using protected methods 157
Subject property 656
MCFOutboundEmail class 705
Notification class 889
NotificationTemplate class 895
SubName property 782
SubQueueName property 782
subscription
subscribing to partial records 735
subscription PeopleCode 1512
SubscriptionProcessId property 782
Substitute method 237
SubstitutionCount Exception class property 489
SubType property 688
subtyping 156
suggestions, submitting cxviii
SummaryLevelNumber property 1702
SummarySetId property 1702
SummaryTreeName property 1703
SummaryUserKeyValue property 1703
superclasses
referencing 178, 185
understanding 156
SuspendFormatting property 1521
SwitchLevel method 1645
SybBindInfo property 1537
SybFetchInfo property 1537
synchronization
considerations 1605
events 1585
See Also synchronization server events
filtering 1594
implementing custom data distribution 1590
Mobile Agent 828
synchronization server events
understanding 1585
using 1588, 1589
Synchronizing property 1603
synchronous mode
loading analytic instances 917
running optimization transactions 918
running transactions 924
understanding the RunSynch method 956
syncid property 906
SyncIDs property 1603
SyncOrder property 151
SyncRequest method
IntBroker class 799
message class 766
SyncServer
class 1585
See Also SyncServer class
objects 1587
See Also SyncServer objects
SyncServer class
list of methods, properties, returns, variables 1988
methods 1595
See Also SyncServer class methods
properties 1599
See Also SyncServer class properties
understanding 1585, 1588, 1595
SyncServer class methods
AddReference 1595
AddReferenceType 1595
CheckForChanges 1596
Notify 1598
SelectAll 1599
SyncServer class properties
ClientPlatform 1599
ConflictAlgorithm 1600
ConflictStatus 1600
ExcludeProperty 1600
ExportLocation 1601
IsReferenceUnresolved 1601
LastSyncDateTime 1601
LastSyncRowCount 1602
PropertyValue 1602
SessionID 1602
Synchronizing 1603
SyncIDs 1603
TypeID 1603
validateID 1604
validateRowCount 1604
validateVersion 1604
SyncServer objects
accessing/declaring/scope 1594
understanding 1587, 1595
%SyncServer system variable 1587, 1595
SyncServiceTimeout property 818
SysCon class 620
system variables
iScript classes 1939

Copyright © 1988-2007, Oracle. All rights reserved.

mapping deprecated to new 2088
%Request 593
%Response 593
%Super 178
supported in mobile PeopleCode 876
%SyncServer 1587, 1595
SysVar Java class 620
%This 171, 173
%ThisMobileObject 831
SystemIDFieldName property 519
SysVar class 620

T

TabDefinition
class 1113
See Also TabDefinition class
collection 1119
See Also TabDefinition collection
TabDefinitions property 1044
TabDefinition class
list of methods, properties, returns 2045
methods 1113
properties 1114
See Also TabDefinition class properties
understanding 1113
TabDefinition class properties
AssignedPagelets 1114
Attributes 1115
Author 1115
AuthorAccess 1115
AvailableCategories 1114
AvailablePagelets 1115
ColumnLayout 1115
CreationDate 1116
Description 1116
DynamicCategories 1116
HelpID 1116
HtmlText 1116
IsHideActionBar 1116
IsLayoutLocked 1117
IsRenameable 1117
Label 1117
LayoutBehavior 1117
Name 1118
OwnerId 1118
Product 1118
PublicAccess 1118
QualifiedURL 1118
SequenceNumber 1119
StyleSheet 1119
ValidFrom 1119
ValidTo 1119
TabDefinition collection
list of methods, properties, returns 2047
methods 1120
See Also TabDefinition collection methods
properties 1122
understanding 1119
TabDefinition collection methods
DeleteItem 1120
First 1120
InsertItem 1121
ItemByName 1121
Next 1122
TabDefinition collection properties 1122
TabDefinitions property 1044
tables
flat table methods 255
INSTALLATION 521
OPTIONS 521
PS_MESSAGE_LOG 921
TabName property 1157
target content references 1013
Task class
list of methods, properties, returns 1993
methods 1733
See Also Task class methods
properties 1737
See Also Task class properties
understanding 1733
Task class methods
Close 1733
Enqueue 1734
Refresh 1735
RefreshStatus 1736
Task class properties
AgentID 1737
ApplicationData 1737
Comments 1737
EnqueueTime 1737
EscalationTime 1737
Language 1737
OriginalTime 1738
OverFlowTime 1738
PhysicalQueueID 1738
TiedAgentID 1738
Task constructor 1714
Task property 1739
TaskDependencyLineType property 395

Copyright © 1988-2007, Oracle. All rights reserved.

TaskList class
list of methods, properties, returns 1994
methods 1738
properties 1739
See Also TaskList class properties
understanding 1738
TaskList class properties
AgentID 1739
PhysicalQueueID 1739
Task 1739
TaskType 1740
Total 1740
TaskList constructor 1715
TaskMilestoneGlyph property 396
tasks
Task class 1733
See Also Task class
TaskList class 1738
See Also TaskList class
TaskList class Task property 1739
TaskTitle property 396
TaskType property 1740
template content references 1013
Template property
Content Reference class 1080
Content Reference Links class 1109
Notification class 889
TemplateControlFieldName property 1833
TemplateDefn class
constructor 1811
methods 1834
TemplateDefn class properties 1836
ObjectOwnerID 1837
TemplateDefn methods
Get 1834
GetActiveTemplateFile 1834
GetTemplateFile 1835
TemplateDefn properties
Description 1836
DistributionChannel 1836
IsReadOnly 1836
IsSubTemplate 1837
LanguageCode 1837
LastUpdateDTTM 1837
LastUpdateOprID 1837
RegisteredBy 1837
RegisteredDTTM 1837
ReportCategoryID 1838
Type 1838
TemplateFile class
constructor 1812
methods 1838
TemplateFile class properties 1840
TemplateFile methods
GetFile 1838
GetMapFile 1839
GetTranslationFile 1839
GetTranslationFileList 1840
TemplateFile properties
EffectiveDate 1840
FileName 1840
MapFileName 1840
Status 1841
TemplateId property 895
TemplateObject property
Content Reference class 1081
Content Reference Links class 1109
NodeTemplate class 1056
PortalRegistry class 1044
templates
content references 1013
deleting 1017
editing OPT_CALL 931
editing PT_OPTCALL 927
NodeTemplate class 1056
See Also NodeTemplate class
NotificationTemplate class 891
See Also NotificationTemplate class
SetTemplate method 9
Template property 1080
Template property (Content Reference Links class) 1109
See Also Template property
TemplateObject property 1081
TemplateObject property (Content Reference Links class) 1109
See Also TemplateObject property
TemplateType property 1081
TemplateType property (Content Reference Links class) 1109
See Also TemplateType property
TemplateType property
Content Reference class 1015, 1081
Content Reference Links class 1109
NotificationTemplate class 895
Terminate method 20
terms 2101
Text property
MCFBodyPart class 653

Copyright © 1988-2007, Oracle. All rights reserved.

MCFEmail class 656
MCFOutboundEmail class 705
NotificationTemplate class 895
PSMessages class 1528
QueryExpression class 1333
Watermark class 1897
TextAngle property 1897
TextFontName property 1897
TextFontSize property 1897
TextStartPosX property 1898
TextStartPosY property 1898
third-party integrators 194
%ThisMobileObject system variable 831
TiedAgentID property 1738
timedout property 906
Timeout property 605
timeouts
Component Interface 419
loading analytic instances 917
running optimization transactions 918
Timeout property 605
TimeStampFieldName property 520
TimeToWaitForResult property 705
TimeZone property 1226
ToolsRelease property 1049
TopRowNumber property 1451
ToString method 486
Total property 1740
TotalAgents property 1733
TotalMemberName property 75
TotalPhysicalQueues property 1722
Trace Setting class properties
API 1533
COBOLStmtTimings 1533
ConnDisRollbackCommit 1533
DBSpecificCalls 1533
ManagerInfo 1533
NetworkServices 1534
NonSSBs 1534
OutputUNICODE 1534
PCExtFcnCalls 1534
PCFcnReturnValues 1534
PCFetchedValues 1535
PCIntFcnCalls 1535
PCListProgram 1535
PCParameterValues 1535
PCProgramStatements 1535
PCStack 1535
PCStartOfPrograms 1536
PCTraceProgram 1536
PCVariableAssignments 1536
RowFetch 1536
SQLStatement 1536
SQLStatementVariables 1537
SSBs 1537
SybBindInfo 1537
SybFetchInfo 1537
TraceFile 1537
Trace Settings
class 1532
See Also Trace Setting class properties
objects 1514
TraceFile property 1537
TraceName property 1581
TraceSettings class 2014
TraceSettings objects 1514
TraceSettings property 1521
tracing
getting for request messages 925
mobile PeopleCode 835, 837
setting for request messages 924
setting options 1514
See Also Trace Settings
TraceName property 1581
transactionid property
WorklistEntry class 907
TransactionId property 783
TransactionID property 818
WSWorklistEntry class 909
transfer functions 483
transfers 422
TransformData class 1990
Translatable property 1088
TranslateEffDtLogic property 1311
TranslateExpression property 1311
TranslateField property 1311
TranslateOption property 1311
translation 333
TranslationFile class
constructor 1813
method 1841
TranslationFile method
GetFile 1841
TranslationFile properties 1842
Description 1842
FileName 1842
LanguageCode 1842
Status property 1842
Tree class
list of methods, properties, returns 2080

Copyright © 1988-2007, Oracle. All rights reserved.

methods 1654
See Also Tree class methods
properties 1676
See Also Tree class properties
understanding 1653
Tree class methods
Audit 1654
AuditByName 1655
Close 1656
Copy 1656
Create 1658
Delete 1659
Exists 1660
FindLeaf 1661
FindNode 1661
FindRoot 1662
InsertRoot 1663
LeafExists 1663
LockTree 1664
NodeExists 1668
Open 1665
OpenWholeTree 1667
Rename 1668
Save 1669
SaveAs 1670
SaveAsDraft 1671
SaveDraft 1672
SetImportMode 1673
TreeLocksNumber 1673
UnlockTree 1674
UpdateLock 1675
Tree class properties
AllValues 1676
AuditDetails 1676
Branches 1676
BranchImageName 1676
BranchLevel 1677
BranchName 1677
Category 1677
Description 1677
DuplicateLeaves 1678
EffDt 1678
HasDetailRanges 1678
HasLockedBranches 1679
IsBranched 1679
IsChanged 1679
IsOpen 1680
IsQueryTree 1680
IsValid 1680
IsVersionChanged 1680
IsWholeTree 1681
KeyBranchName 1681
KeyEffDt 1681
KeyName 1681
KeySetId 1682
KeyUserKeyValue 1682
LeafCount 1682
LeafImageName 1682
LeafOnClipboard 1682
LevelCount 1683
Levels 1683
LevelUse 1683
LockOwner 1684
LockStatus 1684
Name 1684
NodeColImageName 1685
NodeCount 1685
NodeExpImageName 1685
NodeOnClipboard 1685
ParentLevel 1686
ParentName 1686
PerformanceMethod 1686
PerformanceSelector 1687
PerformanceSelectorOption 1687
SetID 1688
Status 1688
Structure 1688
StructureName 1688
TreeImageName 1689
UserKeyValue 1689
UseUpdateReservation 1689
Tree classes
handling errors 1612
implementing 1615
instantiating 1617
Leaf 1619
See Also Leaf class
Level 1632
See Also Level class
list of methods, properties, returns 2074
Node 1634
See Also Node class
relationship between 1610
Tree 1653
See Also Tree class
Tree Structure 1690
See Also Tree Structure class
understanding 1609
verifying leaf/node insertions 1613
Tree collections

Copyright © 1988-2007, Oracle. All rights reserved.

Branch 1618
See Also Branch collection
Level 1629
See Also Level collection
understanding 1612
Tree Manager 1609
tree objects
declaring/scope 1614
getting/opening 1610
understanding 1609, 1653
Tree Structure class
list of methods, properties, returns 2083
methods 1690
See Also Tree Structure class methods
properties 1693
See Also Tree Structure class properties
understanding 1690
Tree Structure class methods
Close 1690
Copy 1690
Create 1691
Delete 1691
Open 1692
Rename 1692
Save 1693
Tree Structure class properties
Description 1693
DetailComponent 1694
DetailField 1694
DetailMenu 1694
DetailMenuBar 1695
DetailMenuItem 1695
DetailMultiNavigate 1695
DetailPage 1695
DetailRecord 1696
IndirectionMethod 1696
KeyName 1698
LevelComponent 1698
LevelMenu 1698
LevelMenuBar 1698
LevelMenuItem 1698
LevelPage 1699
Name 1699
NodeComponent 1700
NodeField 1700
NodeMenu 1700
NodeMenuBar 1700
NodeMenuItem 1701
NodeMultiNavigate 1701
NodePage 1701
NodeRecord 1701
NodeUserKeyField 1702
SummaryLevelNumber 1702
SummarySetId 1702
SummaryTreeName 1703
SummaryUserKeyValue 1703
Type 1703
tree structure objects 1690
tree structures
class 1690
See Also Tree Structure class
objects 1690
portal registry 997
See Also portal registry
relationships between tree classes 1610
understanding 1609
TreeBranchName property
Leaf class 1629
Level class 1633
Node class 1652
TreeEffDt property
Leaf class 1629
Level class 1633
Node class 1653
TreeImageName property 1689
TreeLocksNumber method 1673
TreeName property
Leaf class 1629
Level class 1633
Node class 1653
trees
analytic calculation engine class 12
AnalyticModel class
accessing 27
attaching 22
classes 1609
See Also Tree classes
collections 1612
See Also Tree collections
creating 1615
detaching from dimensions 24
objects 1609
See Also tree objects
structures 1609
See Also tree structures
traversing hierarchies 1703
TreeSetId property
Leaf class 1629
Level class 1634
Node class 1653

Copyright © 1988-2007, Oracle. All rights reserved.

TreeUserKeyValue property
Leaf class 1629
Level class 1634
Node class 1653
TrueXY line chart 377
try-catch statements 483
type checking 156
Type property
AnalyticTypeModelDefn class 148
Chart class appearance 359
comparison class 107
CompIntfPropInfoCollection objects 457
constant class 109
DataSourceDefn class 1817
Field class 520
FunctionCall class 118
MethodInfo 210
Node class 1653
Operation class 121
PropertyInfo 212
Query class 1282
QueryDBRecordField class 1360
QueryExpression class 1334
QueryField class 1312
SearchIndex class 1485
TemplateDefn class 1838
Tree Structure class 1703
variable class 123
TypeID property 1603
typographic conventions cxxi
typographical conventions cxvi

U

UID property 676
Unbranch method 1645
Uniform Resource Identifier (URI) 587
Uniform Resource Locator (URL), *See* URL
UniquePromptName property 1367
universal queue classes
Agent class 1717
See Also Agent class
AgentPhysQueueProps class 1722, 1723
See Also AgentPhysQueueProps class; AgentPhysQueueTasks class
Broadcast class 1725
constructors 1710
See Also universal queue classes constructors
importing 1709
list of constructors, methods, properties, returns 1991
LogicalQueue class 1727
See Also LogicalQueue class
MCFFactory class 1728
See Also MCFFactory class
PhysicalQueue class 1728
See Also PhysicalQueue class
scope 1708
Task class 1733
See Also Task class
TaskList class 1738
See Also TaskList class
understanding 1707
Util class 1740
See Also Util class
Universal Queue Classes
example 1742
universal queue classes constructors
Agent 1710
AgentPhysQueueProps 1711
AgentPhysQueueTasks 1711
Broadcast 1712
LogicalQueue 1712
MCFFactory 1713
PhysicalQueue 1713
Task 1714
TaskList 1715
Util 1716
universal queue objects 1710
universal queues
classes 1707
See Also universal queue classes
objects 1710
Unload method 20
UnlockTree method 1674
UnsetDimFilter 41
UnsetDimSort method 41
UnsetSelectedField method 150
Unshift method 238
unstructured messages
understanding 729
using XmlDoc messages 1545
Update method
IntBroker class 801
Message class 767
Record class 1400

Copyright © 1988-2007, Oracle. All rights reserved.

WorklistEntry class 902
Update mode 525
UpdateData method 479
UpdateLock method 1675
UpdateRanges method 1625
UpdateRunStatus method 1215
UpdateSegment method 768
UpdateXmlDoc method 801
URI 587
URL
assembling to view iScripts 590
Favorite class QualifiedURL property 1165
Favorite class URL property 1166
generating for portal content 591
Pagelet class QualifiedURL property 1150
Pagelet class URL property 1150
Pagelet class URLType property 1151
understanding 587
url property 907
URL property 1081
URL property (Content Reference Links class) 1110
See Also URL property
URLType property 1015, 1082
URLType property (Content Reference Links class) 1110
See Also URLType property
UserTab class QualifiedURL property 1157
URL property
Content Reference class 1081
Content Reference Links class 1110
Favorite class 1166
Pagelet class 1150
using 1016
WorklistEntry class 907
URLType property
Content Reference class 1015, 1082
Content Reference Links class 1110
Pagelet class 1151
Usage property 213
UsageType property
Content Reference class 1013, 1083
Content Reference Links class 1110
UseBurstValueAsOutputFileName property 1834
UseCount property 1367
UsedDefaultConfig property
MCFOutboundEmail class 705
SMTPSession class 710
UsedPrimaryServer property
MCFOutboundEmail class 706
SMTPSession class 710
UseLocalTime property 1531
user functions
accessing 65
names 66
rules 97
adding to analytic model 51
aggregation 75
class properties 98
commenting 98
copying 54
deleting 59
naming 98
renaming 70
setting
rules 97
User property 676
UserFunctionDefn class methods 97
UserFunctionDefn class properties 98
UserHomepage
class 1154
See Also UserHomepage class
UserHomepage class
list of methods, properties, returns 2053
methods 1155
properties 1155
understanding 1154
UserHomepage class methods 1155
UserHomepage class properties 1155
UserId property 1156
UserID property 668
UserKeyValue property 1689
UserName property 711, 818
UserPassword property 711
UsersPart method 100
UserTab 1156
class 1156
See Also UserTab class
collection 1158
See Also UserTab collection
property 1156
UserTab class
list of properties, returns 2054
properties 1156
See Also UserTab class properties
understanding 1156

Copyright © 1988-2007, Oracle. All rights reserved.

UserTab class properties
ColumnLayout 1156
Label 1157
QualifiedURL 1157
SelectedPagelets 1157
SequenceNumber 1157
TabName 1157
UserTab collection
list of methods, properties, returns 2054
methods 1158
See Also UserTab collection methods
properties 1160
understanding 1158
UserTab collection methods
DeleteItem 1158
First 1158
InsertItem 1159
ItemByName 1159
Next 1160
UserTab property 1156
UsesCube method 95
UsesDimension method
CubeCollectionDefn class 95
CubeDefn class 84
UseSimpleURL method 610
UseSpaceForNull property 554
UseUpdateReservation property 1689
Util class 1740
Util constructor 1716
utility classes 184

V

Validate method 72
ValidateAddress method 685
ValidateEnum method 851
validateID property 1604
validateRowCount property 1604
ValidateSOAPDoc method 1545, 1556
validateVersion property 1604
ValidFrom property
Content Reference class 1083
Content Reference Links class 1111
Folder class 1066
TabDefinition class 1119
using 1004
ValidSentAddresses property 706
ValidTo property
Content Reference class 1083
Content Reference Links class 1111
Folder class 1066
TabDefinition class 1119
using 1004
ValidUnsentAddresses method 707
ValidUnsentAddresses property 706
Value property
AttributeValue class 1089
constant class 109
Cookie class 613
Field class 521
Query Metadata class 1344
QueryListValue class 1337
Search Start Options class 1507
SearchField class 1481
SQL class 1582
using character values 521
using INSTALLATION/OPTIONS tables 521
ValueDimensionName property 85
ValuesDifferFromLastSync method 852
variable class
creating 122
method 122
Name property 123
properties 123
Type property 123
Variable property 105
variables
array variables 216
declaring private instance 160, 178
See Also instance variables
fetching data from a select 1566
JavaObject 618
overriding 178
PCVariableAssignments property 1536
placing declarations in application classes 177
SetVariableBounds method 986
SetVariableType method 988
SQLStatementVariables property 1537
supporting for mobile PeopleCode 838
%Synch and %Asynch 934
system 171
See Also system variables
typographic conventions cxxi
using component variables 418
using in application classes 177, 178
See Also global variables
using local with PortalRegistry 1019
See Also local variables
VeggieKey property 1487

Copyright © 1988-2007, Oracle. All rights reserved.

Verified property 480
Version property 783
Visible property
Field class 522
GridColumn class 582
ListViewAttrs class 868
Page class 995
PeerDefaultAttributes class 875
PropertyAttrs class 872
Row class 1419
VisitedNodes property 819
Visual Basic
accessing the API Repository 199
handling errors 1513
inserting data in component interfaces 468
referencing PortalRegistry objects 1180
visual cues cxvii

W

WarningCount property 1475
WarningPending property 1521
warnings cxvii
WasSaved method 852
Watermark class
constructor 1868
properties 1896
Watermark class properties 1897
ImageFile 1896
ImageFileLowerLeftX 1896
ImageFileLowerLeftY 1896
ImageFileUpperRightX 1896
ImageFileUpperRightY 1897
PageIndex 1897
Text 1897
TextAngle 1897
TextFontName 1897
TextStartPosX 1898
TextStartPosY 1898
Watermark property 1890
Web
creating libraries 587
setting attributes 1201
web services
WSWorklistEntry class 908
Width property 359
WinMessage function 921
wl_priority property 907
wldaystoselect property 907
wldaystowork property 907
work records 333
Workflow Notification classes, *See* Notification classes
Worklist class
list of methods, properties, returns 1968
methods 896
understanding 895
Worklist constructor 883
WorklistEntry
creating 910
updating 911
WorklistEntry class
list of methods, properties, returns 1968
methods 896
See Also WorklistEntry class methods
properties 903
See Also WorklistEntry class properties
understanding 896
WorklistEntry class methods
Create 897
GetResponseStatus 898
Reassign 898
Save 899
SaveWithCustomData 900
SelectByKey 901
SelectByMessageId 902
Update 902
WorklistEntry class properties
actiondttm 903
busactivity 903
buseventname 903
busprocname 903
commentshort 903
do_replicate_flag 904
instanceid 904
instselecteddttm 904
inststatus 904
insttimeoutddttm 905
instworkeddttm 905
IsCreatedViaWebService 905
oprid 905
originatorid 905
prevoprid 905
requestmessageid 906
ResponseStatus 906
syncid 906
timedout 906
transactionid 907
url 907

Copyright © 1988-2007, Oracle. All rights reserved.

wl_priority 907
wldaystoselect 907
wldaystowork 907
worklistname 908
WorklistEntry constructor 884
worklistname property 908
WorkLoad property 1723
World Wide Web Consortium (W3C) 1539
Write method 610
Writeable property 151
WriteBlobToFile method 853
WriteLine method
File class 545
Response class 611
WriteRaw method 546
WriteRecord method
File class 546
File Layout example 558
WriteRowset method
File class 548
File Layout example 561
WriteString method 550
WSA_MessageID property 819
WSA_ReplyTo property 819
WSWorklistEntry class
list of methods 1970
list of properties 1970
properties 909
understanding 908
WSWorklistEntry class methods
OnError 908
OnNotify 909
WSWorklistEntry class properties
InstanceID 909
TransactionID 909
WSWorklistEntry classes
worklist entries 908
See Also WSWorklistEntry class
WSWorklistEntry constructor 885

X

X-Priority 633
XAxisCross property 364
XAxisCrossPoint property 377
XAxisLabelOrient property 365
XAxisMax property 377
XAxisMin 377
XAxisPosition property 396
XAxisPrecision property 377
XAxisScaleResolution property 378
XAxisStyle property 365
XAxisTicks property 366
XAxisTitle property 368
XAxisTitleOrient property 368
XAxisTitleStyle property 369
Xlat property 458
XML
building responses 259
defining namespaces 1747
parsing requests 259
SOAP 1539
See Also SOAP
understanding 1745
using with file definitions 547, 549
using with ReadRowset method 539
XML generation class
constructors 1858
QueryDS class 1859
RowsetDS class 1861
XML Publisher
understanding 1801
XML Publisher classes 1801
creating objects 1809
error handling 1808
example 1898
generating reports 1898
hierarchies 1804
importing 1808
life-cycle 1803
list of constructors, methods, properties, returns 1999
object data types 1808
publishing reports 1898
scoping 1808
search operator values 1809
terminology 1802
understanding 1801
XML Publisher engine class
constructors 1864
hierarchy 1806
XML Publisher engine class constructors
EFTProcessor 1864
FOProcessor 1864
FormProcessor 1865
FOUtility 1865
PageNumber 1866
PDFMapTool 1866
PDFMerger 1866
Properties 1867

Copyright © 1988-2007, Oracle. All rights reserved.

RTFProcessor 1867
Watermark 1868
XmlDoc
class 1745
See Also XmlDoc class
objects 1745
See Also XmlDoc objects
property 1560
XmlDoc class
list of functions, methods, properties, returns 1994
methods 1751
See Also XmlDoc class methods
properties 1764
SOAPDoc considerations 1542
understanding 1745
using 1746
XmlDoc class methods
CopyRowset 1751
CopyToPSFTMessage 1752
CopyToRowset 1754
CreateDocumentElement 1756
CreateDocumentType 1757
GenFormattedXmlString 1758
GenXmlString 1759
GetElementsByTagName 1759
LoadIBContent 1760
ParseXmlFromURL 1762
ParseXmlString 1763
XmlDoc objects
creating 1746
declaring/scope 1750
understanding 1745
using 1746
XmlDoc property 1560
XmlNode
class 1765
See Also XmlNode class
objects 1745
See Also XmlNode objects
XmlNode class
handling errors 1749
list of methods, properties, returns 1996
methods 1765
See Also XmlNode class methods
properties 1797
See Also XmlNode class properties
understanding 1749, 1765
XmlNode class methods
AddAttribute 1765
AddAttributeNS 1766
AddCDataSection 1767
AddComment 1768
AddElement 1769
AddElementNS 1770
AddEntityReference 1771
AddNode 1772
AddProcessInstruction 1773
AddText 1774
CopyNode 1774
FindNode 1776
FindNodes 1777
GenXmlString 1778
GetAttributeName 1778
GetAttributeValue 1779
GetCDataValue 1779
GetCDataValues 1780
GetChildNode 1781
GetElement 1781
GetElements 1782
GetElementsByTagName 1783
GetElementsByTagNameNS 1784
InsertCDataSection 1785
InsertComment 1786
InsertElement 1787
InsertElementNS 1789
InsertEntityReference 1790
InsertNode 1791
InsertProcessInstruction 1792
InsertText 1794
RemoveAllChildNode 1795
RemoveChildNode 1796
understanding 1765
XmlNode class properties
AttributesCount 1797
ChildNodeCount 1797
Index 1798
IsNull 1798
LocalName 1798
NamespaceURI 1798
NextSibling 1799
NodeName 1799
NodePath 1799
NodeType 1799
NodeValue 1800
ParentNode 1800
Prefix 1800
PreviousSibling 1800
XmlNode objects
accessing/traversing 1749

Copyright © 1988-2007, Oracle. All rights reserved.

declaring/scope 1750
understanding 1745, 1765
XMLPManager methods
GetDatasourceDefnList 1843
GetReportDefnList 1844
GetTemplateDefnList 1846

Y

Yahoo instant messaging 723
YAxisCrossPoint property 378
YAxisLabelOrient property 366
YAxisMax property 366, 379
YAxisMin property 366, 379
YAxisPosition property 397
YAxisPrecision property 379
YAxisScaleResolution property 379
YAxisStyle property 367
YAxisTicks property 367
YAxisTitle property 369
YAxisTitleOrient property 369
YAxisTitleStyle property 369
YesNo property 458

Z

ZoneOptions property 1488

Copyright © 1988-2007, Oracle. All rights reserved.