ORACLE
PeopleSoft

PeopleTools 8.59: PeopleCode Developer's Guide

July 2021

ORACLE

PeopleTools 8.59: PeopleCode Developer’s Guide
Copyright © 1988, 2021, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or “commercial computer software documentation” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government’s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

Preface: Preface ..... **xi**
Understanding the PeopleSoft Online Help and PeopleBooks xi
Hosted PeopleSoft Online Help xi
Locally Installed Help xi
Downloadable PeopleBook PDF Files xi
Common Help Documentation xi
Field and Control Definitions xii
Typographical Conventions xii
ISO Country and Currency Codes xii
Region and Industry Identifiers xiii
Translations and Embedded Help xiii
Using and Managing the PeopleSoft Online Help xiv
PeopleTools Related Links xiv
Contact Us xiv
Follow Us xiv
Chapter 1: Getting Started with PeopleCode **17**
PeopleCode Overview 17
Creating PeopleCode Programs 18
Chapter 2: Understanding the PeopleCode Language **21**
PeopleCode Language Structure 21
Data Types 21
Conventional Data Types 21
Object Data Types 23
Comments 24
Statements 25
Separators 25
Assignment Statements 25
Language Constructs 26
Branching Statements 27
Conditional Loops 29
Functions 29
Supported Functions 30
Function Definitions 30
Function Declarations 31
Function Calls 31
Function Return Values 32
Function Naming Conflicts 32
Expressions 32
Expression Fundamentals 33
Constants 33
Functions as Expressions 34
System Variables 35
Metastrings 35
Record Field References 35
Definition Name References 36
PeopleCode Reserved Words 37

Copyright © 1988, 2021, Oracle and/or its affiliates.

Variables ... 38
Supported Variable Types ... 39
User-Defined Variables ... 39
User-Defined Variable Declaration and Scope ... 40
Variable Declaration ... 41
User-Defined Variable Initialization ... 41
Restrictions on Variable Use ... 42
Scope of Local Variables ... 42
Duration of Local Variables ... 43
Variables and Functions ... 44
Recursive Functions ... 44
State of Shared Objects Using PeopleSoft Pure Internet Architecture ... 44
Operators ... 45
Math Operators ... 46
Operations on Dates and Times ... 46
String Concatenation ... 47
@ Operator ... 47
Comparison Operators ... 48
Boolean Operators ... 48
Chapter 3: Understanding Objects and Classes in PeopleCode ... 51
Classes and Objects ... 51
Classes ... 51
Objects ... 51
Object Instantiation ... 52
Creating and Using Objects ... 52
Instantiating Objects ... 52
Changing Properties ... 53
Invoking Methods ... 53
Copying Objects ... 55
Assigning Objects ... 55
Passing Objects ... 56
Chapter 4: Referencing Data in the Component Buffer ... 57
Understanding Component Buffer Structure and Contents ... 57
Component Buffer Contents ... 57
Rowsets and Scroll Areas ... 59
Record Fields and the Component Buffer ... 59
Specifying Data with Contextual References ... 60
Understanding Current Context ... 60
Using Contextual Row References ... 62
Using Contextual Buffer Field References ... 63
Specifying Data with References Using Scroll Path Syntax and Dot Notation ... 64
Understanding Scroll Paths ... 65
Structuring Scroll Path Syntax ... 65
Referencing Scroll Levels, Rows, and Buffer Fields ... 68
Chapter 5: Accessing the Data Buffer ... 75
Understanding Data Buffer Access ... 75
Data Buffer Access ... 75
Access Classes ... 75
Data Buffer Model and Data Access Classes ... 76
Understanding Data Buffer Classes Examples ... 76
Employee Checklist Page Structure ... 77

Copyright © 1988, 2021, Oracle and/or its affiliates.

Object Creation Examples....80
Data Buffer Hierarchy Examples....86
Rowset Examples....89
Hidden Work Scroll Example....90
Understanding Current Context....91
Accessing Secondary Component Buffer Data....93
Instantiating Rowsets Using Non-Component Buffer Data....93
Chapter 6: PeopleCode and the Component Processor....95
Understanding the Component Processor....95
Events Outside the Component Processor Flow....95
PeopleCode Program Triggers....95
Understanding PeopleCode Program Triggers....96
Accessing PeopleCode Programs....97
Execution Order of Events and PeopleCode....98
Component Processor Behavior....101
Component Processor Behavior from Page Start to Page Display....102
Component Behavior Following User Actions in the Component....103
Processing Sequences....104
Flow Charts....105
Default Processing....105
Search Processing in Update Modes....108
Search Processing in Add Modes....112
Component Build Processing in Update Modes....114
Row Select Processing....115
Component Build Processing in Add Modes....117
Field Change Processing....118
Row Insert Processing....121
Row Delete Processing....123
Buttons....125
Prompts....125
Pop-Up Menu Display....125
Selected Item Processing....126
Save Processing....126
PeopleSoft Pure Internet Architecture Processing Considerations....128
Deferred Processing Mode....129
PeopleCode Events....132
Activate Event....133
FieldChange Event....133
FieldDefault Event....134
FieldEdit Event....135
FieldFormula Event....135
ItemSelected Event....135
PostBuild Event....136
PreBuild Event....136
PrePopup Event....136
RowDelete Event....136
RowInit Event....137
RowInsert Event....138
RowSelect Event....139
SaveEdit Event....140
SavePostChange Event....141

Copyright © 1988, 2021, Oracle and/or its affiliates.

SavePreChange Event....141
SearchInit Event....141
SearchSave Event....142
Workflow Event....143
PeopleCode Execution in Pages with Multiple Scroll Areas....144
Chapter 7: PeopleCode and PeopleSoft Pure Internet Architecture....145
Considerations Using PeopleCode in PeopleSoft Pure Internet Architecture....145
Using PeopleCode with PeopleSoft Pure Internet Architecture....145
Using Internet Scripts....146
Using the Field Object Style Property....146
Using the HTML Area....147
Using HTML Definitions and the GetHTMLText Function....148
Using HTML Definitions and the GetJavaScriptURL Method....149
Using PeopleCode to Populate Key Fields in Search Dialog Boxes....149
Calling DLL Functions on the Application Server....149
Sample Cross-Platform External Test Function....150
Updating the Installation and PSOPTIONS Tables....151
Chapter 8: Using Methods and Built-In Functions....153
Understanding Restrictions on Method and Function Use....153
Think-Time Functions....153
WinMessage and MessageBox Functions....154
Program Execution with Fields Not in the Data Buffer....156
Errors and Warnings....156
DoSave Function....156
Record Class Database Methods....157
SQL Class Methods and Functions....157
Component Interface Restricted Functions....157
SearchInit PeopleCode Function Restrictions....158
CallAppEngine Function....158
ReturnToServer Function....159
GetPage Function....159
GetGrid and GetAnalyticGrid Functions....159
Publish Method....159
SyncRequest Method....160
Implementing Modal Transfers....160
Understanding Modal Transfers....160
Implementing Modal Transfers....162
Implementing the Multi-Row Insert Feature....163
Using the ImageReference Field....164
Inserting Rows Using PeopleCode....165
Using OLE Functions....166
Understanding OLE Functions....166
Using the Object Data Type....166
Sharing a Single Object Instance....166
Using the Exec and WinExec Functions....167
Using the Select and SelectNew Methods....167
Understanding the Select and SelectNew Methods....167
Using the Select Method....168
Using Standalone Rowsets....170
Understanding Standalone Rowsets....171
Using the Fill Method....171

Copyright © 1988, 2021, Oracle and/or its affiliates.

Using the CopyTo Method....171
Adding Child Rowsets....172
Using Standalone Rowsets to Write a File....173
Using Standalone Rowsets to Read a File....175
Using Errors and Warnings....175
Using Error and Warning Syntax....176
Using Errors and Warnings in Edit Events....176
Using Errors and Warnings in RowSelect Events....177
Using Errors and Warnings in RowDelete Events....177
Using Errors and Warnings in Other Events....178
Using the RemoteCall Feature....178
Understanding RemoteCall Components....178
Deciding Between RemoteCall and PeopleSoft Process Scheduler....181
Modifying PeopleSoft Process Scheduler Programs to Run with RemoteCall....181
Chapter 9: Using HTML Trees and the GenerateTree Function....183
Using the GenerateTree Function....183
Understanding HTML Trees....183
Building HTML Tree Pages....184
Using HTML Tree Rowset Records....185
Using HTML Tree Actions (Events)....187
Initializing HTML Trees....188
Processing Events Passed from a Tree to an Application....191
Adding Mouse-Over Ability to HTML Trees....195
Adding Visual Selection Node Indicators....196
Specifying Override Images....197
Chapter 10: Working With File Attachments....199
Understanding the File Attachment Functions....199
PeopleCode Built-in File Attachment Functions....199
Understanding the File Attachment Architecture....202
Understanding File Attachment Storage Locations....205
Understanding URL Strings Versus URL Objects....207
Developing Applications that Use File Attachment Functions....207
Application Development Process Overview....207
Delivered Record Definitions....209
Managing Entries in File Reference Tables....211
Using the PeopleTools Test Utilities Page....212
Application Development Considerations....213
File Name Considerations....213
Restrictions on Invoking Functions in Certain PeopleCode Events....214
Converting File Names for Files Uploaded by PutAttachment....214
Considerations When Using CopyAttachments....215
Application Deployment and System Configuration Considerations....215
File Attachment Functions in an Environment with Multiple Application Server Domains....216
Configuring the Web Server to Support Additional MIME Types....216
Restricting the File Types That Can Be Uploaded or Downloaded....217
Setting Up Virus Scanning....218
Using the HTML Sanitizer....220
Considerations When Attaching Text Files....224
File Attachment Chunk Size....225
Using Interfaces to the CopyAttachments and CleanAttachments Functions....225
Debugging File Attachment Problems....226

Copyright © 1988, 2021, Oracle and/or its affiliates.

Enabling Tracing on the Web Server or Application Server....226
Problems with Transfers to and from FTP Sites....227
Attachments with non-ASCII File Names....228
Problems Uploading Files....228
Problems Downloading Files....229
Passing Error Messages to the End User....229
Chapter 11: Accessing PeopleCode and Events....231
Understanding PeopleCode Programs and Events....231
Understanding Automatic Backup of PeopleCode....231
Accessing PeopleCode in Application Designer....232
Accessing Record Field PeopleCode....234
Understanding Record Field PeopleCode....234
Accessing Record Field PeopleCode from a Record Definition....235
Accessing Record Field PeopleCode from a Page Definition....235
Accessing Component Record Field PeopleCode....237
Understanding Component Record Field PeopleCode....237
Accessing Component Record Field PeopleCode....237
Accessing Component Record PeopleCode....238
Understanding Component Record PeopleCode....238
Accessing Component Record PeopleCode....239
Accessing Component PeopleCode....239
Understanding Component PeopleCode....239
Accessing Component PeopleCode....240
Accessing Page PeopleCode....240
Understanding Page PeopleCode....240
Accessing Page PeopleCode....240
Accessing Menu Item PeopleCode....241
Understanding Menu Item PeopleCode....241
Defining PeopleCode Pop-Up Menu Items....241
Accessing Menu Item PeopleCode....242
Copying PeopleCode with a Parent Definition....242
Upgrading PeopleCode Programs....242
Chapter 12: Using the PeopleCode Editor....243
Navigating Between PeopleCode Programs....243
Understanding the PeopleCode Editor Window....243
Navigating Between Programs Associated With a Definition and Its Children....244
Navigating Between Programs Associated With Events....244
Using the PeopleCode Editor....245
Understanding the PeopleCode Editor....246
Accessing PeopleCode Editor....247
Editing Functions....249
Adding Line Numbers....250
Find and Replace Dialogs....251
Go To Dialog....251
Validate Syntax Utility....251
Show Database Name Utility....252
Formatting Code Automatically....253
Using Drag-and-Drop Editing....254
Accessing PeopleCode External Functions....254
Accessing PeopleCode Application Packages and Application Classes....255
Accessing Definitions and Associated PeopleCode....257

Copyright © 1988, 2021, Oracle and/or its affiliates.

Accessing Help....257
Setting Up Help....258
Changing Colors in the PeopleCode Editor....258
Selecting a Font for the PeopleCode Editor....258
Changing Word Wrap in the PeopleCode Editor....259
Using the PeopleCode Event Properties....260
Using the Auto-Complete Feature....260
Viewing Event Mapping References....265
Generating PeopleCode Using Drag-and-Drop....266
Generating References to Definitions....267
Generating PeopleCode for a Business Interlink....267
Generating PeopleCode for a Component Interface....267
Generating PeopleCode for a File Layout....269
Chapter 13: Using the SQL Editor....271
Understanding the SQL Editor Window....271
Accessing the SQL Editor....271
Creating SQL Definitions....272
Accessing SQL Definition Properties....273
Creating Dynamic View or SQL View Records....273
Accessing the SQL Editor from Application Engine Programs....274
Using the SQL Editor....275
Chapter 14: Creating Application Packages and Classes....279
Understanding Application Packages....279
Creating Application Packages....280
Understanding Package Names....280
Creating Application Package Definitions....281
Using the Application Package Editor....282
Editing Application Classes....283
Chapter 15: Debugging Your Application....285
Understanding the PeopleCode Debugger....285
Accessing the PeopleCode Debugger....285
Using PeopleCode Debugger Features....286
Visible Current Line of Execution....287
Visible Breakpoints....287
Hover Inspect....287
Single Debugger....288
Variables Panes....288
Call Stack Pane....292
Setting Values for Variables and Properties....294
General Debugging Tips....295
Using PeopleCode Debugger Options....297
Setting Up the Debugging Environment....300
Compiling All PeopleCode Programs at Once....301
Compile All PeopleCode....302
Compile Project PeopleCode....303
Setting PeopleCode Debugger Log Options....303
Interpreting the PeopleCode Debugger Log File....306
Log File Contents....306
Other Items in the Log File....307
Using Application Logging....307
Setting the Application Log Fence in the Configuration File....308

Copyright © 1988, 2021, Oracle and/or its affiliates.

Using the Log Fence with PeopleSoft Analytic Calculation Engine....308
Using the Find In Feature....308
Finding References to Application Packages and Classes....314
Prerequisites....314
Finding Definition References....315
Using Cross-Reference Reports....316
Chapter 16: Improving Your PeopleCode....319
Reducing Trips to the Server....319
Counting Server Trips....320
Using Deferred Mode....320
Hiding and Disabling Fields....320
Using the Refresh Button....321
Updating Totals and Balances....321
Using Warning Messages....321
Using the Fastest Algorithm....322
Using Better Coding Techniques for Improved Performance....322
Running a SQL Trace....322
Optimizing SQL....322
Using the GetNextNumberWithGaps Function....323
Consolidating PeopleCode Programs....323
Moving PeopleCode to a Component or Page Definition....323
Sending Messages in the SavePostChange Event....323
Using Metadata and the RowsetCache Class....323
Setting MaxCacheMemory....324
Writing More Efficient Code....324
Writing More Efficient Code Examples....327
Preventing SQL Injection....331
Appendix A: PeopleCode Editor Short Cut Keys....333
Short Cut Keys in the PeopleCode Editor....333

Copyright © 1988, 2021, Oracle and/or its affiliates.

Preface

Understanding the PeopleSoft Online Help and PeopleBooks

The PeopleSoft Online Help is a website that enables you to view all help content for PeopleSoft applications and PeopleTools. The help provides standard navigation and full-text searching, as well as context-sensitive online help for PeopleSoft users.

Hosted PeopleSoft Online Help

You can access the hosted PeopleSoft Online Help on the Oracle Help Center. The hosted PeopleSoft Online Help is updated on a regular schedule, ensuring that you have access to the most current documentation. This reduces the need to view separate documentation posts for application maintenance on My Oracle Support. The hosted PeopleSoft Online Help is available in English only.

To configure the context-sensitive help for your PeopleSoft applications to use the Oracle Help Center, see Configuring Context-Sensitive Help Using the Hosted Online Help Website.

Locally Installed Help

If you're setting up an on-premise PeopleSoft environment, and your organization has firewall restrictions that prevent you from using the hosted PeopleSoft Online Help, you can install the online help locally. See Configuring Context-Sensitive Help Using a Locally Installed Online Help Website.

Downloadable PeopleBook PDF Files

You can access downloadable PDF versions of the help content in the traditional PeopleBook format on the Oracle Help Center. The content in the PeopleBook PDFs is the same as the content in the PeopleSoft Online Help, but it has a different structure and it does not include the interactive navigation features that are available in the online help.

Common Help Documentation

Common help documentation contains information that applies to multiple applications. The two main types of common help are:

- Application Fundamentals
- Using PeopleSoft Applications

Most product families provide a set of application fundamentals help topics that discuss essential information about the setup and design of your system. This information applies to many or all applications in the PeopleSoft product family. Whether you are implementing a single application, some combination of applications within the product family, or the entire product family, you should be familiar with the contents of the appropriate application fundamentals help. They provide the starting points for fundamental implementation tasks.

Copyright © 1988, 2021, Oracle and/or its affiliates.

In addition, the *PeopleTools: Applications User's Guide* introduces you to the various elements of the PeopleSoft Pure Internet Architecture. It also explains how to use the navigational hierarchy, components, and pages to perform basic functions as you navigate through the system. While your application or implementation may differ, the topics in this user's guide provide general information about using PeopleSoft applications.

Field and Control Definitions

PeopleSoft documentation includes definitions for most fields and controls that appear on application pages. These definitions describe how to use a field or control, where populated values come from, the effects of selecting certain values, and so on. If a field or control is not defined, then it either requires no additional explanation or is documented in a common elements section earlier in the documentation. For example, the Date field rarely requires additional explanation and may not be defined in the documentation for some pages.

Typographical Conventions

The following table describes the typographical conventions that are used in the online help.

Typographical Convention	***Description***
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.
. . . (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 (|).
[ ] (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.
⇒	This continuation character has been inserted at the end of a line of code that has been wrapped at the page margin. The code should be viewed or entered as a single, continuous line of code without the continuation character.

ISO Country and Currency Codes

PeopleSoft Online Help topics use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

ISO country codes may appear as country identifiers, and ISO currency codes may appear as currency identifiers in your PeopleSoft documentation. Reference to an ISO country code in your documentation

Copyright © 1988, 2021, Oracle and/or its affiliates.

does not imply that your application includes every ISO country code. The following example is a country-specific heading: "(FRA) Hiring an Employee."

The PeopleSoft Currency Code table (CURRENCY_CD_TBL) contains sample currency code data. The Currency Code table is based on ISO Standard 4217, "Codes for the representation of currencies," and also relies on ISO country codes in the Country table (COUNTRY_TBL). The navigation to the pages where you maintain currency code and country information depends on which PeopleSoft applications you are using. To access the pages for maintaining the Currency Code and Country tables, consult the online help for your applications for more information.

Region and Industry Identifiers

Information that applies only to a specific 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 region-specific heading: "(Latin America) Setting Up Depreciation"

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in the PeopleSoft Online Help:

- 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 the PeopleSoft Online Help:

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

Translations and Embedded Help

PeopleSoft 9.2 software applications include translated embedded help. With the 9.2 release, PeopleSoft aligns with the other Oracle applications by focusing our translation efforts on embedded help. We are not planning to translate our traditional online help and PeopleBooks documentation. Instead we offer very direct translated help at crucial spots within our application through our embedded help widgets. Additionally, we have a one-to-one mapping of application and help translations, meaning that the software and embedded help translation footprint is identical—something we were never able to accomplish in the past.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Using and Managing the PeopleSoft Online Help

Select About This Help in the left navigation panel on any page in the PeopleSoft Online Help to see information on the following topics:

- Using the PeopleSoft Online Help
- Managing Hosted online help
- Managing locally installed PeopleSoft Online Help

PeopleTools Related Links

PeopleTools 8.59 Home Page

PeopleTools Elasticsearch Home Page

"PeopleTools Product/Feature PeopleBook Index" (PeopleTools 8.59: Getting Started with PeopleTools)

PeopleSoft Hosted Online Help

PeopleSoft Information Portal

PeopleSoft Spotlight Series

PeopleSoft Training and Certification | Oracle University

My Oracle Support

Oracle Help Center

Contact Us

Send your suggestions to psoft-infodev_us@oracle.com. Please include the applications update image or PeopleTools release that you're using.

Follow Us

YouTube

Twitter@PeopleSoft_Info.

PeopleSoft Blogs

Copyright © 1988, 2021, Oracle and/or its affiliates.

LinkedIn

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 1

Getting Started with PeopleCode

PeopleCode Overview

PeopleCode is the proprietary language used by PeopleSoft applications. This chapter provides an overview of PeopleCode and discusses how to create PeopleCode programs.

These topics provide information to consider before you begin to use PeopleCode. In addition to the considerations presented in this section, you should take advantage of all PeopleSoft sources of information, including the installation guides, release notes, and PeopleBooks.

This section provides an overview of the product documentation available for the PeopleCode language, which is split into three documents:

- *PeopleTools: PeopleCode API Reference*

  This document contains information about certain application classes delivered with Oracle's PeopleTools, as well as specifics about each class's methods and properties.

- *PeopleTools: PeopleCode Developer's Guide*

  This document, the document that you are currently reading, provides conceptual information on the PeopleCode language, how PeopleCode interacts with the component buffer, how to develop PeopleCode programs, and a number of other specialized topics.

- *PeopleTools: PeopleCode Language Reference*

  This document contains information about PeopleCode built-in functions, meta-SQL, system variables, and meta-HTML.

The initial topics in this document, *PeopleTools: PeopleCode Developer's Guide*, provide conceptual information on the PeopleCode language, including:

- PeopleCode resembles other programming languages. However, many aspects are unique to the language and the PeopleTools environment. To learn more about the language, see PeopleCode Language Structure.

- PeopleCode is an object-oriented language. To learn about objects and how they're used in PeopleCode, see Classes and Objects.

- The component buffer is the area in memory that stores data for the currently active component. Which fields are loaded into the component buffer, as well as how to access them, is covered in Understanding Component Buffer Structure and Contents.

- The system uses a data buffer as well as the component buffer. The data buffer is used to store data added from sources other than the component, such as from a Application Engine program, an application message, and so on. For information about this buffer, see Understanding Data Buffer Access.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- All PeopleCode is associated with a definition and an event. The events run in a particular order from the Component Processor. To learn more about the Component Processor and the standard event set, see Understanding the Component Processor.
- You should take into account certain considerations when creating applications to be used in the PeopleSoft Pure Internet Architecture. These include how to make your code more efficient when running on the internet, as well as considerations when using specific definitions. See PeopleSoft Pure Internet Architecture Processing Considerations.
- There are restrictions on using some of the functions and methods in the PeopleCode language, as well as considerations for others, like using standalone rowsets and the OLE functions. These are covered in Functions.
- PeopleCode has a tremendous amount of specialized functionality, such as:
  - Using the GenerateTree function to create a tree in your application.
  - Viewing, adding, and deleting files.

  See Using the GenerateTree Function.

  See Understanding the File Attachment Functions .

Creating PeopleCode Programs

Additional topics in this document, *PeopleTools: PeopleCode Developer's Guide*, provide information on creating PeopleCode programs, including:

All PeopleCode programs are associated with a definition as well as an event. To learn more about where you can place your PeopleCode, and have it executed as part of the Component Processor event flow, see Accessing PeopleCode and Events.

See Accessing Page PeopleCode.

Use the PeopleCode editor to create your PeopleCode programs. All the functionality of the PeopleCode editor is described in Using the PeopleCode Editor.

See Using the PeopleCode Editor.

Every PeopleCode program is associated with a definition. The following definitions have additional functionality associated with the PeopleCode editor:

- SQL definitions
- Application Package definitions

See Understanding the SQL Editor Window.

See Creating Application Packages.

After you have created your program, you must run it. Often, that involves fixing any errors that you find. The PeopleCode debugger is an integrated part of PeopleSoft Application Designer, and it has many useful tools for determining where code errors are occurring. All the functionality is described in Debugging your Application.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

See Understanding the PeopleCode Debugger.

After your PeopleCode program is running, you may want to either improve its performance or the user experience. Techniques for doing this are discussed in Improving Your PeopleCode.

See Understanding PeopleCode Programs and Events.

Copyright © 1988, 2021, Oracle and/or its affiliates.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 2

Understanding the PeopleCode Language

PeopleCode Language Structure

This subject assumes that you are familiar with a programming language, such as C, Visual Basic, or Java.

In its fundamentals, PeopleCode syntax resembles other programming languages. Some aspects of the PeopleCode language, however, are specifically related to the PeopleTools environment. Definition name references, for example, enable you to refer to PeopleTools definitions, such as record definitions or pages, without using hard-coded string literals. Other language features, such as PeopleCode data types and metastrings, reflect the close interaction of PeopleTools and SQL. Dot notation, classes, and methods in PeopleCode are similar to other object-oriented languages, like Java.

Typographical Conventions

The following topic describes typographical conventions that apply to PeopleCode and are used throughout PeopleSoft documentation:Typographical Conventions

Data Types

Conventional data types include number, date, string. Use them for basic computing. Object data types instantiate objects from PeopleTools classes. The appropriate use of each data type is demonstrated where the documentation discusses PeopleCode that uses that data type.

Declare variables before you use them.

This section discusses:

- Conventional data types.
- Object data types.

Related Links

Variables

Conventional Data Types

PeopleCode includes these conventional data types:

- any

  When variables and function return values are declared as any, the data type is indeterminate, enabling PeopleTools to determine the appropriate type of value based on context. Undeclared local variables are any by default.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- boolean
- date
- datetime
- float
- integer

Note: The float and integer data types should be used instead of Number only when a performance analysis indicates that the increased speed is useful and an application analysis indicates that the different representations will not affect the results of the computations.

- number
- string
- time

Considerations for Float, Integer, and Number Types

The Integer type is a number represented as a 32-bit signed twos complement number, so it has a range of -2,147,483,648 to 2,147,483,647.

The Float type is a number represented using the machine floating binary point (double precision) representation. This floating binary point representation is not appropriate for exact calculations involving decimal fractions; in particular, calculations involving money. For example, because a tenth (1/10 or .1) cannot be exactly represented in floating binary point, a floating binary point sum of .10 + .10 is not be equal to .20.

The Number type has a size limit of 34 digits, not including the decimal point, with a maximum of 32 digits to the right of the decimal point. Since the Number type is a floating decimal point representation, it is the appropriate data type for calculations involving money.

Operations (other than division) are done using integer arithmetic if the operands are both integers and the destination is an integer, even if the variable is declared as the Number type. The destination is considered to be an integer if one of the following is True:

- The destination is an assignment to an integer variable or parameter.
- The destination is an array subscript.
- The destination is the right-hand operand of a comparison and the left-hand operand is an integer.
- The destination is a when-expression part of an evaluate statement, and the expression evaluated at the start of the evaluate statement is an integer.
- The destination is a for-loop initial, limit, or step expression and the control variable of the for-loop is an integer.

Division (the / operator) is never performed using integer arithmetic. It is always performed using the floating-decimal-point arithmetic, even if the result variable is declared as an Integer type.

Follow these recommendations for assigning types to numbers:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Use Number for most application data values.
- Use Integer when you are counting items, such as rows in a rowset.
- Use Float only when you are tuning the code for performance (after it is already working).

  In addition, you should only use the Float type when you are certain that the resulting loss of precision will not affect the application and that the increase in the speed of the computation makes a difference to the transaction. In general, few applications should use the Float type.

Object Data Types

For most classes in PeopleTools, you need a corresponding data type to instantiate objects from that class.

See Classes and Objects.

PeopleCode includes these built-in component buffer access types:

- Field
- Record
- Row
- Rowset

PeopleCode also includes these built-in display data types:

- AnalyticGrid
- Chart
- Gantt
- Grid
- GridColumn
- OrgChart
- Page
- RatingBoxChart

PeopleCode also includes these built-in internet script data types:

- Cookie
- Request
- Response

PeopleCode includes numerous miscellaneous data types—for example, Array, Chart, Exception, File, Message, XmlDoc, among many others.

Copyright © 1988, 2021, Oracle and/or its affiliates.

API Object Types

Use this data type for any ApiObject, such as a session object, a tree object, a component interface, a portal registry, and so on.

The following ApiObject data type objects can be declared as type Global:

- Session
- PSMessages collection
- PSMessages
- All tree classes (trees, tree structures, nodes, levels, and so on)
- All query classes

All other ApiObject data type objects (such as all the PortalRegistry classes) must be declared as Local.

Comments

Use comments to explain, preferably in language comprehensible to anyone reading your program, what your code does. Comments also enable you to differentiate between PeopleCode delivered with the product and PeopleCode that you add or change. This differentiation helps in your analysis for debugging and upgrades.

Note: Use comments to place a unique identifier marking any changes or enhancements that you have made to a PeopleSoft application. This marker makes it possible for you to search for all the changes you have made, which is particularly helpful when you are upgrading a database.

You insert comments into PeopleCode in these ways:

- You can insert C-style comments that span multiple lines with **/*** at the beginning and ***/** at the end.
- You can use a **Rem** (remark) statement for commenting.

  Put a semicolon at the end of a Rem comment. If you do not, everything up to the end of the next statement is treated as part of the comment.
- You can comment blocks of code and nested C-style comments using **<*** at the start and ***>** at the end.

  Use this type of comment to enclose one set of comments within another set. You generally use this when you are testing code and want to comment out a section that already contains comments.

Warning! In application classes, you will see the use of /+ +/ style comments. *Do not use these in your PeopleCode programs.* These annotations are generated by the compiler. If you use them, they are removed by the system the next time you validate, compile, or save your PeopleCode. They are used to provide signature information on application class methods and properties, and they are regenerated each time the compiler compiles your application class PeopleCode. Instead, use the standard commenting mechanisms listed above.

Note: Commented text cannot exceed a maximum of 16383 characters.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

The following code sample shows comment formatting:

```
<* this program is no longer valid commenting out
entire thing

Rem This is an example of commenting PeopleCode;
/* ----- Logic for Compensation Change ----- */
/* Recalculate compensation change for next row.
Next row is based on prior value of EFFDT.  */

calc_next_compchg(&OLDDT, EFFSEQ, 0);

/* Recalculate compensation change for current row and next row.
Next row is based on new value of EFFDT.  */

calc_comp_change(EFFDT, EFFSEQ, COMP_FREQUENCY, COMPRATE,
CHANGE_AMT, CHANGE_PCT);

calc_next_compchg(EFFDT, EFFSEQ, 0);

*>
```

Note: All text between the <* and *> comment markers is scanned. You may receive an error message for certain conditions, such as mismatched quotation marks, when using this type of comment.

Statements

A statement can be a declaration, an assignment, a program construct (such as a Break statement or a conditional loop), or a subroutine call.

This section discusses:

- Separators.
- Assignment statements.
- Language constructs.
- Branching statements.
- Conditional loops.

Separators

PeopleCode statements are generally terminated with a semicolon. The PeopleCode language accepts semicolons even if they are not required, such as after the last statement completed within an If statement. This functionality enables you to consistently add semicolons after each statement.

Extra spaces are ignored. They are removed by the PeopleCode Editor when you save the code.

Assignment Statements

The assignment statement is the most basic type of statement in PeopleCode. It consists of an equal sign with a variable name on the left and an expression on the right:

```
variableName = expression;
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

The expression on the right is evaluated, and the result is placed in the variable named on the left. Depending on the data types involved, the assignment is passed either by value or by reference.

Assignment by Value

In most types of assignments, the result of the right-hand expression is assigned to the variable as a newly created value, in the variable's own allocated memory area. Subsequent changes to the value of that variable have no effect on any other data.

Assignment by Reference

When both sides of an assignment statement are object variables, the result of the assignment is not to create a copy of the object in a unique memory location and assign it to the variable. Instead, the variable points to the object's memory location. Additional variables can point to the same object location.

For example, both &AN and &AN2 are arrays of type Number. Assigning &AN2 to &AN does *not* assign a copy of &AN2 to &AN. Both array objects point to the same information in memory.

```
Local array of number &AN, &AN2;
Local number &NUM;

&AN = CreateArray(100, 200, 300);
&AN2 = &AN;
&NUM = &AN[1];
```

In the code example, &AN2 and &AN point to the same object: an array of three numbers. If you were to change the value of &AN[2] to 500 and then reference the value of &AN2[2], you would get 500, not 300. On the other hand, assigning &NUM to the first element in &AN (100) is *not* an object assignment. It is an assignment by value. If you changed &AN[1] to 500, then &NUM remains 200.

Note: In PeopleCode, the equal sign can function as either an assignment operator or a comparison operator, depending on context.

Language Constructs

PeopleCode language constructs include:

- Branching structures: If and Evaluate.
- Loops and conditional loops: For, Repeat, and While.
- Break, Continue, and Exit statements loop control and terminating programs.
- The Return statement for returning from functions.
- Variable and function declaration statements: Global, Local, and Component for variables, and Declare Function for functions.
- The Function statement for defining functions.
- Class definition statements.
- Try, Catch, and Throw statements for error handling.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Functions as Subroutines

PeopleCode, like C, does not have subroutines as we generally refer to them. PeopleCode subroutines are the subset of PeopleCode functions only that are defined to return no value or to return a value optionally. Calling a subroutine is the same as calling a function with no return value:

```
function_name([param_list]);
```

Related Links

Branching Statements

Functions

"Function" (PeopleTools 8.59: PeopleCode Language Reference)

"Declare Function" (PeopleTools 8.59: PeopleCode Language Reference)

"CreateException" (PeopleTools 8.59: PeopleCode Language Reference)

"try" (PeopleTools 8.59: PeopleCode Language Reference)

Branching Statements

Branching statements control program flow based on evaluation of conditional expressions.

If, Then, and Else statements

The syntax of If, Then, and Else statements is:

```
If condition Then
   [statement_list_1;]
[Else
   [statement_list_2;]]
End-If;
```

This statement evaluates the Boolean expression *condition.* If *condition* is True, then the If statement executes the statements in *statement_list_1*. If *condition* is False, then the program executes the statements in the Else clause; if there is no Else clause, the program continues to the next statement.

Evaluate Statement

Use the Evaluate statement to check multiple conditions. Its syntax is:

```
Evaluate left_term
   When [relop_1] right_term_1
      [statement_list;]
   [When [relop_n] right_term_n
     [statement_list;]]
   [When-other
      [statement_list;]]
End-Evaluate;
```

The Evaluate statement takes an expression, *left_term,* and compares it to compatible expressions (*right_term*) using the relational operators (*relop*) in a sequence of When clauses. If *relop* is omitted, then the equal sign is assumed. If the result of the comparison is True, the program executes the statements in the When clause, and then moves on to evaluate the comparison in the following When clause. The program executes the statements in all of the When clauses for which the comparison evaluates to True. If none of the When comparisons evaluates to True, the program executes the statement in the When-other

Copyright © 1988, 2021, Oracle and/or its affiliates.

clause, if one is provided. For example, the following Evaluate statement executes only the first When clause. &USE_FREQUENCY in the following example can only have one of three string values:

```
Evaluate &USE_FREQUENCY
When = "never"
   &PROD_USE_FREQ = 0;
When = "sometimes"
   &PROD_USE_FREQ = 1;
When = "frequently"
   &PROD_USE_FREQ = 2;
When-Other
   Error "Unexpected value assigned to &USE_FREQUENCY."
End-Evaluate;
```

To end the Evaluate statement after the execution of a When clause, you can add a Break statement at the end of the clause, as in the following example:

```
Evaluate &USE_FREQUENCY
When = "never"
   &PROD_USE_FREQ = 0;
   Break;
When = "sometimes"
   &PROD_USE_FREQ = 1;
   Break;
When = "frequently"
   &PROD_USE_FREQ = 2;
   Break;
When-Other
   Error "Unexpected value assigned to &USE_FREQUENCY."
End-Evaluate;
```

In rare cases, you may want to make it possible for more than one When clause to execute, as shown in the following example:

```
Evaluate &PURCHASE_AMT
When >= 100000
   &BASE_DISCOUNT = "Y";
When >= 250000
   &SPECIAL_SERVICES = "Y";
When >= 1000000
   &MUST_GROVEL = "Y";
End-Evaluate;
```

For Statement

The For statement repeats a sequence of statements a specified number of times. Its syntax is:

```
For count = expression1 to expression2 [Step i]
   statement_list;
End-For;
```

The For statement initializes the value of *count* to *expression1,* and then increments *count* by *i* each time after it executes the statements in *statement_list.* The program continues in this loop until *count* is equal to *expression2.* If the Step clause is omitted, then *i* equals one. To count backwards from a higher value to a lower value, use a negative value for *i.* You can exit a For loop using a Break statement.

The following example demonstrates the For statement:

```
&MAX = 10;

Local integer &COUNT;
For &COUNT = 1 To &MAX;
   WinMessage("Executing statement list, count = " | &COUNT);
End-For;
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"If" (PeopleTools 8.59: PeopleCode Language Reference)

"Evaluate" (PeopleTools 8.59: PeopleCode Language Reference)

"For" (PeopleTools 8.59: PeopleCode Language Reference)

Conditional Loops

Conditional loops, Repeat and While, repeat a sequence of statements, evaluating a conditional expression each time through the loop. The loop terminates when the condition evaluates to True. You can exit from a conditional loop using a Break statement. If the Break statement is in a loop embedded in another loop, the break applies only to the inside loop.

Repeat Statement

The syntax of the Repeat statement is:

```
Repeat
   statement_list;
Until logical_expression;
```

The Repeat statement executes the statements in *statement_list* once, and then evaluates *logical_expression*. If *logical_expression* is False, the sequence of statements is repeated until *logical_expression* is True.

While Statement

The syntax of the While statement is:

```
While logical_expression
   statement_list;
End-While;
```

The While statement evaluates *logical_expression* before executing the statements in *statement_list.* It continues to repeat the sequence of statements until *logical_expression* evaluates to False.

Related Links

"Repeat" (PeopleTools 8.59: PeopleCode Language Reference)

"While" (PeopleTools 8.59: PeopleCode Language Reference)

Functions

This section discusses:

- Supported functions.
- Function definitions.
- Function declarations.
- Function calls.
- Function return values.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Function naming conflicts.

Related Links

Classes and Objects
Functions
"Function" (PeopleTools 8.59: PeopleCode Language Reference)
"Declare Function" (PeopleTools 8.59: PeopleCode Language Reference)

Supported Functions

PeopleCode supports the following types of functions:

- Built-in: The standard set of PeopleCode functions. These can be called without being declared.
- Internal: Functions that are defined (using the Function statement) within the PeopleCode program in which they are called.
- External PeopleCode: PeopleCode functions defined outside the calling program. These are generally contained in record definitions that serve as function libraries.
- External non-PeopleCode: Functions stored in external (C-callable) libraries.

Note: PeopleSoft Analytic Calculation Engine provides its own set of built-in functions.

See "Understanding the Analytic Model Definition Creation Process" (PeopleTools 8.59: Analytic Calculation Engine).

In addition, PeopleCode supports methods. The main differences between a built-in function and a method are:

- A built-in function, in your code, is on a line by itself, and it does not (generally) have any dependencies.

  You do not have to instantiate an object before you can use the function.

- A method can only be executed by an object (using dot notation).

  You must instantiate the object first.

Function Definitions

PeopleCode functions can be defined in any PeopleCode program. Function definitions must be placed at the top of the program, along with any variable and external function declarations. The syntax for a PeopleCode function definition is as follows:

```
Function name[(paramlist)] [Returns data_type]
    [statements]
End-function
```

By convention, PeopleCode programs are stored in records whose names begin in FUNCLIB_, and they are always attached to the FieldFormula event.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: Application classes can provide an alternative, and sometimes cleaner, mechanism for separating functionality than the functions stored in function libraries.

Related Links

"Function" (PeopleTools 8.59: PeopleCode Language Reference)

"Application Classes General Structure" (PeopleTools 8.59: PeopleCode API Reference)

Function Declarations

If you call an external function from a PeopleCode program, you must declare the function at the top of the program. The syntax of the function declaration varies, depending on whether the external function is written in PeopleCode or compiled in a dynamic link library.

The following is an example of a function declaration of a function that is in another FUNCLIB record definition:

```
Declare Function UpdatePSLOCK PeopleCode FUNCLIB_NODES.MSGNODENAME FieldFormula;
```

Related Links

"Declare Function" (PeopleTools 8.59: PeopleCode Language Reference)

Function Calls

Functions are called with this syntax:

```
function_name([param_list]);
```

The optional parameter list *(param_list)* is a list of expressions, separated by commas, that the function expects you to supply. If a parameter is listed in the function definition, then it is required when the function is called.

You can check the values of parameters that get passed to functions at runtime in the Parameter window of the PeopleCode debugger.

If the return value is required, then the function must be called as an expression, for example:

```
&RESULT = Product(&RAISE_PERCENT, .01, EMPL_SALARY);
```

If the function has an optional return value, it can be called as a subroutine. If the function has no return value, it *must* be called as a subroutine:

```
WinMessage(64, "I can't do that, " | &OPER_NICKNAME | ".");
```

Parameters are always passed to internal and external PeopleCode functions by reference. If the function is supposed to change the data the caller passes, you must also pass in a variable.

Built-in function parameters can be passed by reference or by value, depending on the function. External C function parameters can be passed by value or by reference, depending on the declaration and type.

Related Links

Understanding the PeopleCode Debugger

Variables and Functions

Copyright © 1988, 2021, Oracle and/or its affiliates.

Function Return Values

Functions can return values of any supported data type; some functions do not return any value.

Optional return values occur only in built-in functions. You cannot define a function that optionally returns a value. Optional return values are typical in functions that return a Boolean value indicating whether execution was successful. For example, the following call to DeleteRow ignores the Boolean return value and deletes a row:

```
DeleteRow(RECORD.BUS_EXPENSE_PER, &L1_ROW, RECORD.BUS_EXPENSE_DTL, &L2_ROW);
```

The following example checks the return value and returns a message saying whether it succeeded:

```
if DeleteRow(RECORD.BUS_EXPENSE_PER, &L1_ROW, RECORD.BUS_EXPENSE_DTL, &L2_ROW) then
   WinMessage("Row deleted.");
else
   WinMessage("Sorry -- couldn't delete that row.");
end-if;
```

Function Naming Conflicts

If you define a function with the same name as a built-in function, the function that you defined takes precedence over the built-in function.

Anytime you compile the PeopleCode in the PeopleCode Editor, a warning message appears in the Validate tab, indicating that a user-defined function has the same name as an existing built-in function.

In addition, if you select Compile All PeopleCode, an error message is generated in the log file for every user-defined function that has the same name as a built-in function.

The following is an example error message: User-defined function IsNumber is overriding the built-in function of the same name. (2,98)

If you notice that you named a function the same as a built-in function, and that the built-in function does what you're trying to achieve, replace your function with a reference to the built-in function. The built-in function is probably more efficient. In addition, using the built-in function reduces confusion for people who maintain your code, because if they miss the warning message in the Validate tab, they might assume the built-in function is being called when it is not.

Expressions

This section discusses:

- Expression fundamentals.
- Constants.
- Functions as expressions.
- System variables.
- Metastrings.
- Record field references.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Definition name references.
- PeopleCode reserved words.

Related Links

Variables

Expression Fundamentals

Expressions evaluate to values of PeopleCode data types. A simple PeopleCode expression can consist of a constant, a temporary variable, a system variable, a record field reference, or a function call. Simple expressions can be modified by unary operators (such as a negative sign or logical NOT), or combined into compound expressions using binary operators (such a plus sign or logical AND).

Definition name references evaluate to strings equal to the name of a PeopleTools definition, such as a record or page. They enable you to refer to definitions without using string literals, which are difficult to maintain.

Metastrings (also called meta-SQL) are special expressions used within SQL string literals. At runtime, the metastrings expand into the appropriate SQL for the current database platform.

Constants

PeopleCode supports numeric, string, and Boolean constants, as well as user-defined constants. It also supports the constant Null, which indicates an object reference that does not refer to a valid object.

Note: You can express Date, DateTime, and Time values by converting from String and Number constants using the Date, Date3, DateTime6, DateTimeValue, DateValue, Time3, TimePart, and the TimeValue functions. You can also format a DateTime value as text using FormatDateTime.

Numeric Constants

Numeric constants can be any decimal number. Some examples are:

- 7
- 0.8725
- -172.0036

String Constants

String constants can be delimited by using either single (') or double (") quotation marks. If a quotation mark occurs as part of a string, the string can be surrounded by the other delimiter type. As an alternative, you can include the delimiter twice. Some examples are:

- "This is a string constant."
- 'So is this.'
- 'She said, "This is a string constant."'

Copyright © 1988, 2021, Oracle and/or its affiliates.

- "She said, ""This is a string constant."""

Use the following code to include a literal quotation mark as part of your string:

```
&cDblQuote = '"'; /* singlequote doublequote singlequote */
```

The following also produces a string with two double quotation marks in it:

```
&cDblQuote = """"; /* dquote dquote dquote dquote */
```

You can also directly embed the doubled double quotation mark in strings, such as:

```
&sImage = Char(10) | '<IMG SRC="%IMAGE(' | &pImageName | ')"';
```

Strings must be contained on a single line. If you need to create a multi-line string, you must use concatenation to connect the lines to be a single sting. For example, one method to do this is:

```
&string = "Line 1" | Char(10) | "Line 2" | Char(10);
```

Boolean Constants

Boolean constants represent a truth value. The two possible values are True and False.

Null Constant

Null constants represent an object reference value that does not refer to a valid object. This means that calling a method on the object or trying to get or set a property of it fails. The Null constant is just the keyword Null.

User-Defined Constants

You can define constants at the start of a PeopleCode program. Then you can use the declared constant anywhere that the corresponding value would be allowed. Constants can be defined as numbers, strings, or Boolean values.

User-defined constants can only be declared as Local.

The following is an example of user-defined constant declarations:

```
Constant &Start_New_Instance = True;
Constant &Display_Mode = 0;
Constant &AddMode = "A":
Local Field &Start_Date;
. . .
MyFunction(&Start_New_Instance, &Display_Mode, &Add_Mode);
```

Functions as Expressions

You can use any function that returns a value as an expression. The function can be used on the right side of an assignment statement, passed as a parameter to another function, or combined with other expressions to form a compound expression.

Related Links

Functions

 Copyright © 1988, 2021, Oracle and/or its affiliates.

System Variables

System variables are preceded by a percent (%) symbol whenever they appear in a program. Use these variables to get the current date and time, or to get information about the user, the current language, the current record, page, or component, and more.

Related Links

"Understanding System Variables" (PeopleTools 8.59: PeopleCode Language Reference)

"System Variables Reference" (PeopleTools 8.59: PeopleCode Language Reference)

Metastrings

Metastrings are special SQL expressions. The metastrings, also called meta-SQL, are preceded with a percent (%) symbol, and can be included directly in string literals. They expand at runtime into an appropriate substring for the current database platform. Metastrings are used in or with:

- SQLExec.
- Scroll buffer functions (ScrollSelect and its relatives).
- PeopleSoft Application Designer to construct dynamic views.
- Some rowset object methods (Select, SelectNew, Fill, and so on).
- SQL objects.
- Application Engine.
- Some record class methods (Insert, Update, and so on).
- COBOL.

Related Links

"SQLExec" (PeopleTools 8.59: PeopleCode Language Reference)

"ScrollSelect" (PeopleTools 8.59: PeopleCode Language Reference)

"Understanding Meta-SQL" (PeopleTools 8.59: PeopleCode Language Reference)

Record Field References

Use record field references to retrieve the value stored in a record field or to assign a value to a record field.

Record Field Reference Syntax

References to record fields have the following form:

```
[recordname.]fieldname
```

You must supply the *recordname* only if the record field and your PeopleCode program are in different record definitions.

Copyright © 1988, 2021, Oracle and/or its affiliates.

For example, suppose that in a database for veterinarians you have two records, PET_OWNER and PET. A program in the record definition PET_OWNER must refer to the PET_BREED record field in the PET record definition as PET.PET_BREED.

However, a program in the PET record definition can refer to this same record field more directly as PET_BREED.

If the program is in the PET_BREED record field itself, it can refer to this record field using the caret (^) symbol.

The PeopleCode Editor replaces the caret symbol with the actual record field name.

You can also use object dot notation to refer to record fields, for example:

```
&FIELD = GetRecord(RECORD.PET_OWNER).GetField(FIELD.PET_BREED);
```

See Understanding Component Buffer Structure and Contents.

Legal Record Field Names

A record field name consists of two parts, the record name and the field name, separated by a period.

The field names used in PeopleCode are consistent with the field names allowed in the field definition. Case is ignored, although the PeopleCode Editor for the sake of convention, automatically formats field names in uppercase. A field name can be 1 to 18 characters, consisting of alphanumeric characters determined by your current language setting in Microsoft Windows, and characters #, @, $, and _.

A record name can be 1 to 15 characters, consisting of alphanumeric letters determined by your current language setting in Microsoft Windows, and characters #, @, $, and _.

Definition Name References

Definition name references are special expressions that reference the name of a PeopleTools definition, such as a record, page, component, business interlink, and so on. Syntactically, a definition name reference consists of a reserved word indicating the type of definition, followed by a period, then the name of the PeopleTools definition. For example, the definition name reference RECORD.BUS_EXPENSE_PER refers to the definition name BUS_EXPENSE_PER.

Generally, definition name references are passed as parameters to functions. If you attempt to pass a string literal instead of a definition name reference to such a function, you receive a syntax error.

You also use definition name references outside function parameter lists, for example, in comparisons:

```
If (%Page = PAGE.SOMEPAGE) Then
   /* do stuff specific to SOMEPAGE */
End-If;
```

In these cases, the definition name reference evaluates to a string literal. Using the definition name reference instead of a string literal enables PeopleTools to maintain the code if the definition name changes.

If you use the definition name reference, and the name of the definition changes, the change automatically ripples through the code, so you do not have to change it or maintain it.

In the PeopleCode Editor, if you place your cursor over any definition name reference and right-click, you can select View Definition to open the definition.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

In addition, for most definitions, if you specify a definition that was not created in PeopleSoft Application Designer, you receive an error message when you try to save your program.

Legal and Illegal Definition Names

Legal definition names, as far as definition name references are concerned, consist of alphanumeric letters determined by your current language setting in Microsoft Windows, and the characters #, @, $, and _.

In some cases, however, the definition supports the use of other characters. You can, for example, have a menu item named A&M stored in the menu definition even though & is an illegal character in the definition name reference. The illegal character results in an error when you validate the syntax or attempt to save the PeopleCode.

You can avoid this problem in two ways:

- Rename the definition so that it uses only legal characters.
- Enclose the name of the definition in quotation marks in the reference, for example: ITEMNAME."A&M"

The second solution is a commonly used workaround in cases where the definition name contains illegal characters. If you use this notation, the definition name reference is not treated as a string literal: PeopleTools maintains the reference the same way as it does other definition name references.

Note: If your definition name begins with a number, you must enclose the name in quotation marks when you use it in a definition name reference. For example, CompIntfc."1_DISCPLIN_ACTN".

PeopleCode Reserved Words

Reserved Word	*Common Usage*
AnalyticModel	Used with the GetAnalyticModel method.
BarName	Used with transfers and modal transfers.
BusActivity	Used with TriggerBusinessEvent.
BusEvent	Used with TriggerBusinessEvent.
BusProcess	Used with TriggerBusinessEvent.
CompIntfc	Used with Component Interface Classes.
Component	Used with transfers and modal transfers, as well as for generating URLs.
Field	Used with methods and functions to designate a field.
FileLayout	Used with the SetFileLayout File class method.
HTML	Used with the GetHTMLText function.
Image	Used in with functions and methods to designate an image.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Reserved Word	*Common Usage*
Interlink	Used with the GetInterlink function.
ItemName	Used with transfers and modal transfers.
Market	Used with transfers and URL generation.
MenuName	Used with transfers and modal transfers.
Message	Used with Messaging functions and methods.
Node	Used with transfers and modal transfers, as well as generating URLs.
Operation	Used with the CreateMessage function.
Page	Used with transfers and modal transfers to pass the page item name (instead of the page name), and with controls and other functions to pass the page name.
Panel	Used with the deprecated TransferPanel function. **Note:** Use the TransferPage function and the **Page** reserved word instead.
PanelGroup	Used with the deprecated DoModalPanelGroup function. **Note:** Use the DoModalComponent function and the **Component** reserved word instead.
Portal	Used with transfers and modal transfers, as well as generating URLs.
Record	Used in functions and methods to designate a record.
Rowset	Used in functions and methods to designate a rowset.
RowsetCache	
Scroll	The name of the scroll area in the page. This name is always equal to the primary record of the scroll.
SQL	Used with SQL definitions.
StyleSheet	Used with style sheets.
URL	Used with file attachment functions.

Variables

This section discusses.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Supported variable types.
- User-defined variables.
- User-defined variable declaration and scope.
- Variable declaration.
- User-defined variable initialization.
- Restrictions on variable use.
- Scope of local variables.
- Duration of local variables.
- Variables and functions.
- Recursive functions.
- State of shared objects using PeopleSoft Pure Internet Architecture.

Related Links

System Variables

Supported Variable Types

PeopleCode supports these types of variables:

User-defined variables	These variable names are preceded by an & character wherever they appear in a program. Variable names can be 1 to 1000 characters, consisting of letters A through Z and a through z, digits 0 through 9, and characters #, @, $, and _.
System variables	System variables provide access to system information. System variables have a prefix of the % character rather than the & character. Use these variables wherever you can use a constant, passing them as parameters to functions or assigning their values to fields or to temporary variables.

User-Defined Variables

A user-defined variable can hold the contents of a record field for program code clarity. For example, you may give a variable a more descriptive name than a record field, based on the context of the program. If the record field is from another record, you may assign it to a temporary variable rather than always using the record field reference. This makes it easier to enter the program, and can also make the program easier to read.

Also, if you find yourself calling the same function repeatedly to get a value, you may be able to avoid some processing by calling the function once and placing the result in a variable.

Copyright © 1988, 2021, Oracle and/or its affiliates.

User-Defined Variable Declaration and Scope

The difference between the variable declarations concerns their life spans:

- Global

  The variable is valid for the entire session.

  Global variables can be accessed from different components and applications, including an Application Engine program. A global variable must be declared, however, in each PeopleCode program where it's used. Use global variables rarely, because they are difficult to maintain.

  Global variables are not available to a portal or applications on separate databases.

- Component

  Component variables remain defined and keep their values while any page in the component in which they are defined remains active. Similar to a global variable, a component variable must be declared in each PeopleCode program where it is used.

  Component variables remain defined after a TransferPage, DoModal, or DoModalComponent function. However, variables declared as Component do not remain defined after using the Transfer function, whether you are transferring within the same component or not.

  Component variables act the same as global variables when an Application Engine program is called from a page (using CallAppEngine).

- Local

  The variable is valid for the duration of the PeopleCode program or function in which the variable is defined.

  Local variables declared at the top of a PeopleCode program (or within the main, that is, non-function, part of a program) remain in scope for the life of that PeopleCode program. Local variables declared within a method or function are valid to the end of the method or function and not beyond.

You can explicitly declare variables using the Global, Component, or Local statements, or you can use local variables without declaring them. Here are some examples of explicit variable declarations:

```
Local Number &AGE;
Global String &OPER_NICKNAME;
Component Rowset &MY_ROWSET;
Local Any &SOME_FIELD;
Local ApiObject &MYTREE;
Local Boolean &Compare = True;
Component PTFP_FEED:FeedFactory &cFeedFactory;
```

Variable declarations are usually placed above the main body of a PeopleCode program (along with function declarations and definitions). The exception is the Local declaration, which you can use within a function or the main section of a program. You can declare variables as any of the PeopleCode data types. If a variable is declared as an Any data type, or if a variable is not declared, PeopleTools uses an appropriate data type based on context.

Note: Declare a variable as an explicit data type unless the variable will hold a value of an unknown data type.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

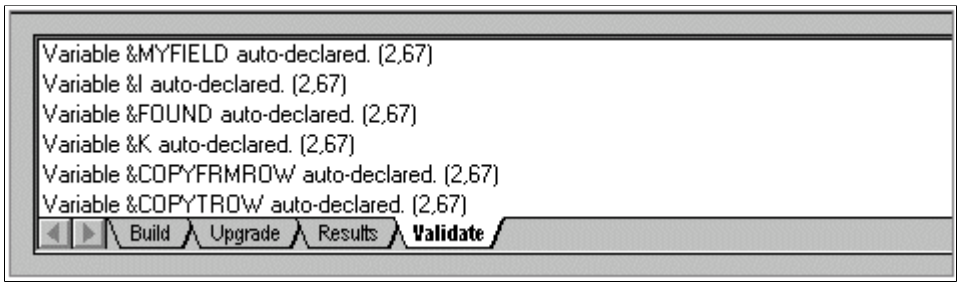
You can check the values of Global, Component, or Local variables at runtime in the different variable windows of the PeopleCode debugger. Local variables declared within a function appear in the Function Parameters window.

Variable Declaration

Declare variables before you use them. If you do not declare a variable, it is automatically declared with the scope Local and the data type Any. You receive a warning message in the Validation tab of the PeopleSoft Application Designer output window for every variable that is not declared when you save the PeopleCode program, as shown in the following example:

Image: Validation tab with auto-declared variables

The following image shows the validation tab with auto-declared variables.



If you declared all the variables, you can use these values to ensure you do not have misspellings. For example, if you declared a variable as &END_DATE and then accidentally spell it as &EDN_DATE, the “new variable” appears on the Validate tab when you save the program.

Another reason to declare variables is for the design-time checking. If you declare a variable of one data type and then assign to it a value of a different type, the PeopleCode Editor catches that assignment as a design-time error when you try to save the program. With an undeclared variable, the assignment error does not appear until runtime.

The following example produces a design-time error when you try to save the program:

```
Local Field &DATE;

&DATE = GetRecord(RECORD.DERIVED_HR);
```

In addition, if you declare variables, the Find Object Reference feature finds embedded definitions. For example, suppose you wanted to find all occurrences of the field DEPT_ID. If you have not declared &MyRecord as a record, Find Object References does not find the following reference of the field DEPT_ID:

```
&MyRecord.DEPT_ID.Visible = False;
```

User-Defined Variable Initialization

To declare and initialize variables in one step, use the following format:

```
Local String &MyString = "New";

Local Date &MyDate = %Date;
```

This method is available only for variables with the scope of Local.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Though you can declare more than one variable on a single line, you can only initialize one variable on a line. The following code creates a syntax error when you try to save the program:

```
Local Number &N1, &N2 = 5;
```

You cannot declare a variable, then initialize it in a second declaration statement. The following produces a duplicate declaration error when you try to save the program:

```
Global Number &N1;
...
Local String &N1 = "Str";  /* Duplicate definition. */
```

If you do not initialize variables, either when you declare them or before you use them, strings are initialized as Null strings, dates and times as Null, and numbers as zero.

Restrictions on Variable Use

The following data types can only be declared as Local:

- JavaObject
- Interlink

  Note: Interlink objects can be declared as type Global in an Application Engine program.

- TransformData
- XmlNode

The following ApiObject data type objects can be declared as Global:

- Session
- PSMessages collection
- PSMessage
- All tree classes (trees, tree structures, nodes, levels, and so on)
- Query classes

All other ApiObject data type objects (such as all the PortalRegistry classes) must be declared as Local.

Scope of Local Variables

The two types of local variables are: program-local and function-local.

- A program-local variable is declared as local in the main part of the program and is local to that program.
- A function-local variable is declared as local inside a function and is local only to that function.

See Recursive Functions.

A program-local variable can be affected by statements anywhere in the program. For example, suppose RECORD_A.FIELD_A.FieldFormula has two functions, FUNC_1 and FUNC_2, and both modify a local

 Copyright © 1988, 2021, Oracle and/or its affiliates.

variable named &TEMP. They could affect each other, as they both use the same variable name in the same PeopleCode program.

If, however, FUNC_3 is defined in RECORD_B_FIELD_B.FieldFormula and makes reference to &TEMP, it is *not* the same &TEMP as in RECORD_A.FIELD_A.FieldFormula. This difference becomes important when FUNC_1 calls FUNC_3. Technically, both functions exist at the same time, one inside the other, but &TEMP is a different variable for each of them. However, if FUNC_1 calls FUNC_2, then &TEMP is the same variable for both.

Duration of Local Variables

A local variable is valid for the duration of the PeopleCode program or function in which it is defined. A PeopleCode program is defined as what the PeopleCode Editor in Application Designer presents in a single window: a chunk of PeopleCode text associated with a single item (a record field event, a component record event, and so on.)

When the system evaluates a PeopleCode program and calls a function in the same PeopleCode program, a new program evaluation is not started.

However, when a function from a different PeopleCode program is called (that is, some PeopleCode text associated with a different item), the current PeopleCode program is suspended, and the Component Processor starts evaluating the new program. This means that any local variables in the calling program (called A) are no longer available. Those in the called program (called B) are available.

Even if the local variables in the A program have the same name as those in the B program, they are different variables and are stored separately.

If the called program (B) in turn calls a function in program A, a new set of program A's variables are allocated, and the called function in A uses these new variables. Thus, this second use of program A gets another lifetime, until execution returns to program B.

The following is an example of pseudocode to show how this might work. (This is non-compiled, non-working code. To use this example, you'd have to enter a similar program without the external declaration of the function in the other, not yet compiled, one.)

```
Program A (Rec.Field.FieldChange):
local number &temp;
declare function B1 PeopleCode Rec.Field FieldFormula;
/* Uncomment this declaration and comment above to compile this the first time.
   function B1
   end-function;
*/

function A1
WinMessage("A1: &temp is " | &temp);
&temp = &temp + 1;
A2();
B1();
A2();
end-function;

function A2
WinMessage("A2: &temp is " | &temp);
&temp = &temp + 1;
end-function;

A1();

Program B (Rec.Field.FieldFormula):
local number &temp;
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
declare function A2 PeopleCode Rec.Field FieldChange;

function B1
WinMessage("B1: &temp is " | &temp);
&temp = &temp + 1;
A2();
end-function;
```

When this is compiled and run, it produces the following output:

```
A1: &temp is 0
A2: &temp is 1
B1: &temp is 0
A2: &temp is 0
A2: &temp is 2
```

Variables and Functions

PeopleCode variables are always passed to functions by reference. This means, among other things, that a function can change the value of a variable passed to it so that the variable has the new value on return to the calling routine.

For example, the Amortize built-in function expects you to pass it variables into which it places the amount of a loan payment applied towards interest (&PYMNT_INTRST), the amount of the payment applied towards principal (&PYMNT_PRIN), and the remaining balance (&BAL). It calculates these values based on information that the calling routine supplies in other parameters:

```
&INTRST_RT=12;
&PRSNT_BAL=100;
&PYMNT_AMNT=50;
&PYMNT_NBR=1;
Amortize(&INTRST_RT, &PRSNT_BAL, &PYMNT_AMNT, &PYMNT_NBR,
&PYMNT_INTRST, &PYMNT_PRIN, &BAL);
&RESULT = "Int=" | String(&PYMNT_INTRST) | " Prin=" |
String(&PYMNT_PRIN) | " Bal=" | String(&BAL);
```

Recursive Functions

PeopleCode supports True recursive functions. A function can call itself, and each possibly recursive call of the function has its own independent copy of the parameters and function-local variables.

When writing recursive functions, be careful about passing variables as parameters, because PeopleCode implements such calls by reference. This means that if you call a function such as:

```
Function Func(&n as Number)
&n = 3;
End-Function;
local &x = 5;
Func(&x);
```

After the call to Func(&x), &x has the value 3, not 5. If the call was Func(Value(&x)), after the call &x is still 5.

State of Shared Objects Using PeopleSoft Pure Internet Architecture

Consider the following scenario:

- A local and a global variable refer to the same object.
- That object is used in a modal component.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Instead of completing the modal component, the user clicks the browser Back button.

In general, the global state of the object is restored. If the object has not been destroyed from the global state, the global state of the object is used for local references; otherwise, the local state is used for local references.

Here is an example:

```
Global array of number &Global_Array;
Local array of number &Local_Array;

&Global_Array = CreateArray(1, 2, 3);
&Local_Array = &Global_Array;
DoModal(Page.PAGENAME, "", - 1, - 1, 1, Record.SHAREDREC, 1);

/* return to here */
&Local_Array [1] = - 1;
&Global_Array [2] = - 2;
WinMessage(&Local_Array | " is " | &Local_Array.Join());
WinMessage(&Global_Array | " is " | &Global_Array.Join());
```

The following program, program 2, is located on the modal page the user is transferred to:

```
Global array of number &Global_Array;
&Global_Array[3] = -3;
```

The following program, program 3, is also located on the modal page:

```
Global array of number &Global_Array;
&Global_Array = CreateArray(1, 2, -3);
```

If program 2 is run, the output is the following:

```
&Local_Array is -1, -2, -3
&Global_Array is -1, -2, -3
```

However, if program 3 is run, thereby destroying the original global state, the output is the following:

```
&Local_Array is -1, 2, 3
&Global_Array is 1, -2, -3
```

Related Links

"System Variables Reference" (PeopleTools 8.59: PeopleCode Language Reference)
Understanding the PeopleCode Debugger

Operators

PeopleCode expressions can be modified and combined using math, string, comparison, and Boolean operators.

This section discusses:

- Math operators.
- Operations on dates and times.
- String concatenation.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- @ operator.
- Comparison operators.
- Boolean operators.

Math Operators

PeopleCode uses standard mathematical operators:

- +

  Add

- -

  Subtract (or unary negative sign)

- *

  Multiply

- /

  Divide

- **

  Exponential

Exponentiation occurs before multiplication and division; multiplication and division occur before addition and subtraction. Otherwise, math expressions are evaluated from left to right. You can use parentheses to force the order of operator precedence.

The minus sign can also, of course, be used as a negation operator, as in the following expressions:

```
-10
- &NUM
- Product(&PERCENT_CUT, .01, SALARY)
```

Operations on Dates and Times

You can add or subtract two date values or two time values, which provides a Number result. In the case of dates, the number represents the difference between the two dates in days. In the case of time, the number represents the difference in seconds. You can also add and subtract numbers to or from a time or date, which results in another date or time. Again, in the case of days, the number represents days, and in the case of time, the number represents seconds.

The following table summarizes these operations:

Operation	***Result Type***	***Result Represents***
Time + number of seconds	Time	Resulting time
Date + number of days	Date	Resulting date

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Operation	*Result Type*	*Result Represents*
Date - date	Number	Difference in days
Time - time	Number	Difference in seconds
Date + time	DateTime	Date and time combined

String Concatenation

The string concatenation operator (|) is used to combine strings. For example, assuming &OPER_NICKNAME is “Dave”, the following statement sets &RETORT to “I can’t do that, Dave.”

```
&RETORT = "I can't do that, " | &OPER_NICKNAME |  "."
```

The concatenation operator automatically converts its operands to strings. This conversion makes it easy to write statements that display mixed data types. For example:

```
&DAYS_LEFT = &CHRISTMAS - %Date;
WinMessage("Today is " | %Date | ".  Only " | &DAYS_LEFT | " shopping days left!");
```

@ Operator

The @ operator converts a string storing a definition reference into the definition. This is useful, for example, if you want to store definition references in the database as strings and retrieve them for use in PeopleCode; or if you want to obtain a definition reference in the form of a string from the operator using the Prompt function.

To take a simple example, if the record field EMPLID is currently equal to 8001, the following expression evaluates to 8001:

```
@"EMPLID"
```

The following example uses the @ operator to convert strings storing a record reference and a record field reference:

```
&STR1 = "RECORD.BUS_EXPENSE_PER";
&STR2 = "BUS_EXPENSE_DTL.EMPLID";
&STR3 = FetchValue(@(&STR1), CurrentRowNumber(1), @(&STR2), 1);
WinMessage(&STR3, 64);
```

Note: String literals that reference definitions are not maintained by PeopleTools. If you store definition references as strings, then convert them with the @ operator in the code, this creates maintenance problems whenever definition names change.

The following function takes a rowset and a record, passed in from another program, and performs some processing. The GetRecord method does not take a variable for the record, however, you can dereference the record name using the @ symbol. Because the record name is never hard-coded as a string, if the record name changes, this code does not have to change.

```
Function Get_My_Row(&PASSED_ROWSET, &PASSED_RECORD)

   For &ROWSET_ROW = 1 To &PASSED_ROWSET.RowCount
      &UNDERLYINGREC = "RECORD." | &PASSED_ROWSET.DBRecordName;
      &ROW_RECORD = &PASSED_ROWSET.GetRow(&ROWSET_ROW).GetRecord(@(&UNDERLYINGREC))⇒
;
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
      /* Do other processing */
   End-For;
End-Function;
```

Comparison Operators

Comparison operators compare two expressions of the same data type. The result of the comparison is a Boolean value. The following table summarizes these operators:

Operator	***Meaning***
=	Equal
!=	Not equal
<>	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

You can precede any of the comparison operators with the word *Not,* for example:

- Not=
- Not<
- Not>=

Expressions formed with comparison operators form logical terms that can be combined using Boolean operators.

String comparisons are case-sensitive. You can use the Upper or Lower built-in functions to do a case-insensitive comparison.

Related Links

"Lower" (PeopleTools 8.59: PeopleCode Language Reference)
"Upper" (PeopleTools 8.59: PeopleCode Language Reference)

Boolean Operators

The logical operators And, Or, and Not are used to combine Boolean expressions. The following table shows the results of combining two Boolean expressions with And and Or operators:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Expression 1	*Operator*	*Expression 2*	*Result*
False	And	False	False
False	And	True	False
True	And	True	True
False	Or	False	False
False	Or	True	True
True	Or	True	True

In complex logical expressions using the operations And, Or, and Not, Not takes the highest precedence, And is next, and Or is lowest. Use parentheses to override precedence. (Generally, it is a good idea to use parentheses in logical expressions anyway, because it makes them easier to decipher.) If used on the right side of an assignment statement, Boolean expressions must be enclosed in parentheses.

The following are examples of statements containing Boolean expressions:

```
If ((&HAS_FLEAS Or
      &HAS_TICKS) And
      SOAP_QTY <= MIN_SOAP_QTY) Then
   SOAP_QTY = SOAP_QTY + OrderFleaSoap(SOAP_ORDER_QTY);
End-If;
```

The Not operator negates Boolean expressions, changing a True value to False and a False value to True. The Not operator can also negate comparison operators.

There are several methodologies for negating a Boolean value. You can use the Not operator:

```
Local boolean &Flag;

&Flag = True;
&Flag = ( Not &Flag);
If Not &Flag Then
   WinMessage("Flag = False");
End-If;
```

Alternatively, you can compare the Boolean value to False:

```
Local boolean &Flag;

&Flag = True;
&Flag = (&Flag = False);
If Not &Flag Then
   WinMessage("Flag = False");
End-If;
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 3

Understanding Objects and Classes in PeopleCode

Classes and Objects

PeopleSoft delivers classes of objects that you can manipulate with PeopleCode. In addition, you can extend the existing classes or create your own. The delivered classes may or may not have a graphic user interface equivalent; some are representations of data structures that occur only at runtime. With PeopleCode, you can manipulate data in the data buffer easily and consistently. These classes enable you to write code that's more readable, more easily maintained, and more useful.

This section discusses:

- Classes.
- Objects.
- Object instantiation.

Classes

A *class* is the formal definition of an object and acts as a template from which an instance of an object is created at runtime. The class defines the properties of the object and the methods used to control the object's behavior.

PeopleSoft delivers predefined classes, such as Array, File, Field, SQL, and so on. You can create your own classes using the Application class. You can also extend the functionality of the existing classes using the Application class.

Related Links

"Understanding Application Classes" (PeopleTools 8.59: PeopleCode API Reference)

Objects

An *object* represents a unique instance of a data structure defined by the template provided by its class. Each object has its own values for the variables belonging to its class and responds to methods defined by that class. This is the same for classes provided by PeopleSoft and for classes you create yourself.

After an object has been created (instantiated) from a class, you can change its properties. A *property* is an attribute of an object. Properties define:

- Object characteristics, such as name or value.
- The state of an object, such as deleted or changed.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Some properties are read-only and cannot be set, such as Name or Author. Other properties can be set, such as Value or Label.

Objects are different from other data structures. They include code (in the form of *methods*), not just static data. A *method* is a procedure or routine, associated with one or more classes, that acts on an object.

An analogy to illustrate the difference between an object and its class is the difference between a car and the blue Citroen with license plate number TS5800B. A class is a general category, while the object is a specific instance of that class. Each car comes with standard characteristics, such as four wheels, an engine, or brakes, that define the class and are the template from which the individual car is created. You can change the properties of an individual car by personalizing it with bumper stickers or racing stripes, which is like changing the Name or Visible property of an object. The model and date that the car is created are similar to read-only properties because you cannot alter them. A tune-up acts on the individual car and changes its behavior, much as a method acts on an object.

Object Instantiation

A class is the blueprint for something, like a bicycle, a car, or a data structure. An object is the actual thing that is built using that class (or blueprint.) From the blueprint for a bicycle, you can build a specific mountain bike with 23 gears and tight suspension. From the blueprint of a data structure class, you build a specific instance of that class. *Instantiation* is the term for building that copy, or an instance, of a class.

Creating and Using Objects

This section discusses how to:

- Instantiate objects.
- Change object properties.
- Invoke methods.
- Copy objects.

Instantiating Objects

Generally you instantiate objects (create them from their classes) using built-in functions or methods of other objects. Some objects are instantiated from data already existing in the data buffer. Think about this kind of object instantiation as taking a chunk of data from the buffer, encapsulating it in code (methods and properties), manipulating it, then freeing the references. Some objects can be instantiated from a previously created definition, such as a page or file layout definition, instead of from data.

The following example creates a field object:

```
Local field &MyField

&MyField = GetField();
```

Get functions, which include functions such as GetField, GetRecord, and so on, generally provide access to data that already exists, whether in the data buffers or from an existing definition.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Create functions, which include functions such as CreateObject, CreateArray, CreateRecord, generally create defined objects that do not yet exist in the data buffer. Create functions create only a buffer structure. They do not populate it with data. For example, the following function returns a record object for a record that already exists in the component buffer:

```
&REC = GetRecord();
```

The following example creates a standalone record. However, there is no data in &REC2. The specified record definition must be created previously, but the record does not have to exist in either the component or data buffer:

```
&REC2 = CreateRecord(EMP_CHKLST_ITM);
```

Objects with no built-in functions can only be instantiated from a session object (such as tree classes, component interfaces, and so on). For most of these classes, when you use a Get function, all you get is an identifier for the object. To fully instantiate the object, you must use an Open method.

Related Links

"Understanding Session Class" (PeopleTools 8.59: PeopleCode API Reference)

Changing Properties

To set or get characteristics of an object, or to determine the state of an object, you must access its properties through *dot notation* syntax. Follow the reference to the object with a period, followed by the property, and assign it a value. The format is generally as follows:

```
Object.Property = Value
```

The following example hides the field &MYFIELD:

```
&MYFIELD.Visible = False
```

You can return information about an object by returning the value of one of its properties. In the following example, &X is a variable that is assigned the value found in the field &MYFIELD:

```
&X = &MYFIELD.Value
```

In the following example, a property is used as the test for a condition:

```
If &ROWSET.ActiveRowCount <> &I Then
```

Invoking Methods

You also use dot notation to execute methods. Follow the reference to the object with a period, then with the method name and any parameters the method takes. The format is generally:

```
Object.method();
```

You can string methods and property values together into one statement. The following example strings together the GetField method with the Name property:

```
If &REC_BASE.GetField(&R).Name = &REC_RELLANG.GetField(&J).Name Then
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

Some methods return a Boolean value: True if the method executes successfully; False if it does not. The following method compares all like-named fields of the current record object with the specified record. This method returns as True if all like-named fields have the same value:

```
If &MYRECORD.CompareFields(&OTHERRECORD) Then
```

Other methods return a reference to an object. The GetCurrEffRow method returns a row object:

```
&MYROW = &MYROWSET.GetCurrEffRow();
```

Some methods do not return anything. Each method's documentation indicates what it returns.

Many objects have default methods. Instead of entering the name of the method explicitly, you can use that method's parameters. Objects with default methods are *composite objects;* that is, they contain additional objects within them. The default method is generally the method used to get the lower-level object.

A good example of a composite object is a record object. Record definitions are composed of field definitions. The default method for a record object is GetField.

The following lines of code are equivalent:

```
&FIELD = &RECORD.GetField(FIELD.EMPLID);
&FIELD = &RECORD.EMPLID;
```

Note: If the field you're accessing has the same name as a record property (such as NAME) you cannot use the shortcut method for accessing the field. You must use the GetField method.

Another example of default methods concerns rowsets and rows. Rowsets are made up of rows, so the default method for a rowset is GetRow. The two specified lines of code are equivalent: They both get the fifth row of the rowset:

```
&ROWSET = GetRowSet();

/*the next two lines of code are equivalent */

&ROW = &ROWSET.GetRow(5);
&ROW = &ROWSET(5);
```

The following example illustrates the long way of enabling the Name field on a second-level scroll area (the code is executing on the first-level scroll area):

```
GetRowset(SCROLL.EMPLOYEE_CHECKLIST).GetRow(1).
GetRecord(EMPL_CHKLST_ITM).GetField(FIELD.NAME).Enabled = True;
```

Using default methods enables you to shorten the previous code to the following:

```
GetRowset(SCROLL.EMPLOYEE_CHECKLIST)(1).EMPL_CHKLST_ITM.NAME.
Enabled = True;
```

Expressions of the form *class.name.property* or *class.name.method(..)* are converted to a corresponding object. For example, the code `&temp = RECORD.JOB.IsChanged;` is evaluated as if it were `&temp = GetRecord(RECORD.JOB).IsChanged;`.

Furthermore, the code `JOB.EMPLID.Visible = False;` is evaluated as if it were `GetField(JOB.EMPLID).Visible = False;`.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Copying Objects

Many of the classes delivered with PeopleTools have some sort of copy method, such as the rowset class CopyTo, the tree class Copy, and so on. Unless specifically identified (such as the message class CopyRowsetDelta) all copy methods use the current data of the object. This may be different than the original data values if the object was retrieved from the database and the values in it have been changed either by an end-user or a PeopleCode program.

Assigning Objects

When you assign one object to another, you do not create a copy of the object, but only make a copy of the reference.

In the following example, &A1 and &A2 refer to the same object. The assignment of &A1 to &A2 does not allocate any database memory or copy any part of the original object. It makes &A2 refer to the same object to which &A1 refers.

```
Local Array of Number &A1, &A2;

&A1 = CreateArray(2, 4, 6, 8, 10);
&A2 = &A1;
```

Image: Representation of two arrays

The following diagram shows how references of two arrays point to the same object.

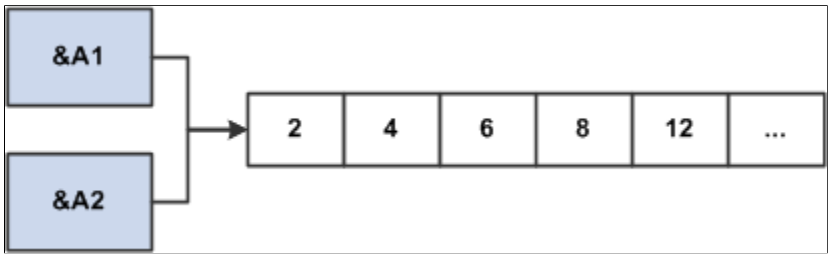
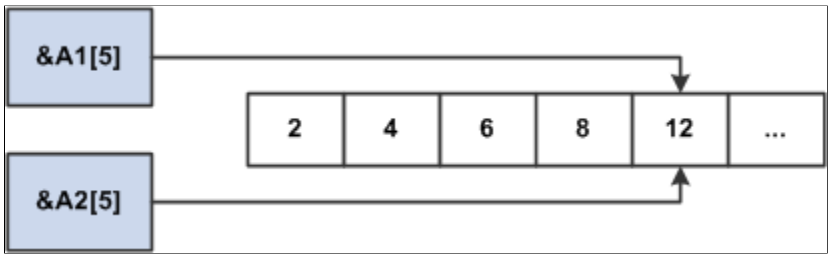


Image: Representation of two arrays with same content

If the next statement is `&A2[5] = 12;`, then &A1[5] also equals 12, as shown in the following diagram:



The following example is *not* considered an object assignment:

```
Local number &NUM;
Local Array of Number &A1;

&A1 = CreateArray(2, 4, 6, 8, 10);
&NUM = &A1[3];
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

&NUM is of data type Number, which is not an object type. If you later change the value of &NUM in the program, you will not change the element in the array.

Passing Objects

All PeopleCode objects can be passed as function parameters. You can pass complex data structures between PeopleCode functions (as opposed to passing long lists of fields). If a function is passed an object, the function works on the actual object, not on a copy of the object.

In the following simple example, a reference to the Visible property is passed, not the *value* of Visible. This enables the MyPeopleCodeFunction either to get or set the value of the Visible property:

```
MyPeopleCodeFunction(&MyField.Visible);
```

In the following example, the function Process_Rowset loops through every row and record in the rowset passed to it and executes an Update statement on each record in the rowset. This function can be called from any PeopleCode program and can process any rowset that is passed to it.

```
Local Rowset &RS;
Local Record &REC;

Function Process_RowSet(&ROWSET as Rowset);

   For &I = 1 To &ROWSET.Rowcount
      For &J = 1 To &ROWSET.Recordcount
         &REC = &ROWSET.GetRow(&I).GetRecord(&J);
         &REC.Update();
      End-For;
   End-For;
End-Function;

&RS = GetLevel0();

Process_RowSet(&RS);
```

The following function takes a rowset and a record passed in from another program. GetRecord does not take a variable for the record; however, you can use the @ symbol to dereference the record name.

```
Function Get_My_Row(&PASSED_ROWSET, &PASSED_RECORD)

   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;
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 4

Referencing Data in the Component Buffer

Understanding Component Buffer Structure and Contents

This section discusses:

- Component buffer contents.
- Rowsets and scroll areas.
- Record fields in the component buffer.

Related Links

Specifying Data with References Using Scroll Path Syntax and Dot Notation

Component Buffer Contents

PeopleCode frequently must refer to data in the component buffer, the area in memory that stores data for the currently active component.

The two methods for specifying a piece of data in the component buffer from within PeopleCode are:

- Contextual references, which refer to data relative to the location of the currently executing PeopleCode program.
- References using scroll path syntax, which provide a complete, or absolute, path through the component buffer to the referenced component.

In addition to the built-in functions used to access the component buffer, PeopleCode provides enhanced access to structured data buffers using the object syntax. Use the object-oriented PeopleCode to resolve contextual ambiguities when you reference a nonprimary record field that appears on more than one scroll level in a component. As with built-in functions, object syntax provides for both relative and absolute references to component buffer data.

See Classes and Objects.

The component buffer consists of rows of buffer fields that hold data for the records associated with page controls, including primary scroll records, related display records, derived/work records, and Translate table records. PeopleCode can reference buffer fields associated with page controls and other buffer fields from the primary scroll record and related display records.

See Record Fields and the Component Buffer.

Primary scroll records are the principal SQL tables or views associated with page scroll levels. A primary scroll record uniquely identifies a scroll level in the context of its page: each scroll level can have only one primary scroll record, and the same primary scroll record cannot occur on more than one scroll area at the same level of the page. Parent-child relations between primary scroll records determine the

Copyright © 1988, 2021, Oracle and/or its affiliates.

dependency structure of the scroll areas on the page. The primary record on a level one scroll area must be a child of the primary record on level zero, the primary record on a level two scroll area must be a child of the primary record on its enclosing level one scroll area, and the primary record on a level three scroll area must be a child of the primary record on its enclosing level two scroll area.

Note: Level zero may have multiple records.

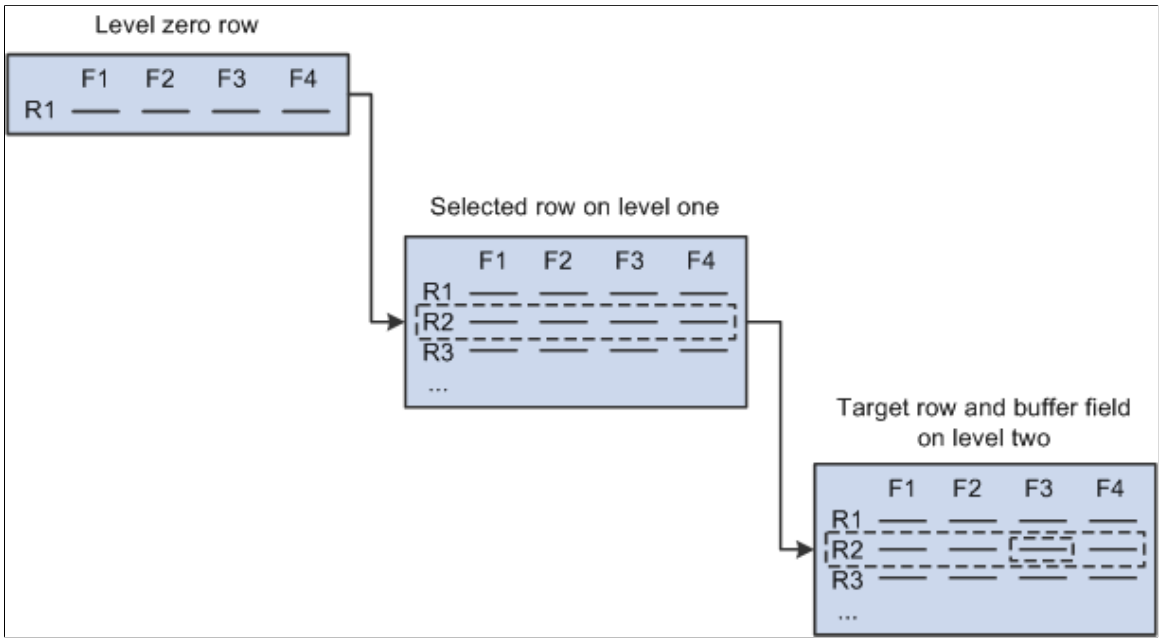
The hierarchical relations among scroll areas, controlled by hierarchical relations among primary scroll records, enable the user and PeopleCode to drill down through the scroll hierarchy to access any buffer field, including related display, derived/work, and Translate table buffer fields, which occupy space on the same rows as the primary scroll record buffer fields with which they are associated.

For example, to access a page field on level two of a page, a user must:

1. Select a field on level one of the page.
2. Scroll to and select the field on level two of the page.

Image: Scroll path to a buffer field

The following diagram illustrates the scroll path taken by the user to a buffer field.



To access the same field in the component buffer, PeopleCode must:

1. Specify a scroll area and row on scroll level one: this selects a subset of dependent rows on level two.
2. Specify a scroll area and row on scroll level two.
3. Specify the recordname.fieldname on the level two row.

PeopleCode component buffer functions use a common scroll path syntax for locating scrolls, rows, and fields in multiple-scroll pages.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Rowsets and Scroll Areas

Rowsets enable more consistent, more convenient, and less ambiguous manipulation of buffer data than previous built-in functions could achieve. It's a hierarchical data object that can represent an entire scroll area and all of its subordinate scroll areas.

A rowset can contain the entire contents of a component buffer, or the contents of any lower-level scroll area plus all of its subordinate buffer data. The hierarchical structure of component levels—scroll area, row, record, field—is provided by the new object data types, Rowset, Row, Record, and Field.

You can access any rowset, row, record, or field within the buffer using the dot notation inherent in PeopleTools 8 object-oriented programming. This enables you to reference fields within a record object, records within a row object, and rows within a rowset object as properties of the parent objects.

Related Links

Understanding Data Buffer Access
"Understanding Rowset Class" (PeopleTools 8.59: PeopleCode API Reference)
Classes and Objects

Record Fields and the Component Buffer

The record fields in the component buffer are a superset of those accessible to the user through page controls. In most cases, PeopleCode can reference any record field in a scroll area's primary scroll record or in a related display record, not just those fields that are associated with page controls. The following table lists record types and locations:

Type and Location of Record	*Presence in Component Buffer*
Primary record on scroll levels greater than zero	On scroll levels greater than zero, all record fields from the primary scroll record are in the component buffer. PeopleCode can refer to any record field on the primary scroll record, even if it is not associated with a page control.
Primary record on scroll level zero	If scroll level zero of a page contains only controls associated with primary scroll record fields that are search keys or alternate search keys, then only the search key and alternate search key fieldsave in the component buffer, not the entire record. The values for the fields come from the keylist, and the record cannot run RowInit PeopleCode. If level zero contains at least one record field from the primary scroll record that is not a search key or alternate search key, then all the record fields from the primary scroll record are available in the buffer. (For this reason, you may sometimes need to add one such record field at level zero of the page to make sure that all the record fields of the level-zero primary record can be referenced from PeopleCode.)
Related display record fields	The buffer contains the related display record field, plus any record fields from the related display record that are referenced by PeopleCode programs. You can reference any record field in a related display record.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Type and Location of Record	*Presence in Component Buffer*
Derived/work record fields	Only derived/work record fields associated with page controls are in the component buffer. Other record fields from the derived/work record cannot be referenced from PeopleCode.
Translate table record fields	Only Translate table fields associated with page controls are available in the component buffer. Other fields from the Translate table cannot be referenced from PeopleCode.

Note: In RowSelect PeopleCode, you can refer only to record fields on the record that is currently being processed.

Specifying Data with Contextual References

In a contextual reference, PeopleCode refers to a row or buffer field determined by the context in which a PeopleCode program is currently executing.

This section includes an overview of current context and discusses how to:

- Use contextual row references.
- Use contextual buffer field references.

Understanding Current Context

All PeopleCode programs, with the exception of programs associated with standard menu items, execute in a current context. The current context determines which buffer fields can be contextually referenced from PeopleCode, and which row of data is the current row on each scroll level at the time a PeopleCode program is executing.

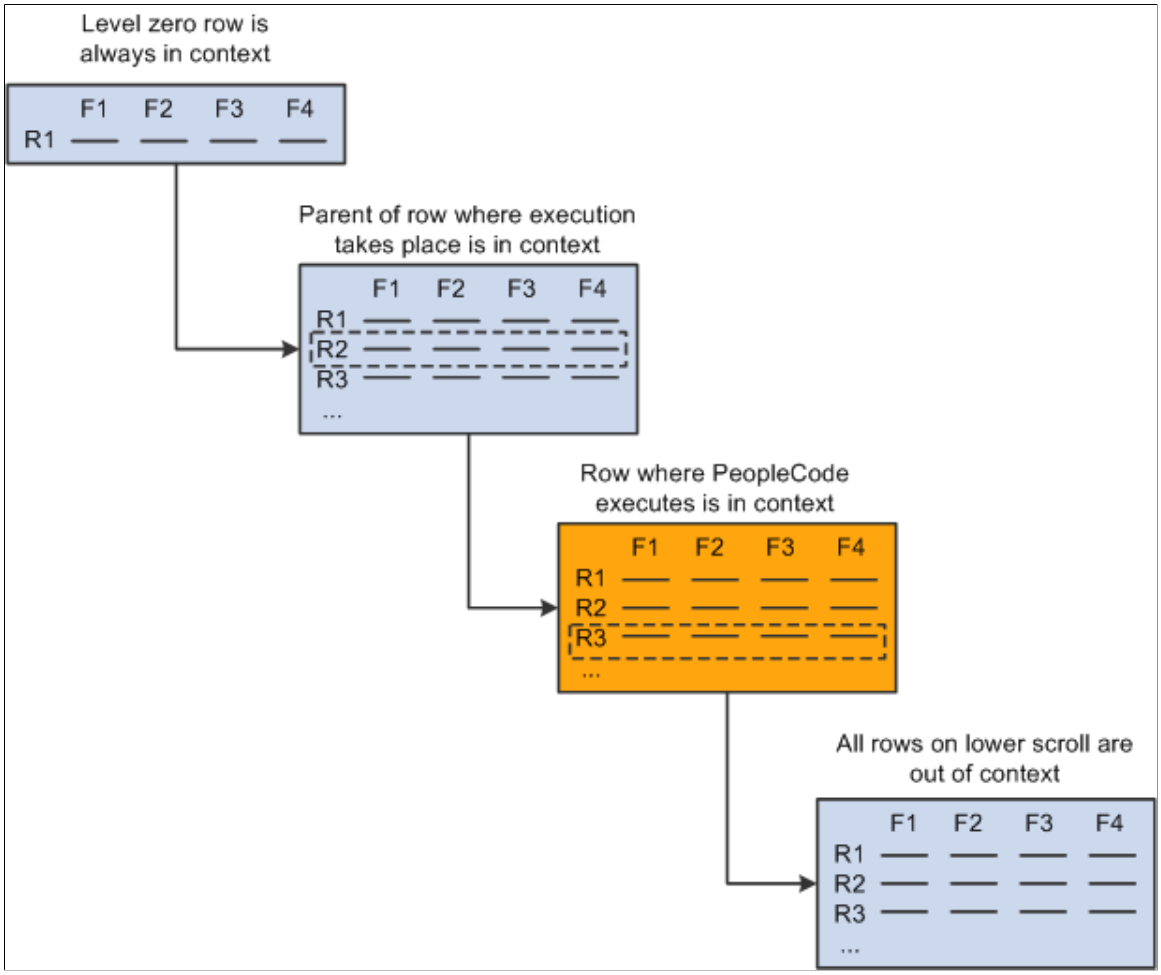
The current context comprises a subset of the buffer fields in the component buffer, determined by the row of data where a PeopleCode program is executing. The current context includes:

- All buffer fields in the row of data where the PeopleCode program is executing.
- All buffer fields in rows that are hierarchically superior to the row where the PeopleCode program is executing.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Context of PeopleCode executing on a level two scroll area

In the following diagram, all rows enclosed in dotted rectangles are part of the current context:



In the preceding diagram, a PeopleCode program is executing in a buffer field on row R3 on scroll level two. The rows in scroll level two are dependent on row R2 on scroll level one. The rows in scroll level one are dependent on the single row at scroll level zero. The current context consists of all the buffer fields at level two row R3, level one row R2, and level zero row R1. The rows in the current context on levels one and two are the current rows on their respective scroll areas. The single row on level zero is always current and is included in any current context. All rows other than the current rows and the level zero row are outside the current context. No current row can be determined on scroll areas below the one where the PeopleCode is executing.

With PeopleTools 8, contextual references work within the structure of a rowset object, and can include references to all field objects, record objects, row objects, and rowset objects in the current context.

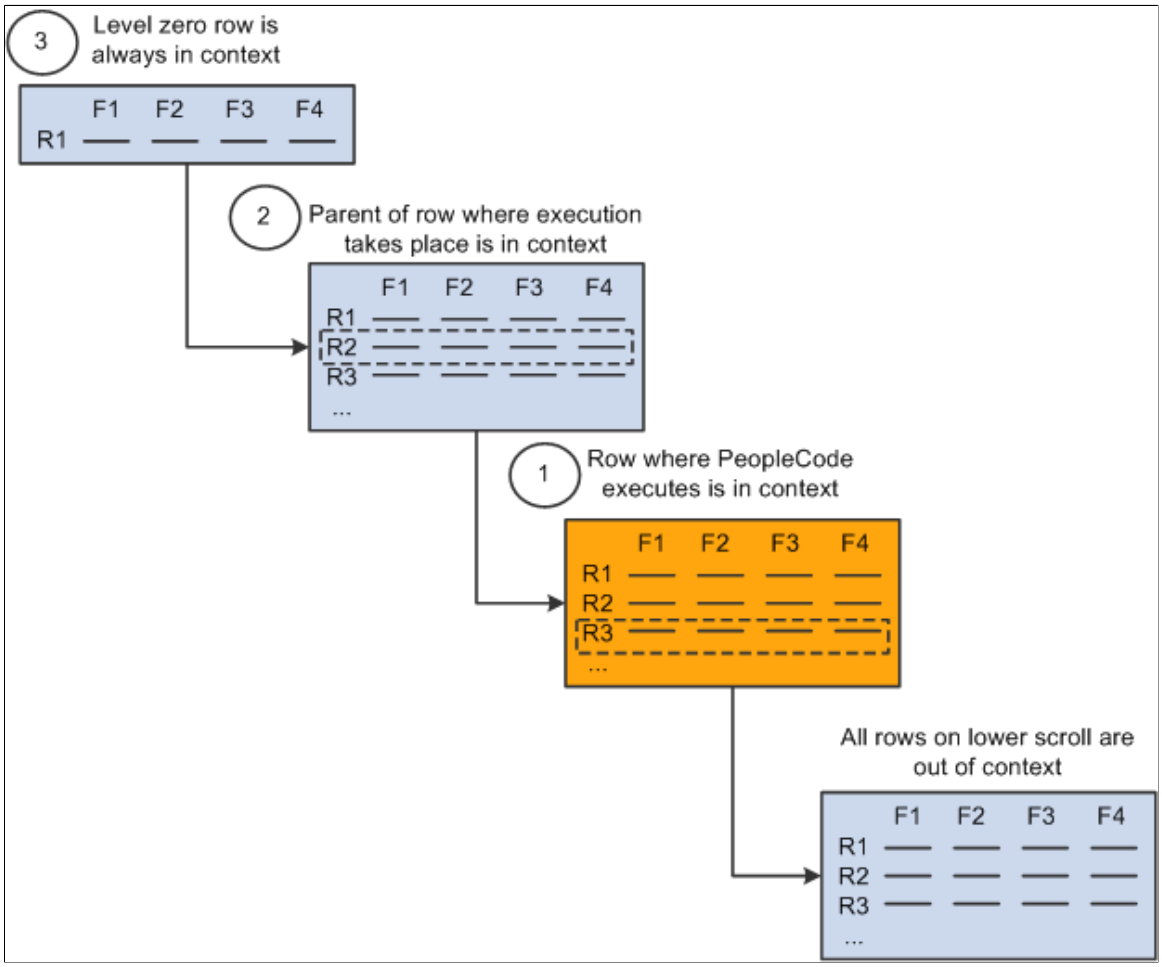
Contextual Reference Processing Order

PeopleCode resolves contextual references at runtime by first checking the row where the PeopleCode program is executing. If PeopleCode does not find an appropriate buffer field, it looks in progressively higher rows in the current context. The following diagram indicates this processing order:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Processing order of a contextual reference

The following diagram indicates the processing order of a contextual reference by PeopleCode program.



In typical pages, this processing order is not significant; however, if the same record occurs on more than one level of a page, you should understand how the direct reference is resolved.

Using Contextual Row References

A contextual row reference refers to a row in the current context on level one or lower in the page. Because each scroll area uses a unique primary record, the name of that record uniquely identifies whichever row is in the current context for that scroll level. A contextual row reference uses a **RECORD.** *recordname* component name reference to specify the scroll level of the intended row, resulting in a reference to the current row at the specified scroll level.

For example, you can use contextual row references with the RecordDeleted, RecordNew, and RecordChanged functions:

```
If RecordDeleted(RECORD.SOME_REC) Then...
```

With PeopleTools 8 object-oriented programming, a row can be referenced by specifying parent rows or rowsets of the current rowset:

```
If GetRowSet().ParentRowset.ParentRow.IsDeleted Then...
```


Copyright © 1988, 2021, Oracle and/or its affiliates.

In early versions of PeopleTools, you could make contextual row references using a *recordname.fieldname* expression:

```
HideRow(SOME_REC.ANY_FIELD)

If RecordDeleted(SOME_REC.ANY_FIELD) Then...
```

This syntax is still supported.

Related Links

Understanding Current Context

Using Contextual Buffer Field References

A contextual buffer field reference is a type of PeopleCode expression that refers to a buffer field by specifying a record field. The row of the buffer field is determined by the current context of the PeopleCode program where the reference is made. You can use a contextual buffer field reference to retrieve or update the value in the buffer field, to pass the buffer field value to a function, or to reference an instance of a page control associated with the buffer field. The following statements use contextual buffer field references:

```
/* Assigns value of variable to buffer field */
SOME_RECORD.SOME_FIELD = &VAL;
/* Assigns value of buffer field to variable */
&VAL = SOME_RECORD.SOME_FIELD;
/* Hides instance of control associated with buffer field */
Hide(SOME_RECORD.SOME_FIELD);
```

With PeopleTools 8 object-oriented programming, a field object incorporates information about both the record field on which the buffer field is based and the page control with which the buffer field is associated. By referring to the field object, you either make a contextual buffer field reference or you change an interface attribute of the associated page control, depending on the object property you use. The following example has the same effect as a contextual buffer field reference:

```
/* Assigns value of a variable to a buffer field */
&MYFIELD.Value = &SOMEVAL;
```

Contextual Buffer Field Reference Ambiguity

Nonprimary record fields may appear on more than one scroll level in a page. For example, a page may use a derived/work field DERIVED_JS.CALC_1 as a work field on level one and level two of the same page. This creates distinct DERIVED_JS.CALC_1 buffer fields for rows on both levels. Because of the order in which PeopleCode resolves contextual buffer field references, if the contextual reference `&VAL = DERIVED_JS.CALC_1;` executes in a PeopleCode program on a level-two row, the reference always retrieves the buffer field value on the current row on level two. PeopleCode on level two is unable to retrieve the value of the DERIVED_JS.CALC_1 on level one using a contextual reference.

To explicitly reference the DERIVED_JS.CALC_1 buffer field on level one, use a component buffer function with a scroll path:

```
&VAL = FetchValue(SCROLL.level1_scrollname, CurrentRowNumber(1), DERIVED_JS.CALC_1)⇒
;
```

The CurrentRowNumber function returns the current row on level one, or the parent row of the level two row where the PeopleCode program is executing.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Ambiguous Contextual References to Buffer Fields on Level Zero

Level zero of a page contains only a single row of data, and the buffer fields in this row are always in the current context. For this reason you can almost always refer to a level zero buffer field using a contextual reference. However, referential ambiguity can make it impossible to reference a buffer field on level zero contextually. For example, a page may use a derived/work field DERIVED_JS.CALC_1 as a work field on level zero and level one of the same page. This creates distinct DERIVED_JS.CALC_1 buffer fields for rows on both levels. Because of the order in which PeopleCode resolves contextual field references, if the `&VAL = DERIVED_JS.CALC_1;` contextual reference executes in a PeopleCode program on a level-one row, it always retrieves the buffer field value on the current row on level one.

To explicitly reference the DERIVED_JS.CALC_1 buffer field on level zero, you must use a component buffer function with this syntax:

Function([*recordname.*]*fieldname, rownum*)

Here *rownum,* because it is on level zero, is always equal to one. In the previous example of the DERIVED_JS.CALC_1 field, you would use this statement:

```
&VAL = FetchValue(DERIVED_JS.CALC_1, 1);
```

Ambiguous References with Objects

With PeopleTools 8 object-oriented programming, even if two field objects that are in different rowsets contain buffer data that's based on the same underlying record field, references to those objects are inherently unique, because each is instantiated with respect to a specific point in the hierarchy of the buffer. Any manipulation of a field object's interface properties affects only the page control with which it's associated.

The following example instantiates a field object using the long form, to emphasize the hierarchical form of the data. It then hides the field's associated page control. Because this is a unique instance of the field, based on its hierarchy, hiding this field does not affect the visibility of any other page control associated with the same record field:

```
&MYFIELD = GetRowset(SCROLL.EMPL_CHECKLIST).GetRow(&I).
GetRecord(RECORD.EMPL_CHECKLIST).GetField(EMPL_CHECKLIST.EMPLID);
&MYFIELD.Visible = False;
/* the same code, using the "short" form */
&MYFIELD = GetRowset(SCROLL.EMPL_CHECKLIST).GetRow(&I).
EMPL_CHECKLIST.EMPLID;
```

Note: Any change in a field object's value affects both the underlying record field and the value of any other field object oriented on the same record field. This behavior is the same as the behavior of contextual buffer field references that alter the field value.

Related Links

Specifying Data with References Using Scroll Path Syntax and Dot Notation

Specifying Data with References Using Scroll Path Syntax and Dot Notation

This section provides an overview of scroll paths and discusses how to:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Structure scroll path syntax.
- Reference scroll levels, rows, and buffer fields.

Related Links

Understanding Data Buffer Access

Understanding Scroll Paths

A scroll path is a construction found in the parameter lists of many component buffer functions, which specifies a scroll level in the currently active page. Additional parameters are required to locate a row or a buffer field at the specified scroll level.

Scroll path syntax enables you to eliminate ambiguous references, which, although rare, do sometimes occur in complex components.

See Using Contextual Buffer Field References.

PeopleTools also provides the convenience of object-oriented dot notation and default methods, which produce inherently non-ambiguous references from PeopleCode programs to component buffer data. There are examples of dot notation in this section along with examples of the legacy scroll path syntax.

Structuring Scroll Path Syntax

PeopleTools offers two legacy constructions for scroll paths: a standard scroll path syntax (**Record.***recordname*) and an alternative syntax using a **Scroll.***scrollname* expression. The latter is more powerful in that it can process some rare cases where a **Record.***recordname* expression results in an ambiguous reference.

Scroll Path Syntax with Record.recordname

Here is the standard scroll path syntax:

```
[Record.level1_recname, level1_row, [Record.level2_recname, level2_row, ]] Record.t⇒
arget_recname
```

If the target level (the level you want to reference) is one, you must supply only the **Record.***target_recname* parameter. If the target scroll level is greater than one, you must provide scroll name and row level parameters for all hierarchically superior scroll levels, beginning at level one. The following table indicates the scroll path syntax for the three possible target scroll levels:

Target Level	*Scroll Path Syntax*
1	**Record.**target_recname
2	**Record.**level1_recname, level1_row, **Record.**target_recname
3	**Record.**level1_recname, level1_row, **Recor⇒ d.**level2_recname, level2_row, **Record.**tar⇒ get_recname

If you are referring to a row or a buffer field, additional parameters are required after the scroll path.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The following table describes the standard scroll path syntax parameters:

Syntax Parameters	*Description*
Record.*level1_recname*	Specifies the name of a record associated with scroll level one, normally the primary scroll record. This parameter is required if the target scroll level is two or three.
level1_row	An integer that selects a row on scroll level one. This parameter is required if the target scroll level is two or three.
Record.*level2_recname*	Specifies the name of a record associated with scroll level two, normally the primary scroll record. This parameter is required if the target row is on scroll level three.
level2_row	An integer that selects a row on scroll level two. This parameter is required if the target row is on scroll level three.
Record.*target_recname*	Specifies a record associated with the target scroll level, generally the primary scroll record. The scroll can be on level one, two, or three of the active page.

Scroll Path Syntax with Scroll.scrollname

As an alternative to **Record.***recordname* expressions in scroll path constructions, PeopleTools permits use of a **Scroll.***scrollname* expression. Scroll paths using **Scroll.***scrollname* are functionally identical to those using **Record.***recordname,* except that **Scroll.***scrollname* expressions are more strict: they can refer only to a scroll level's primary record; whereas **Record.***recordname* expressions can refer to any record in the scroll level, which in some rare cases can result in ambiguous references. (This can occur, for example, if the **Record.***recordname* expression inadvertently references a related display record in another page in the component.) Use of **Record.***recordname* is still permitted, and there is no requirement to use the **Scroll.***scrollname* alternative unless it is needed to avoid an ambiguous reference.

The *scrollname* is the same as the scroll level's primary record name.

The scroll name cannot be viewed or changed through the PeopleSoft Application Designer interface. Here is the complete scroll path syntax using **Scroll.***scrollname* expressions:

```
[Scroll.level1_scrollname, level1_row, [Scroll.level2_scrollname, level2_row, ]], S⇒
croll.target_scrollname
```

The target scroll level in this construction is the scroll level that you want to specify. If the target level is one, you need to supply only the **Scroll.***target_scrollname* parameter. If the target scroll level is greater than one, you need to provide scroll name and row-level parameters for hierarchically superior scroll

 Copyright © 1988, 2021, Oracle and/or its affiliates.

levels, beginning at level one. The following table indicates the scroll path syntax for the three possible target scroll levels:

Target Level	*Scroll Path Syntax*
1	**Scroll.***target_scrollname*
2	**Scroll.***level1_scrollname*, *level1_row*, **Sc⇒ roll.***target_scrollname*
3	**Scroll.***level1_scrollname*, *level1_row*, **Sc⇒ roll.***level2_scrollname*, *level2_row*, **Scro⇒ ll.***target_scrollname*

If the component you are referring to is a row or a buffer field, additional parameters are required after the scroll path.

The following table describes the alternative scroll path syntax parameters:

Parameter	*Description*
Scroll.*level1_scrollname*	Specifies the name of the page's level-one scroll area. This is always the same as the name of the scroll level's primary scroll record. This parameter is required if the target scroll level is two or three.
level1_row	An integer that selects a row on scroll level one. This parameter is required if the target scroll level is two or three.
Scroll.*level2_scrollname*	Specifies the name of the page's level two scroll area. This is always the same as the name of the scroll level's primary scroll record. This parameter is required if the target row is on scroll level three.
level2_row	An integer that selects a row on scroll level two. This parameter is required if the target row is on scroll level three.
Scroll.*target_scrollname*	The scroll name of the target scroll level, which can be level one, two, or three of the active page.

Related Links

Referencing Scroll Levels, Rows, and Buffer Fields

Copyright © 1988, 2021, Oracle and/or its affiliates.

Referencing Scroll Levels, Rows, and Buffer Fields

You can reference a scroll level using the *scrollpath* construct only. Functions that reference rows for buffer fields require additional parameters. The following table summarizes the three types of component buffer references:

Target Component	*Reference Syntax*	*Example Function*
Scroll level	`scrollpath`	`HideScroll(scrollpath);`
Row	`scrollpath, row_number`	`HideRow(scrollpath, row_number);`
Field	`scrollpath, row_number, [recordname.]fieldname`	`FetchValue(scrollpath, row_number, fieldname);`

PeopleTools 8 provides an alternative to the scroll level, row, and field components in the form of the data buffer classes Rowset, Row, Record, and Field, which you reference using dot notation with object methods and properties. The following table demonstrates the syntax for instantiating and manipulating objects in the current context from these classes:

Target Object	*Example Instantiation*	*Example Operation*
Rowset	`&MYROWSET = GetRowset();`	`&MYROWSET.Refresh();`
Row	`&MYROW = GetRow();`	`&MYROW.CopyTo(&SOMEROW);`
Record	`&MYRECORD = GetRecord();`	`&MYREC.CompareFields(&REC);`
Field	`&MYFIELD = GetRecord().fieldname;`	`&MYFIELD.Label = "Last Name";`

The following sections provide examples of functions using scroll path syntax, which refer to an example page from a fictitious veterinary clinic database. The page has three scroll levels, shown in the following table:

Level	*Scroll Name (Primary Scroll Record Name)*
0	VET
1	OWNER
2	PET
3	VISIT

The examples given for PeopleTools 8 object-oriented syntax assumes that the following initializing code was executed:

```
Local Rowset  &VET_SCROLL, &OWNER_SCROLL, &PET_SCROLL, &VISIT_SCROLL;

&VET_SCROLL = GetLevel0();
&OWNER_SCROLL = &VET_Scroll.GetRow(1).GetRowSet(Scroll.OWNER);
&PET_SCROLL = &OWNER_Scroll.GetRow(2).GetRowSet(Scroll.PET);
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
&VISIT_SCROLL = &PET_Scroll.GetRow(2).GetRowSet(Scroll.VISIT);
```

Referring to Scroll Levels

The HideScroll function provides an example of a reference to a scroll level. The syntax of the function is:

HideScroll(*scrollpath*)

where *scrollpath* is

[**Record.***level1_recname*, *level1_row*, [**Record.***level2_recname*, *level2_row*,]] **Record.***target_recname*

To reference the level 1 scroll in the example, use this syntax:

```
HideScroll(Record.OWNER);
```

This hides the OWNER, PET, and VISIT scroll areas on the example page.

In PeopleTools 8, the object-oriented version of this is:

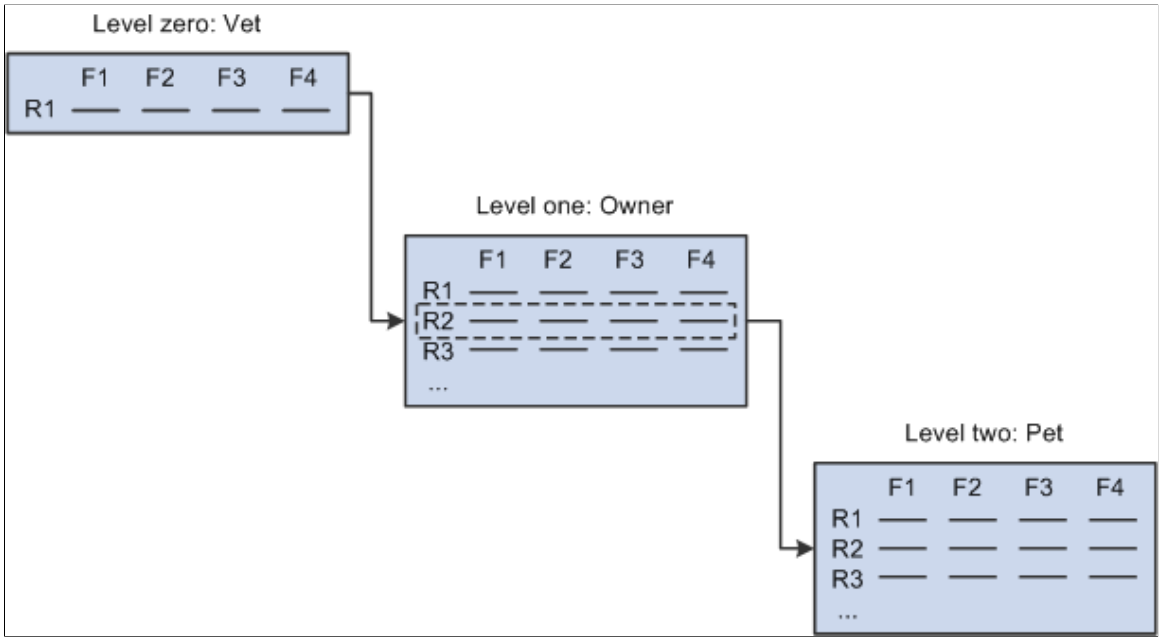
```
&OWNER_Scroll.HideAllRows();
```

To hide scroll levels two and below, supply the primary record and row in scroll level one, and then the record identifying the target scroll area:

```
HideScroll(Record.OWNER, &L1ROW, Record.PET);
```

Image: Sample scroll path

The following diagram shows the scroll path of this statement, assuming that the value of &L1ROW is 2:



Similarly, to hide the VISIT scroll area on level three, you specify rows on scroll levels one and two.

```
HideScroll(Record.OWNER, &L1ROW, Record.PET, &L2ROW, Record.VISIT);
```

To use the **Scroll.***scrollname* syntax, the previous example could be written as the following:

```
HideScroll(Scroll.OWNER, &L1ROW, Scroll.PET, &L2ROW, Scroll.VISIT);
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

In PeopleTools 8, the object-oriented version of this is:

```
&VISIT_Scroll.HideAllRows();
```

Referring to Rows

Referring to rows is the same as referring to scroll areas, except that you need to specify the row you want to select on the target scroll area. As an example, examine the HideRow function, which hides a specific row in the level three scroll area of the page. Here is the function syntax:

HideRow(*scrollpath, target_row*)

To hide row number &ROW_NUM on level one:

```
HideRow(Record.OWNER, &ROW_NUM);
```

To do the same using the **Scroll.***scrollname* syntax:

```
HideRow(Scroll.OWNER, &ROW_NUM);
```

In PeopleTools 8, the object-oriented version of this for the OWNER rowset is:

```
&OWNER_SCROLL(&ROW_NUM).Visible = False;
```

On level two:

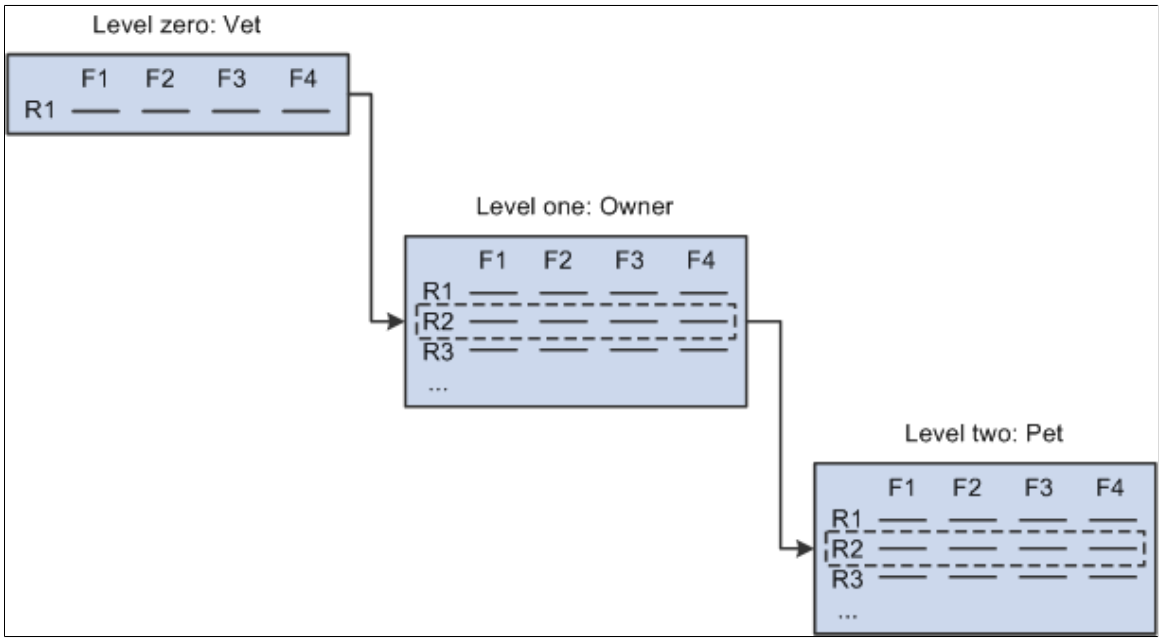
```
HideRow(Record.OWNER, &L1_ROW), Record.PET, &ROW_NUM);
```

In PeopleTools 8, the object-oriented version of this for the PET rowset is:

```
&PET_SCROLL(&ROW_NUM).Visible = False;
```

Image: Scroll path statement

The following diagram indicates the scroll path of this statement, assuming that the value of &L1_ROW is 2 and that &ROW_NUM is equal to 2:



 Copyright © 1988, 2021, Oracle and/or its affiliates.

On level three:

```
HideRow(Record.OWNER, CurrentRowNumber(1), Record.PET,
CurrentRowNumber(2), Record.VISIT, &ROW_NUM);
```

In PeopleTools 8, the object-oriented version of this for the VISIT rowset is:

```
&VISIT_SCROLL(&ROW_NUM).Visible = False;
```

Referring to Buffer Fields

Buffer field references require a [*recordname.*]*fieldname* parameter to specify a record field. The combination of scroll level, row number, and record field name uniquely identifies the buffer field. Here is the syntax:

FetchValue(*scrollpath*, *target_row*, [*recordname.*]*fieldname*)

Assume, for example, that record definitions in the veterinary database have the following fields that you want to reference:

Record	***Sample Field***
OWNER	OWNER_NAME
PET	PET_BREED
VISIT	VISIT_REASON

You could use the following examples to retrieve values on levels one, two, or three from a PeopleCode program executing on level zero.

To fetch a value of the OWNER_NAME field on the current row of scroll area one:

```
&SOMENAME = FetchValue(Record.OWNER, &L1_ROW, OWNER.OWNER_NAME);
```

In PeopleTools 8, the object-oriented version of this for the OWNER rowset is:

```
&SOMENAME = &OWNER_SCROLL(&L1_ROW).OWNER.OWNER_NAME;
```

To fetch PET_BREED on level two:

```
&SOMEBREED = FetchValue(Record.OWNER, &L1_ROW, Record.PET, &L2_ROW, PET.PET_BREED);⇒
```

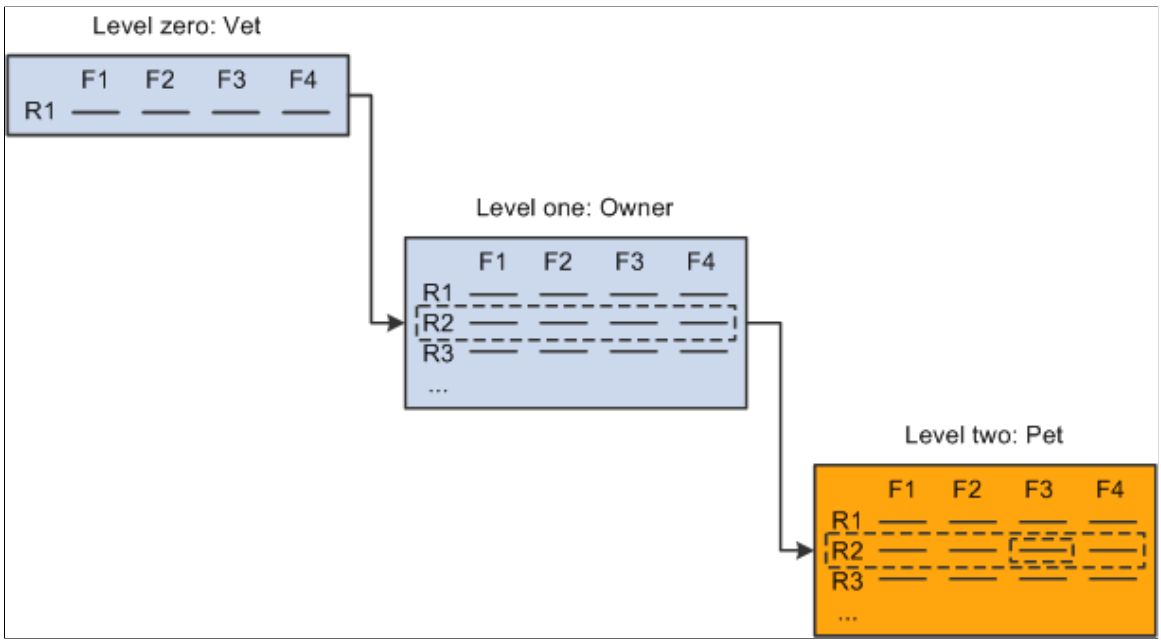
In PeopleTools 8, the object-oriented version of this for the PET rowset is:

```
&SOMEBREED = &PET_SCROLL(&L2_ROW).PET.PET_BREED;
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Scroll path to target field

The following diagram indicates the scroll path to the target field, assuming that &L1_ROW equals 2, &L2_ROW equals 2, and field F3 is PET.PET_BREED.



To fetch VISIT_REASON on level three:

```
&SOMEREASON = FetchValue(Record.OWNER, &L1_ROW, Record.PET,
&L2_ROW, Record.VISIT, &L3_ROW, VISIT.VISIT_REASON);
```

To do the same using the **Scroll.***scrollname* syntax:

```
&SOMEREASON = FetchValue(Scroll.OWNER, &L1_ROW, Scroll.PET,
&L2_ROW, Scroll.VISIT, &L3_ROW, Scroll.VISIT_REASON);
```

In PeopleTools 8, the object-oriented version of this is:

```
&SOMEREASON = &VISIT_SCROLL(&L3_ROW).VISIT.VISIT_REASON;
```

Using CurrentRowNumber

The CurrentRowNumber function returns the current row, as determined by the current context, for a specific scroll level in the active page. CurrentRowNumber is often used to determine a value for the *level1_row* and *level2_row* parameters in scroll path constructions. Because current row numbers are determined by the current context, CurrentRowNumber cannot determine a current row on a scroll level outside the current context (a scroll level below the level where the PeopleCode program is currently executing).

For example, you could use a statement like this to retrieve the value of a buffer field on level three of the PET_VISITS page, in a PeopleCode program executing on level two:

```
&VAL = FetchValue(Record.OWNER, CurrentRowNumber(1),
Record.PET, CurrentRowNumber(2), Record.VISIT, &TARGETROW,
VISIT_REASON);
```

Because the PeopleCode program is executing on level two, CurrentRowNumber can return values for levels one and two, but not three, because level three is outside of the current context and has no current row number.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Looping Through Scroll Levels

Component buffer functions are often used in For loops to loop through the rows on scroll levels below the level where the PeopleCode program is executing. The following loop, for example could be used in PeopleCode executing on a level two record field to loop through rows of data on level three:

```
For &I = 1 To ActiveRowCount(Record.OWNER,
CurrentRowNumber(1), Record.PET, CurrentRowNumber(2), Record.VISIT)
    &VAL = FetchValue(Record.OWNER, CurrentRowNumber(1),
Record.PET, CurrentRowNumber(2), Record.VISIT, &I, VISIT_REASON);
   If &VAL = "Fleas" Then
      /* do something about fleas */
   End-If;
End-For;
```

A similar construct may be used in accessing other level two or level one scroll areas, such as work scroll areas.

In these constructions, the ActiveRowCount function is often used to determine the upper bounds of the loop. When ActiveRowCount is used for this purpose, the loop goes through all of the active rows in the scroll (rows that have not been specified as deleted). If you use TotalRowCount to determine the upper bounds of the loop, the loop goes through all of the rows in the scroll area: first those that have not been specified as deleted, then those that have been specified as deleted.

Related Links

Structuring Scroll Path Syntax

Understanding Current Context

"CurrentRowNumber" (PeopleTools 8.59: PeopleCode Language Reference)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 5

Accessing the Data Buffer

Understanding Data Buffer Access

This section discusses:

- Data buffer access.
- Access classes.
- Data buffer model and data access objects.

Data Buffer Access

In addition to the built-in functions you use to access the component buffer, classes of objects are available that provide access to structured data buffers using the PeopleCode object syntax.

The data buffers accessed by these classes are typically the component buffers that are loaded when you open a component. However, these classes may also be used to access data from general data buffers, loaded by an Application Engine program, a component interface, and so on.

The methods and properties of these classes provide functionality that is similar to what has been available using built-in functions. However, they also provide improved consistency, flexibility, and new functionality.

Access Classes

The four data buffer classes are: Rowset, Row, Record, and Field. These four classes are the foundation for accessing component buffer data through the new object syntax.

A field object, which is instantiated from the Field class, is a single instance of data within a record. It is based on a field definition.

A record object, which is instantiated from the Record class, is a single instance of a data within a row. It is based on a record definition. A record object consists of one to *n* fields.

A row object, which is 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 area is a row. A row may have one to *n* child rowsets. For example, a row in a level two scroll area may have *n* level three child rowsets.

A rowset object is a data structure used to describe hierarchical data. It is made up of a collection of rows. A component scroll area is a rowset. You can also have a level zero rowset.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Data Buffer Model and Data Access Classes

The data model assumed by the data buffer classes is that of a PeopleTools component, where scroll bars or grids are used to describe a hierarchical, multiple-occurrence data structure. You can access these classes using dot notation.

The four data buffer classes relate to each other in a hierarchical manner. The main points to understand these relationships are:

- A record contains one or more fields.
- A row contains one or more records and zero or more child rowsets.
- A rowset contains one or more rows.

For component buffers, think of a rowset as a scroll area on a page that contains all of the data in that scroll area. A level zero rowset contains all the data for the entire component. You can use rowsets with application messages, file layouts, business interlinks, and other definitions in addition to components. A level zero rowset from a component buffer only contains one row: the keys that the user specifies to initiate that component. A level zero rowset from data that is not a component, such as a message or a file layout, might contain more than one level zero row.

The following is basic PeopleCode that traverses through a two-level component buffer using dot notation syntax. Level zero is based on record QA_INVEST_HDR, and level one is based on record QA_INVEST_LN.

```
Local Rowset &HDR_ROWSET, &LINE_ROWSET;
Local Record &HDR_REC, &LINE_REC;
&HDR_ROWSET = GetLevel0();

For &I = 1 to &HDR_ROWSET.RowCount
   &HDR_REC = &HDR_ROWSET(&I).QA_INVEST_HDR;
   &EMPLID = &HDR_REC.EMPLID.Value;
   &LINE_ROWSET = &HDR_ROWSET(&I).GetRowset(1);
   For &J = 1 to &LINE_ROWSET.RowCount
      &LINE_REC = &LINE_ROWSET(&J).QA_INVEST_LN;
      &LINE_SUM = &LINE_SUM + &LINE_REC.AMOUNT.Value;
   End-For;
End-For;
```

Each rowset is declared and instantiated. In general, your code is easier to read and maintain if you follow this practice.

Understanding Data Buffer Classes Examples

This section discusses:

- Employee Checklist page structure.
- Object creation examples.
- Data buffer hierarchy examples.
- Rowset examples.
- Hidden work scroll area example.

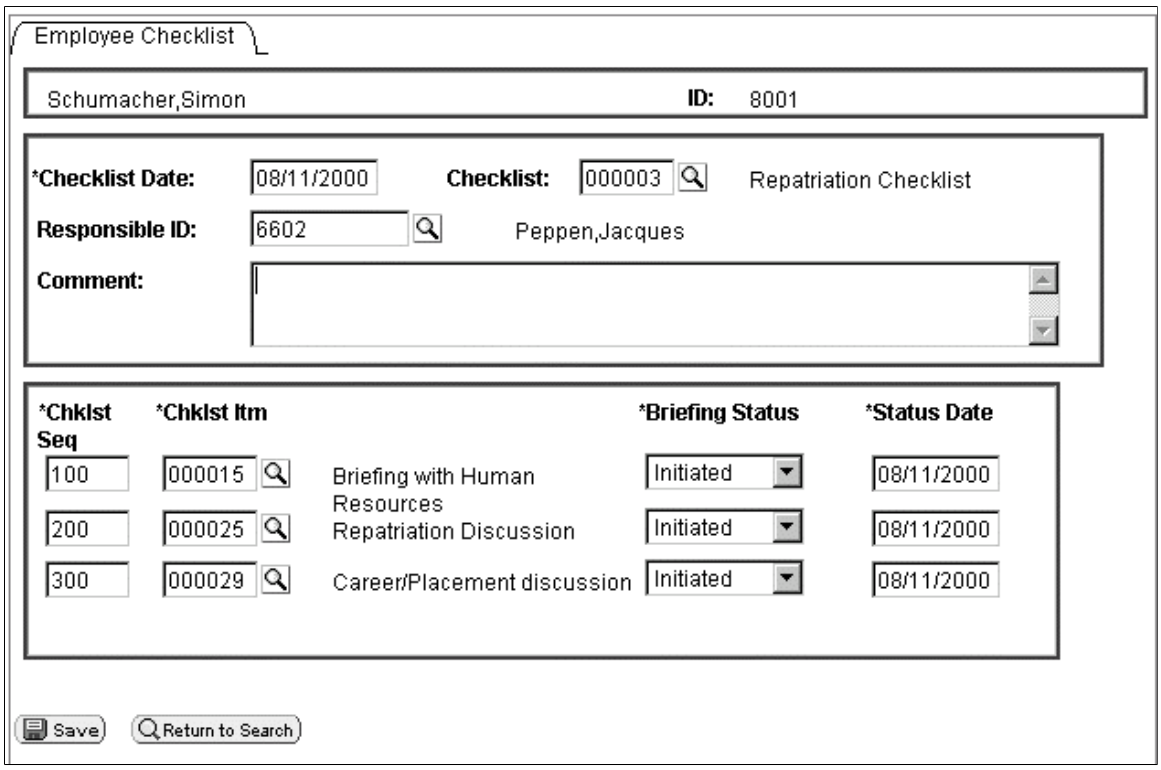
 Copyright © 1988, 2021, Oracle and/or its affiliates.

Employee Checklist Page Structure

Most of the examples in this section use the Employee Checklist page.

Image: Employee Checklist page

The following image illustrates a structure in a Employee Checklist page.



This page has the following record structure:

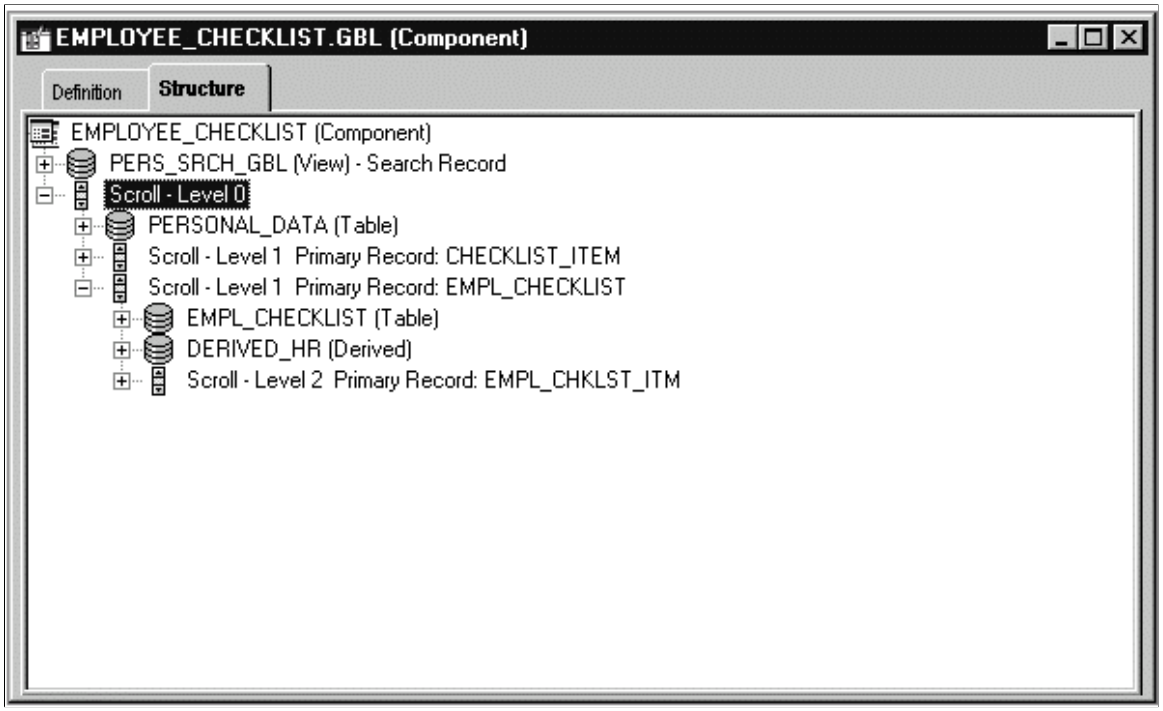
Scroll Level	*Associated Primary Record*	*Rowset and Variable Name*
Level zero	PERSONAL_DATA	Level zero rowset: &RS0
Level one scroll area	EMPL_CHECKLIST	Level one rowset: &RS1
Level one hidden work scroll area	CHECKLIST_ITEM	Level one rowset: &RS1H
Level two scroll area	EMPL_CHKLST_ITM	Level two rowset: &RS2

Another way of looking at the structure of a component is to use the Structure view. All the scroll areas are labeled, and the primary record is associated with each:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: EMPLOYEE_CHECKLIST structure

The following image explains EMPLOYEE_CHECKLIST structure of a component where in all the scroll areas are labeled, and the primary record is associated with each other.



In the example, the visible level one scroll area also has only one row. That row is made up of the following records:

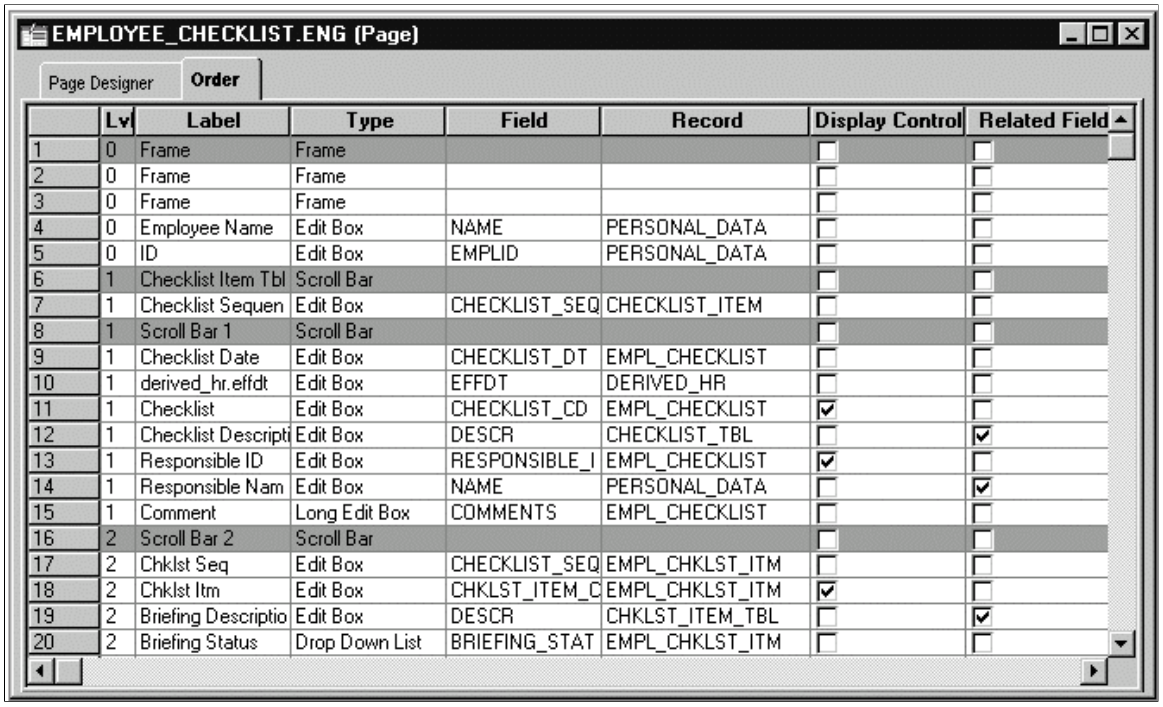
- EMPL_CHECKLIST
- DERIVED_HR
- CHECKLIST_TBL
- PERSONAL_DATA

You can see which records are associated with a scroll area by looking at the Order view for a page:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: EMPLOYEE_CHECKLIST page Order view showing records

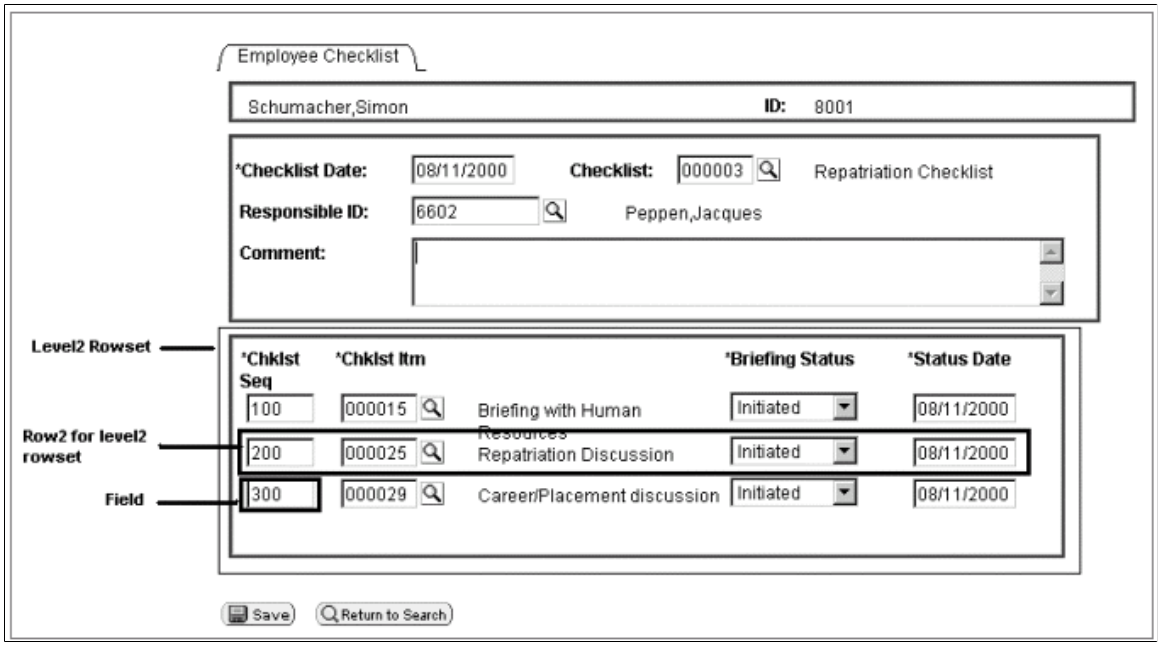
The following image is an example of EMPLOYEE_CHECKLIST page. This page shows the Order view of the records associated with a scroll area.



	Lvl	Label	Type	Field	Record	Display Control	Related Field
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 Descripti	Edit Box	DESCR	CHECKLIST_TBL	☐	☑
13	1	Responsible ID	Edit Box	RESPONSIBLE_I	EMPL_CHECKLIST	☑	☐
14	1	Responsible Nam	Edit Box	NAME	PERSONAL_DATA	☐	☑
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_C	EMPL_CHKLST_ITM	☑	☐
19	2	Briefing Descriptio	Edit Box	DESCR	CHKLST_ITEM_TBL	☐	☑
20	2	Briefing Status	Drop Down List	BRIEFING_STAT	EMPL_CHKLST_ITM	☐	☐

Image: EMPLOYEE_CHECKLIST rowsets and rows

The level two rowset has three rows. Each row is made up of two records: the primary record, EMPL_CHKLST_ITM, and CHKLST_ITM_TBL, the record associated with the related display field DESCR. The following example shows the rowset:



Every record has fields associated with it, such as NAME, EMPLID and CHECKLIST_SEQ. These fields are associated with the record definitions; they are not the fields that appear on the page.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Object Creation Examples

When declaring variables, use the class with the same name as the data buffer access data type (rowset objects should be declared as type Rowset, field objects as type Field, and so on). Data buffer variables can be of type Local, Global, or Component.

The following declarations are assumed throughout the examples that follow:

```
Local Rowset &LEVEL0, &ROWSET;
Local Row &ROW;
Local Record &REC;
Local Field &FIELD;
```

Level Zero Access

The following code instantiates a rowset object, from the Rowset class, that references the level zero rowset, containing all the page data. It stores the object in the &LEVEL0 variable.

```
&LEVEL0 = GetLevel0();
```

The level zero rowset contains all the rows, rowsets, records, and fields underneath it.

If the level zero rowset is formed from component buffer data, then the level zero rowset has one row of data and that row contains all the child rowsets, which in turn contain rows of data that contain other child rowsets.

If the level zero rowset is formed from buffer data, such as from an application message, then the level zero rowset may contain more than one row of data. Each row of the level zero rowset contains all the child rowsets associated with that row, which in turn contain rows of data that contain other child rowsets.

Use a level zero rowset when you want an absolute path to a lower-level object or to do some processing on the entire data buffer. For example, suppose you load all new data into the component buffers and want to redraw the page. You could use the following code:

```
/* Do processing to reload Component Buffers */
&LEVEL0 = GetLevel0();
&LEVEL0.Refresh();
```

Rowset Object

The following code instantiates a rowset object that references the rowset that contains the currently running PeopleCode program:

```
&ROWSET = GetRowset();
```

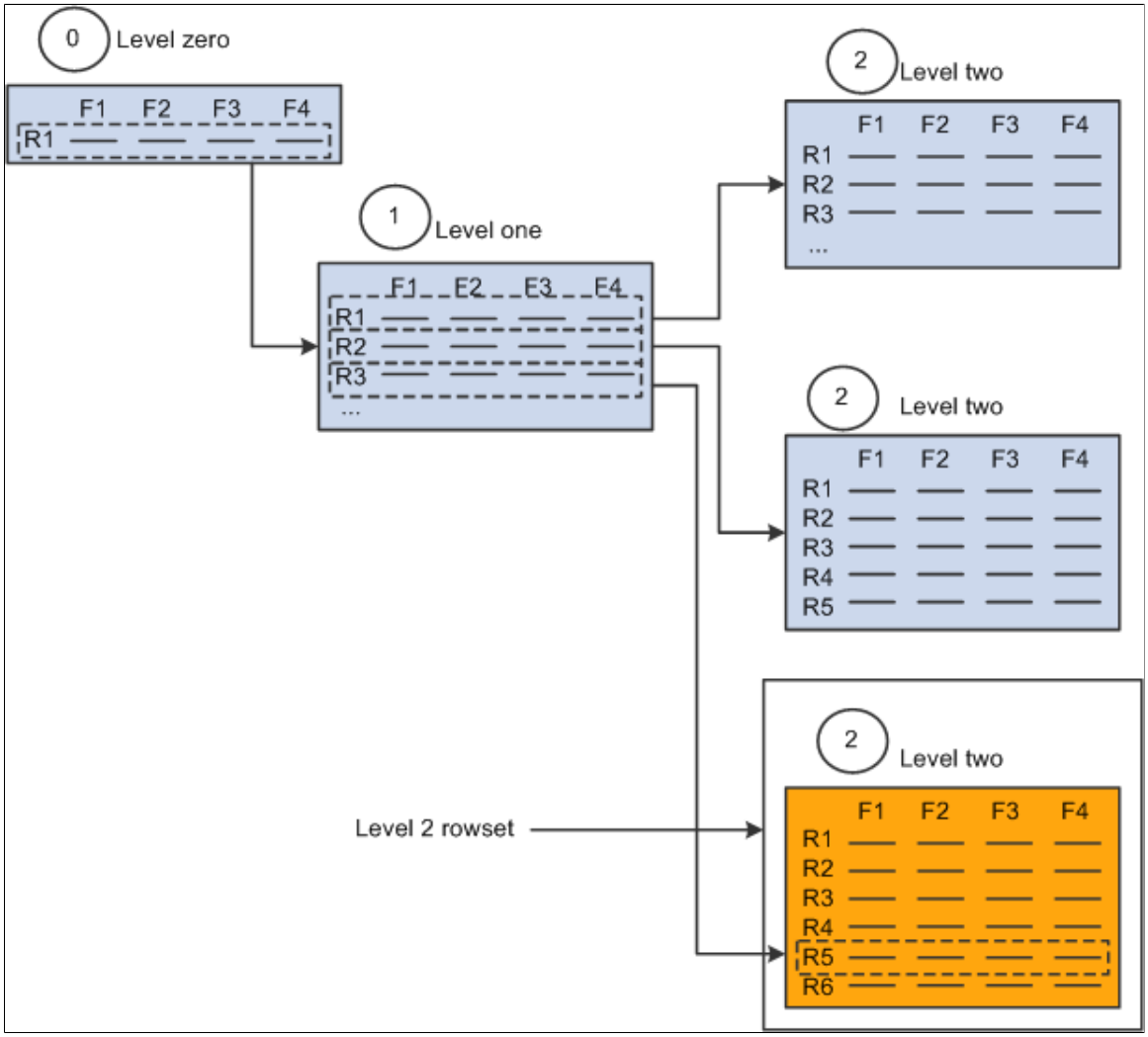
You might later use the &ROWSET variable and the ActiveRowCount property to iterate over all the rows of the rowset, to access a specific row (using the GetRow method), or to hide a child rowset (by setting the Visible property).

The level one rowset contains all the level two rowsets. However, the level two rowsets can only be accessed using the different rows of the level one rowset. From the level zero or level one rowset, you can only access a level two rowset by using the level one rowset and the appropriate row.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Level two rowset from level one row

For example, suppose your program is running on some field of row five of a level two scroll area, which is on row three of its level one scroll area. The resulting rowset contains all the rows of the level two scroll area that are under the row three of the level one scroll area. The rowset does not contain any data that is under any other level two scroll areas. The following diagram illustrates these results:



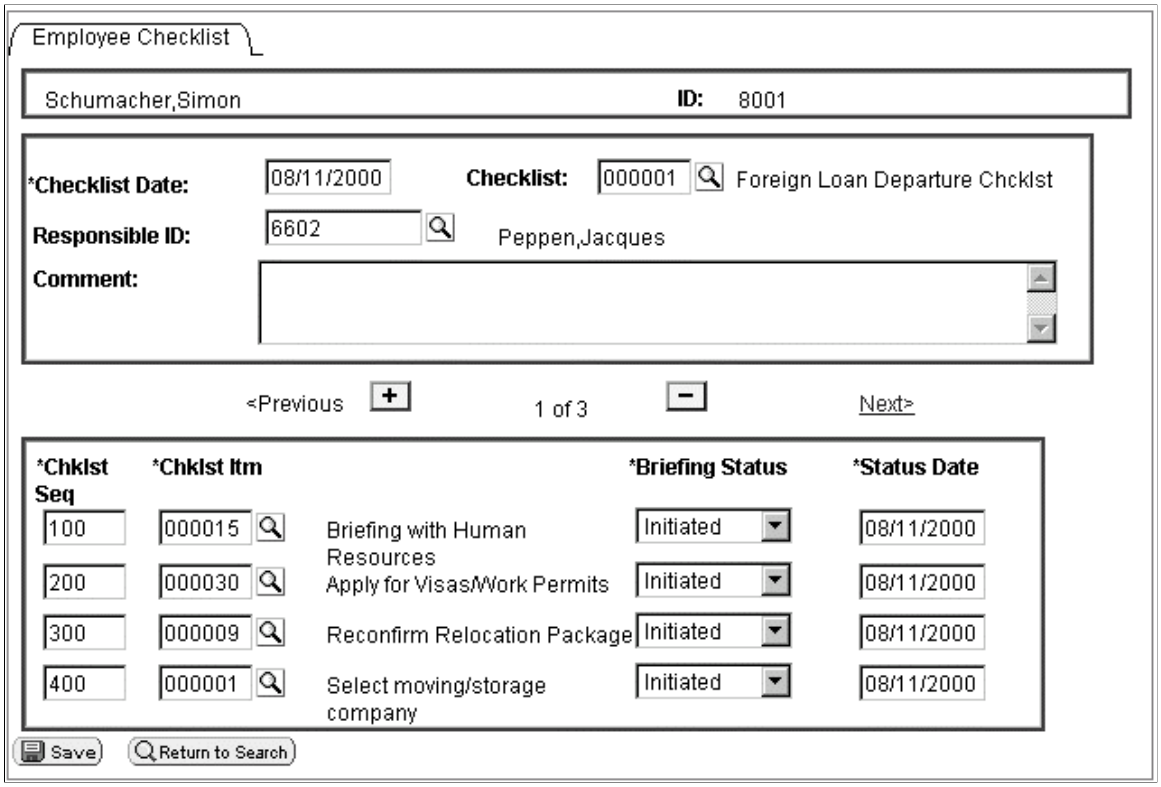
A further illustration uses an example from the Employee Checklist page.

Suppose that one employee was associated with three different checklists: Foreign Loan Departure, Foreign Loan Arrival, and Foreign Loan Host. The checklist code field (CHECKLIST_CD) on the first level of the page drives the entries on the second level. Each row in the level one rowset produces a different level two rowset.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: EMPLOYEE_CHECKLIST Foreign Loan Departure checklist

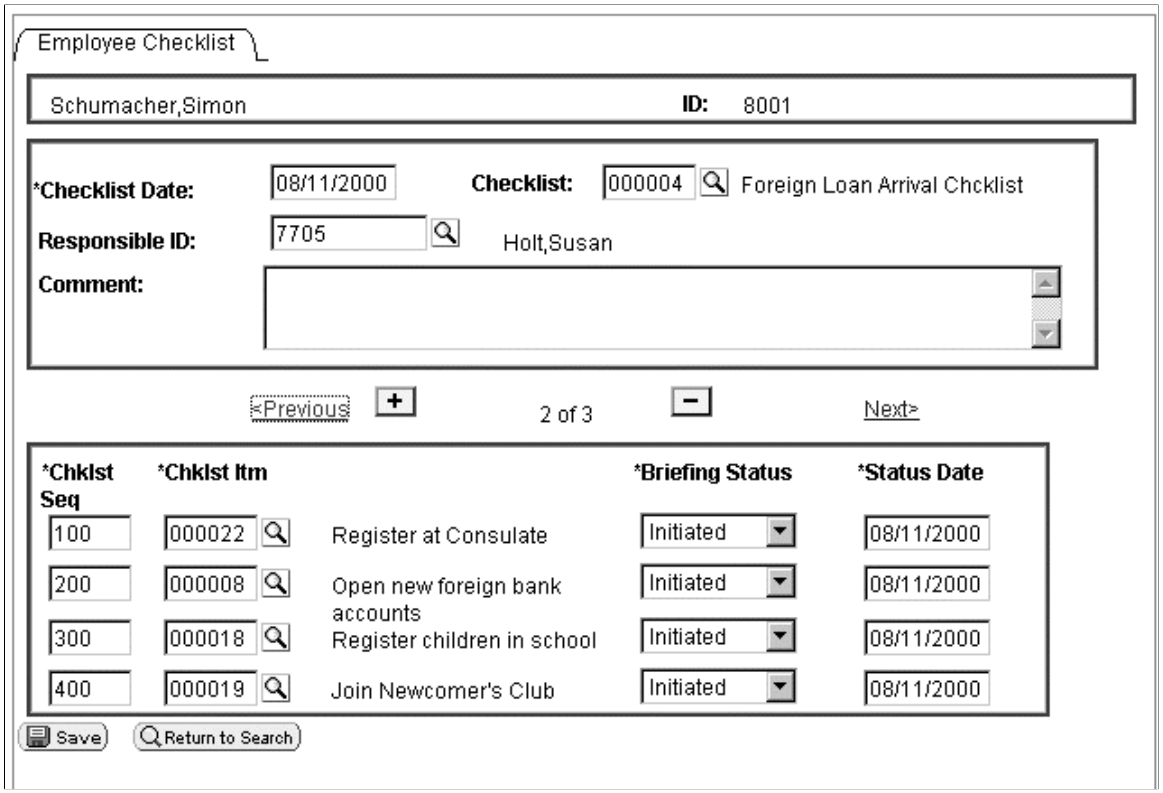
The Foreign Loan Departure checklist (000001) produces a checklist that contains such items as Briefing with Human Resources and Apply for Visas/Work permits, as shown in the following example:



 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: EMPLOYEE_CHECKLIST Foreign Load Arrival Checklist

The Foreign Loan Arrival checklist (0000004) produces a checklist that contains items such as Register at Consulate and Open New Foreign Bank Accounts, as shown in the following example:



Row Object

When you create a page, you put fields from different records onto the page. You can think of this as creating a type of pseudo-SQL join. The row returned from this pseudo-join is a row object.

For example, the first level scroll area of the EMPLOYEE_CHECKLIST page contains the following fields, associated with these records:

Field	*Record*
CHECKLIST_DT	EMPL_CHECKLIST
CHECKLIST_CD	EMPL_CHECKLIST
COMMENTS	EMPL_CHECKLIST
DESCR	CHECKLIST_TBL
NAME	PERSONAL_DATA
RESPONSIBLE_ID	EMPL_CHECKLIST

Copyright © 1988, 2021, Oracle and/or its affiliates.

The pseudo-SQL join might look like this:

```
JOIN A.CHECKLIST_DT, A.CHECKLIST_CD, A.COMMENTS, B.DESCR, C.NAME, A.RESPONSIBLE_ID
FROM PS_EMPL_CHECKLIST A, PS_CHECKLIST_TBL B, PS_PERSONAL_DATA C, WHERE. . .
```

What goes into the Where clause is determined by the level zero of the page. For our example, the value is `WHERE EMPLID=8001.`

When the component is opened, data is loaded into the component buffers. Any row returned by the pseudo-SQL statement is a level one row object. The following table shows a returned row:

CHECKLIST_DT	*CHECKLIST_CD*	*COMMENTS*	*DESCR*	*NAME*	*RESPONSIBLE_ID*
12/03/98	000001		Foreign Loan Department Checklist	Peppen, Jacques	6602

Record Object

A record definition is a definition of what your underlying SQL database tables look like and how they process data. After you create record definitions, you build the underlying SQL tables that contain the application data that your users enter online in your production environment.

When you create a record object using the CreateRecord function, you are creating an area in the data buffers that has the same structure as the record definition, but no data.

When you instantiate a record object from the Record class using some variation of GetRecord, that record object references a single row of data in the SQL table.

Note: The data in the record that you retrieve is based on the row, which is analogous to setting keys to return a unique record.

The following code instantiates a record object for referencing the EMPL_CHECKLIST record of the specified row:

```
&REC = &ROW.GetRecord(RECORD.EMPL_CHECKLIST);
```

Using the short method, the following line of code is identical to the previous line:

```
&REC = &ROW.EMPL_CHECKLIST;
```

You might later use the *&REC* variable and the CopyFieldsTo property to copy all like-named fields from one record to another. In the following example, two row objects are created, the copy from row (COPYFRMROW) and the copy to row (COPYTROW). Using these rows, like-named fields are copied from CHECKLIST_ITEM to EMPL_CHKLST_ITM.

```
For &I = 1 To &ROWSET1.ActiveRowCount
   &COPYFRMROW = &ROWSET1.GetRow(&I);
   &COPYTROW = &RS2.GetRow(&I);
   &COPYFRMROW.CHECKLIST_ITEM.CopyFieldsTo(&COPYTROW.EMPL_CHKLST_ITM);
End-For;
```

A row may contain more than one record: in addition to the primary database record, you may have a related display record or a derived record. You can access these records as well. The level one rowset,

 Copyright © 1988, 2021, Oracle and/or its affiliates.

&ROWSET1, is made up of many records. The following accesses two of them: EMPL_CHECKLIST and DERIVED_HR.

```
&REC1 = &ROW.EMPL_CHECKLIST;
&REC2 = &ROW.DERIVED_HR;
```

Field Object

The following instantiates a field object, from the Field class, that is used to access a specific field in the record:

```
&FIELD = &REC.GetField(FIELD.CHECKLIST_CD);
```

You might later use the *&FIELD* variable as a condition:

```
If ALL(&FIELD) Then
```

Here is another example:

```
If &FIELD.Value = "N" Then
```

Note: The data in the field that you retrieve is based on the record, which is in turn based on the row.

You can also set the value of a field. Using the GetField function does not create a copy of the data from the component buffer. Setting the value or a property of the field object sets the actual component buffer field or property.

See Assigning Objects.

In the following example, the type of field is verified, and the value is replaced with the tangent of that value if it is a number

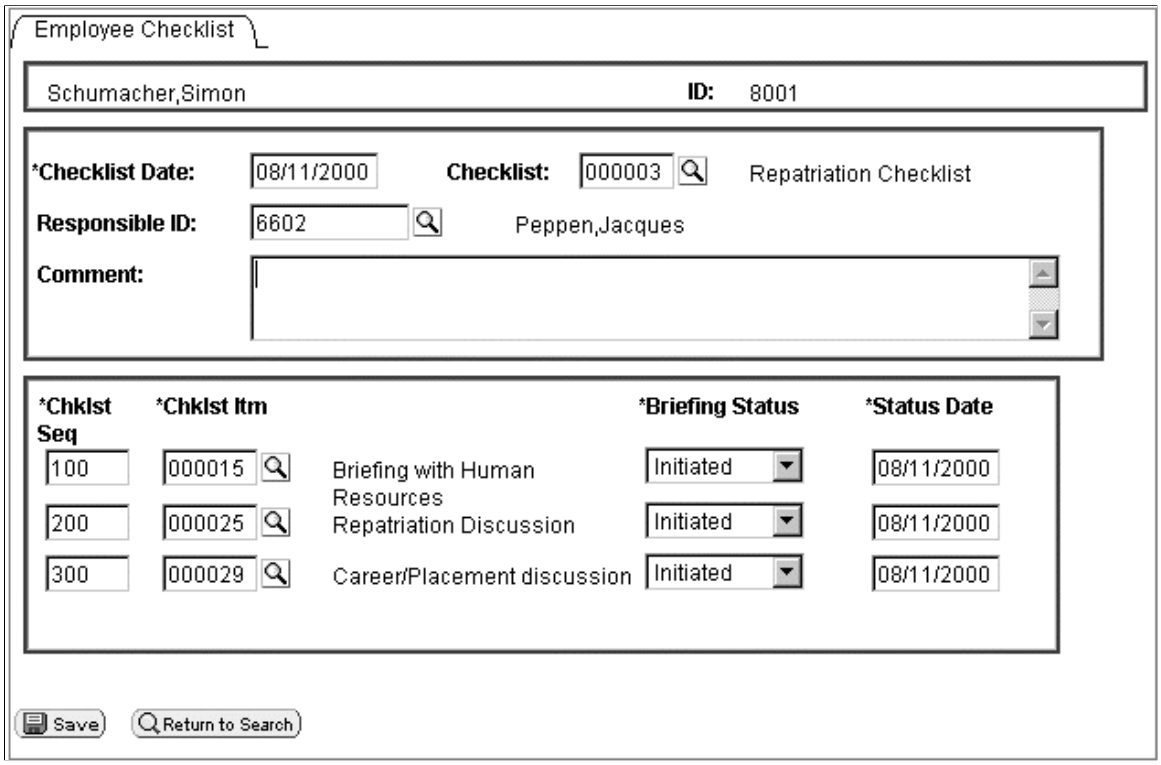
```
If &FIELD.Type <> "NUMBER" Then
   /* do error recording */
Else
   &FIELD.Value = Tan(&FIELD.Value);
End-If;
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

Data Buffer Hierarchy Examples

Image: EMPLOYEE_CHECKLIST repatriation checklist

The following EMPLOYEE_CHECKLIST image is an example of data buffer hierarchy.



Suppose you want to access the BRIEFING_STATUS field at level two of the following page:

First, determine where your code is running. For this example, the code is starting at a field on a record at level zero. However, you do not always have to start at level zero.

If you start with level zero, you must traverse the data hierarchy, through the level one rowset to the level two rowset, before you can access the record that contains the field.

Obtaining the Rowset

You first obtain the level zero rowset, which is the PERSONAL_DATA rowset. You do not need to know the name of the level zero rowset to access it:

```
&LEVEL0 = GetLevel0();
```

Obtaining Rows

The next object to get is a row. As the following code is working with data that is loaded from a page, only one row is at level zero. However, if you have rowsets that are populated with data that is not based on component buffers (for example, an application message), you may have more than one row at level zero.

```
&LEVEL0_ROW = &LEVEL0(1);
```


Copyright © 1988, 2021, Oracle and/or its affiliates.

Obtaining Child Rowsets

To obtain the level two rowset, traverse through the level one rowset first. Therefore, the next object to get is the level one rowset, as shown in the following example:

```
&LEVEL1 = &LEVEL0_ROW.GetRowset(SCROLL.EMPL_CHECKLIST);
```

Obtaining Subsequent Rows

If you are traversing a page, obtain the appropriate row after you get a rowset. To process all the rows of the rowset, set this functionality up in a loop, as shown in the following example:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   . . .
End-For;
```

Obtaining Subsequent Rowsets and Rows

Traverse another level in the page structure to access the second level rowset, and then use a loop to access the rows in the level two rowset.

Because we are processing all the rows at level one, we are just adding code to the previous For loop. As we process through all the rows at level two, we are adding a second For loop. The new code is in bold in the following example:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   &LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.
EMPL_CHKLST_ITM);
   For &J = 1 to &LEVEL2.ActiveRowCount
      &LEVEL2_ROW = &LEVEL2(&J);
   . . .
   End-For;
End-For;
```

Obtaining Records

A row always contains a record, and it may contain only a child rowset, depending on how your page is set up. GetRecord is the default method for a row, so all you have to specify is the record name.

Because we are processing all the rows at level two, we just add code to the For loops of the previous example. The new code is in bold:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   &LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
   For &J = 1 to &LEVEL2.ActiveRowCount
      &LEVEL2_ROW = &LEVEL2(&J);
      &RECORD = &LEVEL2_ROW.EMPL_CHKLST_ITM;
   . . .
   End-For;
End-For;
```

Obtaining Fields

Records are made up of fields. GetField is the default method for a record, so all you have to specify is the field name.

Copyright © 1988, 2021, Oracle and/or its affiliates.

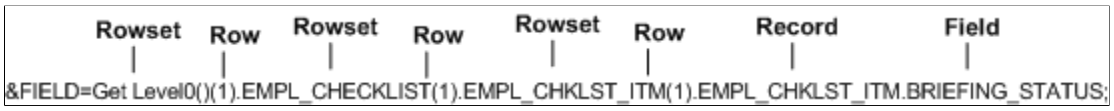
Because we are processing all the rows at the level one, we are just adding code to the For loops of the previous example. The new code is in bold:

```
For &I = 1 to &LEVEL1.ActiveRowCount
   &LEVEL1_ROW = &LEVEL1(&I);
   &LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
   For &J = 1 to &LEVEL2.ActiveRowCount
      &LEVEL2_ROW = &LEVEL2(&J);
      &RECORD = &LEVEL2_ROW.EMPL_CHKLST_ITM;
      &FIELD = &RECORD.BRIEFING_STATUS;
      /* Do processing */
   End-For;
End-For;
```

Using Shortcuts

Image: Rowset example

The previous code is the long way of accessing this field. The following example uses shortcuts to access the field in one line of code. The following code assumes all rows are level one:



Here's another method of expressing the code:

Object Type	***Code***
Rowset	&LEVEL0 = GetLevel0();
Row	&LEVEL0_ROW = &LEVEL0(1);
Rowset	&LEVEL1 = &LEVEL0_ROW.GetRowset(SCROLL.E⇒ MPL_CHECKLIST);
	For &I = 1 to &LEVEL1.ActiveRowCount
Row	&LEVEL1_ROW = &LEVEL1(&I);
Rowset	&LEVEL2 = &LEVEL1_ROW.GetRowset(SCROLL.E⇒ MPL_CHKLST_ITM);
	For &J = 1 to &LEVEL2.ActiveRowCount⇒
Row	&LEVEL2_ROW = &LEVEL2(&J);
Record	&RECORD = &LEVEL2_ROW.EMPL_CHKLST⇒ _ITM;

Copyright © 1988, 2021, Oracle and/or its affiliates.

Object Type	*Code*
Field	&FIELD = &RECORD.BRIEFING_STATUS;⇒
	/* Do processing */
	End-For;
	End-For;

Rowset Examples

The following code example traverses up to four levels of rowsets and could easily be modified to do more. This example only processes the first record in every rowset. To process every record, set up another For loop (For &R = 1 to &LEVELX.RECORDCOUNT, and so on). Notice the use of the ChildCount function (to process all children rowsets within a rowset), ActiveRowCount, IsChanged, and dot notation.

In the following example, ellipses indicate where application-specific code should go.

```
&Level0_ROWSET = GetLevel0();
For &A0 = 1 To &Level0_ROWSET.ActiveRowCount

    ...

/*************************/
/* Process Level 1 Records */
/*-------------------------*/
   If &Level0_ROWSET(&A0).ChildCount > 0 Then
   For &B1 = 1 To &Level0_ROWSET(&A0).ChildCount
      &LEVEL1_ROWSET = &Level0_ROWSET(&A0).GetRowset(&B1);
      For &A1 = 1 To &LEVEL1_ROWSET.ActiveRowCount
      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 */
/*---------------------------------*/
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
/* 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 */
/*********************************/

End-For; /* A0 - Loop */
```

Hidden Work Scroll Example

In the FieldChange event for the CHECKLIST_CD field on the EMPL_CHECKLIST record, a PeopleCode program does the following:

1. Flushes the rowset and hidden work scroll area.
2. Performs a Select statement on the hidden work scroll area based on the value of the CHECKLIST_CD field and the effective date.
3. Clears the level two scroll area.
4. Copies like-named fields from the hidden work scroll area to the level two scroll area.

The following example shows how to do this using built-in functions.

```
&CURRENT_ROW_L1 = CurrentRowNumber(1);

&ACTIVE_ROW_L2 = ActiveRowCount(RECORD.EMPL_CHECKLIST,
&CURRENT_ROW_L1, RECORD.EMPL_CHKLST_ITM);

If All(CHECKLIST_CD) Then
   ScrollFlush(RECORD.CHECKLIST_ITEM);
   ScrollSelect(1, RECORD.CHECKLIST_ITEM, RECORD.CHECKLIST_ITEM,
"Where Checklist_Cd = :1 and EffDt = (Select Max(EffDt) From
PS_Checklist_Item Where Checklist_Cd = :2)",
CHECKLIST_CD, CHECKLIST_CD);

   &FOUNDDOC = FetchValue(CHECKLIST_ITEM.CHKLST_ITEM_CD, 1);
   &SELECT_ROW = ActiveRowCount(RECORD.CHECKLIST_ITEM);

   For &I = 1 To &ACTIVE_ROW_L2
      DeleteRow(RECORD.EMPL_CHECKLIST, &CURRENT_ROW_L1, RECORD.EMPL_CHKLST_ITM, 1);
   End-For;

   If All(&FOUNDDOC) Then
      For &I = 1 To &SELECT_ROW
         CopyFields(1, RECORD.CHECKLIST_ITEM, &I, 2,
RECORD.EMPL_CHECKLIST, &CURRENT_ROW_L1, RECORD.EMPL_CHKLST_ITM, &I);
         If &I <> &SELECT_ROW Then
            InsertRow(RECORD.EMPL_CHECKLIST, &CURRENT_ROW_L1,
RECORD.EMPL_CHKLST_ITM, &I);
         End-If;
      End-For;
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
   End-If;
End-If;
```

The following example performs the same function as the previous code, only it uses the data buffer classes:

1. Flushes the rowset and hidden work scroll area (&RS1H).
2. Performs a Select statement on &RS1H based on the value of the CHECKLIST_CD field and the effective date.
3. Clears the level two rowset (&RS2).
4. Copies like-named fields from &RS1H to &RS1.

```
Local Rowset &RS0, &RS1, &RS2, &RS1H;

&RS0 = GetLevel0();
&RS1 = GetRowset();
&RS2 = GetRowset(SCROLL.EMPL_CHKLST_ITM);
&RS1H = &RS0.GetRow(1).GetRowset(SCROLL.CHECKLIST_ITEM);

&MYFIELD = CHECKLIST_CD;

If All(&MYFIELD) 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);

   For &I = 1 To &RS2.ActiveRowCount
      &RS2.DeleteRow(1);
   End-For;

&FOUND = &RS1H.GetCurrEffRow().CHECKLIST_ITEM. CHKLST_ITEM_CD.Value;

   If All(&FOUND) Then
      For &I = 1 To &RS1H.ActiveRowCount
         &COPYFRMROW = &RS1H.getrow(&I);
         &COPYTROW = &RS2.getrow(&I);
         &COPYFRMROW.CHECKLIST_ITEM.CopyFieldsTo(&COPYTROW.EMPL_CHKLST_ITM);
         If &I <> &RS1H.ActiveRowCount Then
            &RS2.InsertRow(&I);
         End-If;
      End-For;
   End-If;
End-If;
```

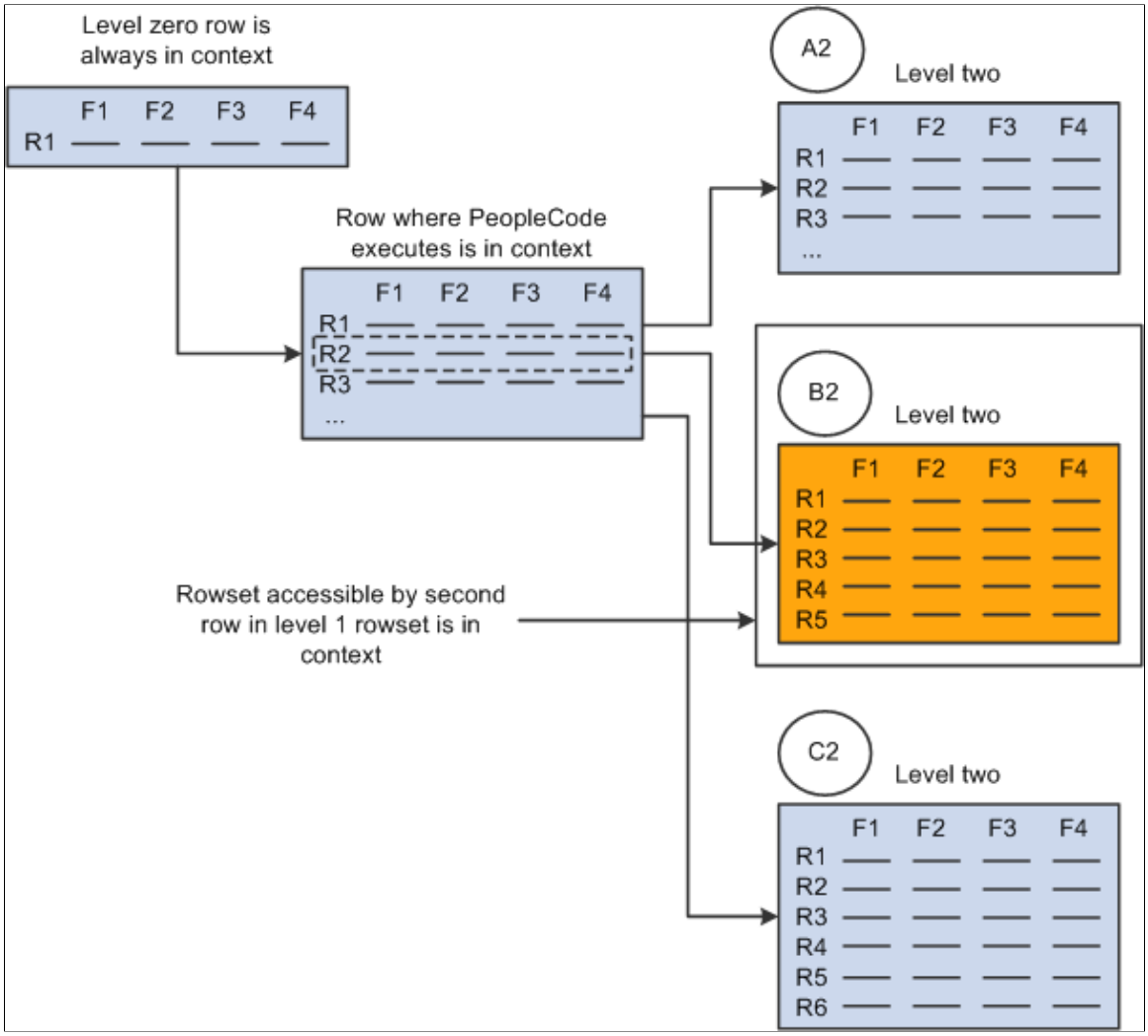
Understanding Current Context

Most PeopleCode programs run in a current context. The current context determines which buffer fields can be contextually referenced from PeopleCode, and which row of data is the current row on each scroll level at the time a PeopleCode program is running.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Current context for rowsets

The current context for the data buffer access classes is similar to the current context for accessing the component buffer, as shown in the following diagram:



In this example, a PeopleCode program is running in a buffer field on the second row of the level one rowset. The following code returns a row object for the second row of the level one rowset, because that is the row that is the current context.

```
Local Row &ROW

&ROW = GetRow();
```

The following code returns the B2 level two rowset because of the current context:

```
Local Rowset &ROWSET2

&ROWSET2 = &ROW.GetRowset(SCROLL.EMPL_CHKLST_ITM);
```

This code does not return either the C2 or the A2 rowsets. It returns only the rowset associated with the second row of the level one rowset.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Creating Records or Rowsets and Current Context

When you instantiate a record object using the CreateRecord function, you are only creating an area in the data buffers that has the same structure as the record definition. It does not contain any data. This record object does not have a parent rowset and is not associated with a row. It is a freestanding record object and, therefore, is not considered part of the current context.

The same concept applies when you instantiate a rowset object using the CreateRowset function. You are only creating an area in the data buffers that has the same structure as the records or rowset that the new rowset is based on. The rowset does not contain any data. This type of rowset does not have a parent rowset or row.

Related Links

Specifying Data with Contextual References

"CreateRecord" (PeopleTools 8.59: PeopleCode Language Reference)

"CreateRowset" (PeopleTools 8.59: PeopleCode Language Reference)

Accessing Secondary Component Buffer Data

When a secondary page is run, the data for its buffers is copied from the parent component to a buffer structure for the secondary page. That is, two copies of this data are made. The data buffer classes give access to both of these copies of the data. Direct field references (*recname.fieldname*) always use the current context to determine which value to access. So, in general, when using a secondary page, make sure that references are based on the secondary page.

Instantiating Rowsets Using Non-Component Buffer Data

Both the application message and the file layout technologies represent hierarchical data, and both use the rowset, row, record, and field hierarchy. Though you use different methods to instantiate a rowset object for this data, you still use the same rowset, row, record, and field methods and properties to manipulate the data. (Any exceptions are marked in the documentation.)

To instantiate a rowset for a message:

```
&MSG = CreateMessage(OPERATION.EMPLOYEE_DATA);
&MYROWSET = &MSG.GetRowset();
```

To instantiate a rowset for a file layout:

```
&MYFILE = GetFile(&SOMENAME, "R", "UTF8");

If &MYFILE.IsOpen Then
     &MYFILE.SetFileLayout(FILELAYOUT.SOMELAYOUT);
     &MYROWSET = &MYFILE.ReadRowset();
End-if;
```

In an Application Engine program, the default state record is considered the primary record and the main record in context. You can access the default state record using the following:

```
&STATERECORD = GetRecord();
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

If you have more than one state record associated with an Application Engine program, you can access them the same way you would access other, nonprimary data records, by specifying the record name. For example:

```
&ALTSTATE = GetRecord(RECORD.AE_STATE_ALT);
```

Related Links

Using Standalone Rowsets

"Understanding Managing Messages" (PeopleTools 8.59: Integration Broker)

"Understanding Application Engine Meta-SQL" (PeopleTools 8.59: Application Engine)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 6

PeopleCode and the Component Processor

Understanding the Component Processor

The Component Processor is the PeopleTools runtime engine that controls processing of an application from the time that a user requests a component from an application menu until the database is updated and processing of the component is complete.

Events Outside the Component Processor Flow

An Application Engine program can have a PeopleCode program as an action. Though the right-hand drop-down list box on the PeopleCode Editor window displays the text *OnExecute,* the PeopleCode program really is not an event. Any PeopleCode contained in an Application Engine action is executed only when the action is executed.

A component interface can have user-defined methods associated with it. These methods are not part of any processor flow; they are called as needed by the program executing the component interface.

Security has a signon event during sign-in. This is actually PeopleCode programs on a record field that you have specified in setting up security.

Though application packages have a right-hand drop-down list box on the PeopleCode Editor window that displays the text *OnExecute,* this is not an event. Any PeopleCode contained in the application class is only executed when called explicitly in a PeopleCode program.

Related Links

"Understanding Component Interface Class" (PeopleTools 8.59: PeopleCode API Reference)
"Understanding Application Classes" (PeopleTools 8.59: PeopleCode API Reference)
"Specifying PeopleCode Actions" (PeopleTools 8.59: Application Engine)
"PeopleSoft Sign In" (PeopleTools 8.59: Security Administration)

PeopleCode Program Triggers

This section provides an overview of PeopleCode program triggers and discusses how to:

- Access PeopleCode programs.
- Associate execution order of events and PeopleCode.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Understanding PeopleCode Program Triggers

PeopleCode can be defined on events associated with a record field, a component, a component record, and many other definitions. During the course of the component processor's flow of execution, these PeopleCode events (or exit points) are encountered in a specific sequence. When an event is encountered, the component processor runs any PeopleCode for that event on each definition.

The following definitions have events that are part of the component processor flow:

Items	***Event Triggers***
Components	Programs associated with a component definition.
Component records	Programs associated with a component record.
Component record fields	Programs associated with a component record field.
Pages	Programs associated with a page definition.
Record fields	Programs associated with a specific field of a record definition.
Menu items	Programs associated with a menu item.

Suppose a user changes the data in a page field, and then presses Tab to move out of the field. This user action initiates the FieldEdit PeopleCode event. The FieldEdit event affects only the field and row where the change took place.

If you have two FieldEdit PeopleCode programs, one associated with the record field and a second associated with the component record field, both programs execute, but only on the specific field and row of data. The FieldEdit PeopleCode program associated with the record field is executed first, and then the FieldEdit PeopleCode program associated with the component record field is executed.

By contrast, suppose a user has opened a component for updating. As part of building the component, the component processor encounters the RowInit event. This event triggers any RowInit PeopleCode programs on every record field on every row of data in the component. In a scroll area with multiple rows of data, each RowInit PeopleCode program is executed once for each row.

In addition, if you have RowInit PeopleCode associated with both the record field and the component record, both programs are executed on every row of data in the component. The RowInit PeopleCode program associated with the record field is executed first, and then the RowInit PeopleCode program associated with the component record is executed. If you set the value of a field with the record field RowInit PeopleCode, and then reset the field with the component record RowInit PeopleCode, the second value appears to the user.

When you develop with PeopleCode, you must consider when and where your programs are triggered during execution of the component processor flow in addition to what that program executes.

This section discusses how to:

- Access PeopleCode programs.
- Understand the execution order of events and PeopleCode.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

Execution Order of Events and PeopleCode

Accessing PeopleCode Programs

Every PeopleCode program is associated with a PeopleCode event and is often referred to by that name, such as RowInit PeopleCode or FieldChange PeopleCode. These programs are accessible from, and associated with, different items. The following table lists items and associated events.

Note: During search processing in update modes or add mode, the SearchInit and SearchSave events (in the Component Record column of the table) are available only for the search record associated with a component.

Component Events	*Component Record Events*	*Component Record Field Events*	*Page Events*	*Record Field Events*	*Menu Events*
PostBuild PreBuild SavePostChg SavePreChg Workflow	RowDelete RowInit RowInsert RowSelect SaveEdit SavePostChg SavePreChg SeachInit SearchSave	FieldChange FieldDefault FieldEdit PrePopup	Activate	FieldChange FieldDefault FieldEdit FieldFormula PrePopup RowDelete RowInit RowInsert RowSelect SaveEdit SavePostChg SavePreChg SearchInit SearchSave Workflow	ItemSelected

The following table lists types of PeopleCode programs and where to access them in Application Designer.

PeopleCode Programs	*Location in Application Designer*
Components*, component records*, and component record fields	Component definitions
Pages	Page definitions

Copyright © 1988, 2021, Oracle and/or its affiliates.

PeopleCode Programs	*Location in Application Designer*
Record fields	Record definitions
Menu items	Menu definitions

* PeopleCode programs can be associated with components and component records outside of Application Designer. The PeopleSoft Related Content Framework can be used to map application class PeopleCode programs to component and component record events. This allows custom PeopleCode programs to be defined for a component without customizing the component definition in Application Designer. These custom PeopleCode programs can be configured to run before or after any PeopleCode program defined for the same event from the component definition.

Note: While the PeopleSoft Related Content Framework is used to complete this configuration, these PeopleCode programs do not constitute or render as related content.

Related Links

"Mapping Application Class PeopleCode to Component Events " (PeopleTools 8.59: Portal Technology)

Execution Order of Events and PeopleCode

In PeopleSoft systems, the component is the representation of a transaction. Therefore, any PeopleCode that is associated with a transaction should be in events associated with some level of the component itself (component, component record, or component record field). If you associate code with the correct transaction, you do not have to check for the component that is issuing it (such as surrounding your code with dozens of `If %Component =` statements). Consequently, this makes record definitions more reusable, and your code is more maintainable. However, code that should be executed every time a field is edited should be at the record field level and not placed on the component.

For example, if you have start and end dates for a course, you would always want to make sure that the end date was after the start date. Your program to check the dates would go on the SaveEdit at the record field level.

All similarly named component events are executed after the like-named record event. The PeopleCode program associated with the record field event is executed first, and then the PeopleCode program associated with the like-named component event is executed. If you set the value of a field with the record field PeopleCode, and then reset the field with like-named component PeopleCode, the second value is displayed to the user.

Events After Field Changes

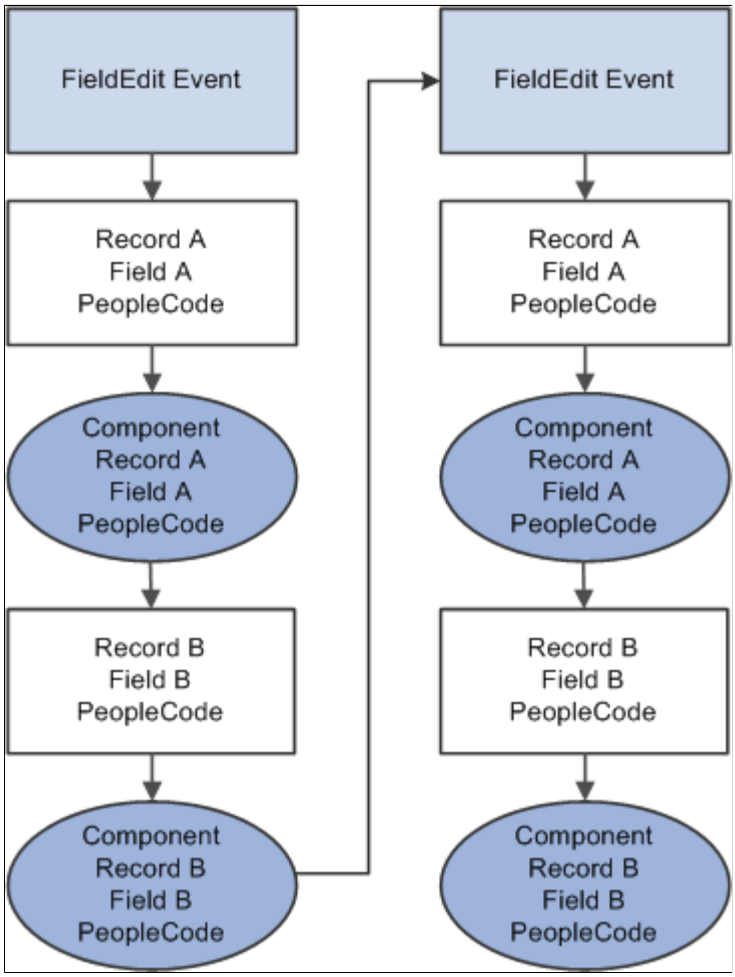
The following events occur after a user changes a field:

```
Record.recordA.fielda.FieldEdit -> Component.recordA.fielda.FieldEdit ->
Record.recordB.fieldb.FieldEdit -> Component.recordB.fieldb.FieldEdit ->
Record.recordA.fielda.FieldChange -> Component.recordA.fielda.FieldChange ->
Record.recordB.fieldb.FieldChange -> Component.recordB.fieldb.FieldChange ->
```


Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Flow of events and PeopleCode programs after a user changes a field

The following diagram shows the flow of events and PeopleCode programs after a user changes a field.



Events After User Saves

The following events occur after a user saves:

```
Record.recordA.fielda.SaveEdit ->
Record.recordA.fieldb.SaveEdit ->
Record.recordA.fieldc.SaveEdit ->
Component.recordA.SaveEdit

Record.recordB.fielda.SaveEdit ->
Record.recordB.fieldb.SaveEdit ->
Record.recordB.fieldc.SaveEdit ->
Component.recordB.SaveEdit

Record.recordA.fielda.SavePreChange ->
Record.recordA.fieldb.SavePreChange ->
Record.recordA.fieldc.SavePreChange ->
Component.recordA.SavePreChange

Record.recordB.fielda.SavePreChange ->
Record.recordB.fieldb.SavePreChange ->
Record.recordB.fieldc.SavePreChange ->
Component.recordB.SavePreChange

Record.recordA.fieldA.WorkFlow ->
```

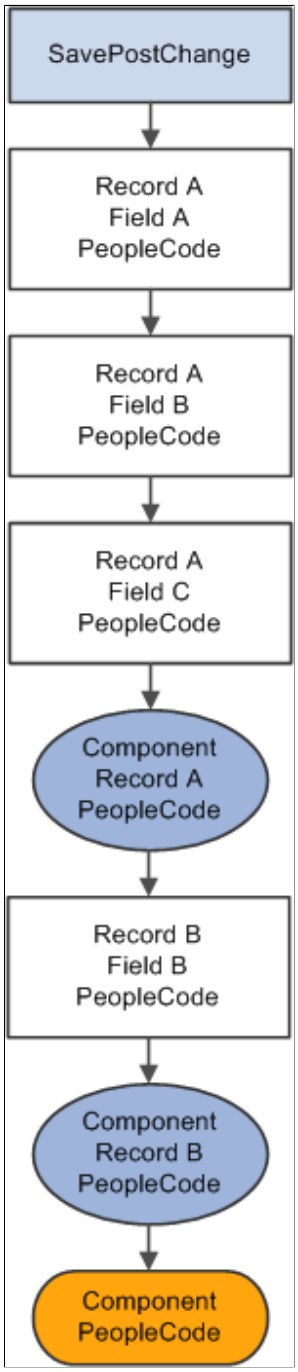
Copyright © 1988, 2021, Oracle and/or its affiliates.

```
Record.recordB.fieldB.WorkFlow ->
Record.reocrdC.fieldC.WorkFlow
Component.Workflow

Record.recordA.fielda.SavePostChange ->
Record.recordA.fieldb.SavePostChange ->
Record.recordA.fieldc.SavePostChange ->
Component.recordA.SavePostChange

Record.recordB.fielda.SavePostChange ->
Component.recordB.SavePostChange
Component.SavePostChange
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Flow of PeopleCode programs after SavePostChange event

The following diagram shows the event flow of PeopleCode programs after SavePostChange event.



Note: SaveEdit does not fire for deleted rows, but SavePreChange, Workflow, and SavePostChange do.

Component Processor Behavior

This section discusses:

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Component Processor behavior from page start to page display.
- Component Processor behavior following user actions in the component.

Note: Components behave differently when run in deferred mode .

Related Links

Deferred Processing Mode
Processing Sequences

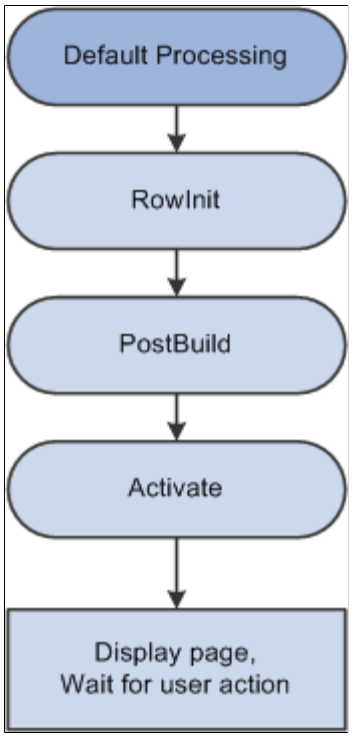
Component Processor Behavior from Page Start to Page Display

Before a user selects a component, the system is in reset state, in which no component is displayed. The Component Processor flow of execution begins when a user selects a component from a PeopleSoft menu. The Component Processor then:

1. Performs search processing, in which it obtains and saves search key values for the component.
2. Retrieves from the database server any data needed to build the component.
3. Builds the component, creating buffers for the component data.
4. Performs any additional processing for the component or the page.
5. Displays the component and waits for user action.

Image: Processing up to Page Display

The following flowchart shows the behavior of a component processor from page start to page display at a high level:



 Copyright © 1988, 2021, Oracle and/or its affiliates.

Component Behavior Following User Actions in the Component

After a component is built and displayed, the Component Processor can respond to a number of possible user actions. The following table lists the user actions and briefly describes the resulting processing:

See Processing Sequences.

User Action	*Description*
Row Insert Processing	When a user requests a row insert, the Component Processor adds a row of data in the active scroll area, then displays the page again and waits for another action. See Row Insert Processing.
Row Delete Processing	When a user requests a row delete, the Component Processor flags the current row as deleted, then displays the page again and waits for another action. See Row Delete Processing.
Field Modification	If a user edits a page field, then leaves the field, the Component Processor performs standard edits (such as checking the data type and checking for values out of range). If the contents of the field do not pass the standard system edits, the Component Processor redisplays the page with an error or warning message and changes the field's color to the system color for field edit errors, usually red. Until the user corrects the error, the Component Processor does not let the user save changes or navigate to another field. If the contents of the field pass the standard system edits, the system redisplays the page and waits for further action. See Field Change Processing.
Prompts	If a user clicks the prompt icon next to a field, a list of values for the prompt field appears. If the Allow Search Events for Prompt Dialogs check box is selected in the record field properties for the search key, the SearchInit event will trigger before the prompt dialog appears. If the user clicks the Look Up button the SearchSave event will trigger. If the end-user clicks the detail button next to a date field, a calendar appears. If the user clicks Return To Search, or presses Alt+2, a search page appears, enabling the user to enter an alternate search key or partial value. See Prompts, Search Processing in Update Modes.
Pop-up Menu Display	If a user clicks the pop-up icon next to a field, a pop-up menu appears. This can be a default pop-up menu or one that has been defined by the developer. If the user clicks the pop-up icon at the bottom of the page, the pop-up menu for the page appears. See Pop-Up Menu Display.

Copyright © 1988, 2021, Oracle and/or its affiliates.

User Action	*Description*
ItemSelected Processing	A user can select an item from a pop-up menu to execute a command. See Selected Item Processing.
Push Button	A user can click a button to execute a command. See Buttons.
Save Processing	A user can direct the system to save a component by clicking Save or by pressing Alt+1. If any component data has been modified, the system also prompts the user to save a component when the Next or List button is clicked, or when a new action or component is selected. The Component Processor first validates the data in the component, and then updates the database with the changed component data. After the update, a SQL Commit command finalizes the changes. See Save Processing.

Processing Sequences

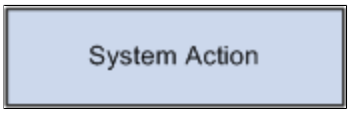
This section presents an overview of flow charts and discusses:

- Default processing.
- Search processing in update mode.
- Search processing in add mode.
- Component build processing in update mode.
- Row select processing.
- Component build processing in add mode.
- Field change processing.
- Row insert processing.
- Row delete processing.
- Buttons.
- Prompts.
- Pop-up menu display.
- Selected item processing.
- Save processing.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Flow Charts

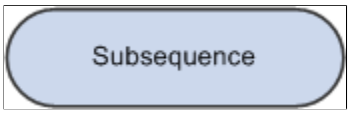
Actions and PeopleCode events can occur in various sequences within the Component Processor's flow of execution. Flow charts represent each sequence. In a flow chart, different shapes and colors represent different concepts.



Blue rectangles represent actions taken by the system.

Dark rhomboids represent branches (decision points) in the logic.

Dark ellipses represent PeopleCode events.



Light ellipses are subprocesses.

Most processing sequences correspond to high-level component processor behaviors. However, two important subsequences occur only in the context of a larger sequence. These subsequences are:

- Default processing, which occurs in a number of different contexts.
- Row select processing, which most commonly occurs as a part of component build in any of the update action modes.

  Row select processing also occurs when a ScrollSelect or related function is executed to load data into a scroll area.

See Component Processor Behavior, Default ProcessingDefault Processing.

Note: Variations may occur in processing sequences, particularly when a PeopleCode function within a processing sequence initiates another processing sequence. For example, if a row of data is inserted or deleted programmatically during the component build sequence, a row insert or row delete sequence is initiated. Also note that components that run in deferred mode behave differently.

See Deferred Processing Mode.

Default Processing

In default processing, any blank fields in the component are set to their default values. You can specify the default value either in the record field properties or in FieldDefault PeopleCode. If no default value is specified, the field is left blank.

Note: In PeopleSoft Pure Internet Architecture, if a user changes a field, but there is nothing to cause a trip to the server on that field, default processing and FieldFormula PeopleCode do not run. They only run when another event causes a trip to the server.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Default processing is relatively complex. The following two sections describe how default processing works on the level of the individual field, and how default processing works in the broader context of the component.

Field-Level Default Processing

During default processing, the Component Processor examines all fields in all rows of the component. On each field, it performs the following:

1. If the field is set to NULL (blank) for a character field, or set to 0 for a numeric field, the Component Processor sets the field to any default value specified in the record field properties for that field.

2. If no default value for the field is defined in the record field properties, then the Component Processor initiates the FieldDefault event, which triggers any FieldDefault PeopleCode associated with the record field or the component record field.

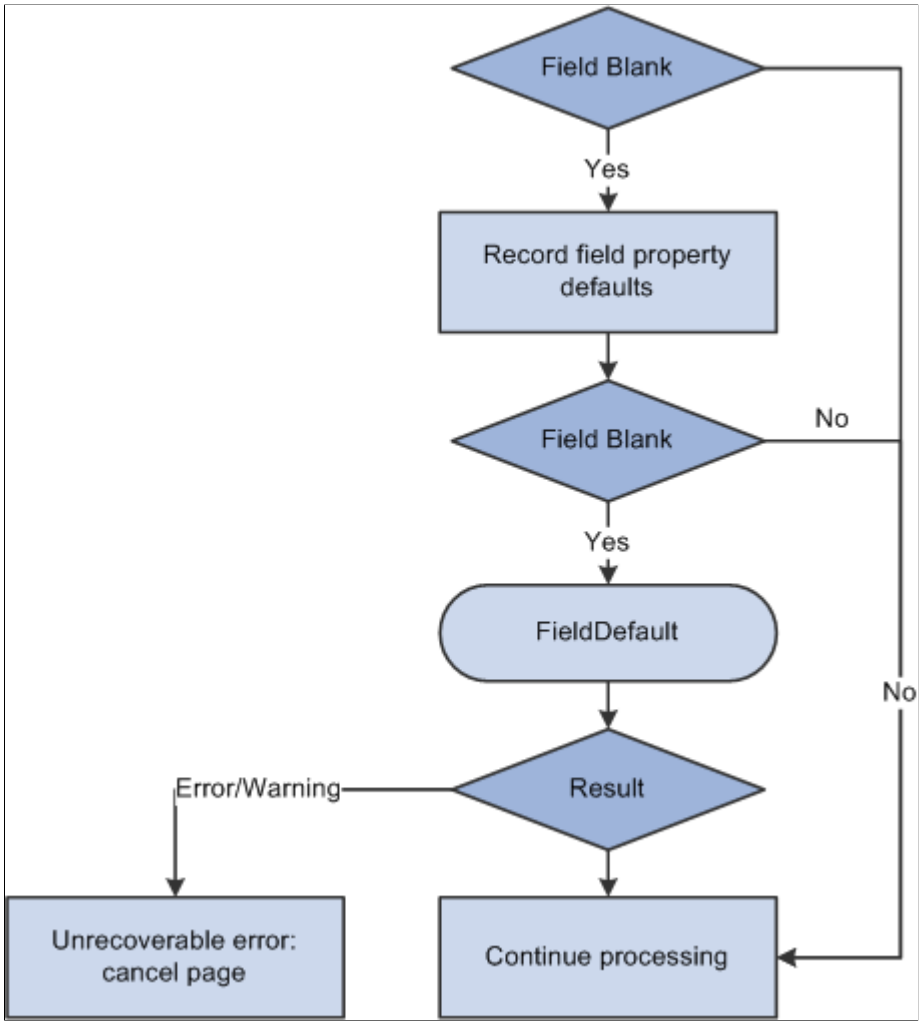
3. If an error or warning executes in any FieldDefault PeopleCode, a runtime error occurs.

Important! Avoid using error and warning statements in FieldDefault PeopleCode.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Field-level default sequence flow

The following flowchart explains the logic of Field-level default processing where in the Component Processor examines all fields in all rows of the component in a sequence flow.



Default Processing on Component Level

Under normal circumstances, default processing in a component is relatively simple: each field on each row of data undergoes field-level default processing. For typical development tasks, this is all you need to be concerned with. However, the complete context of default processing is somewhat more complex.

See Default Processing.

During component-level default processing, the Component Processor performs these tasks:

1. Field-level default processing is performed on all fields on all rows of data in the component.
2. If any field is still blank and any other field in the component has changed, field-level default processing may be repeated, in case a condition changed that causes default processing to now assign a value to something that was previously left blank.
3. The FieldFormula Event is initiated on all fields on all rows of data in the component.

Copyright © 1988, 2021, Oracle and/or its affiliates.

This PeopleCode event is often used for FUNCLIB_ (function library) record definitions to store shared functions, so normally no PeopleCode programs execute.

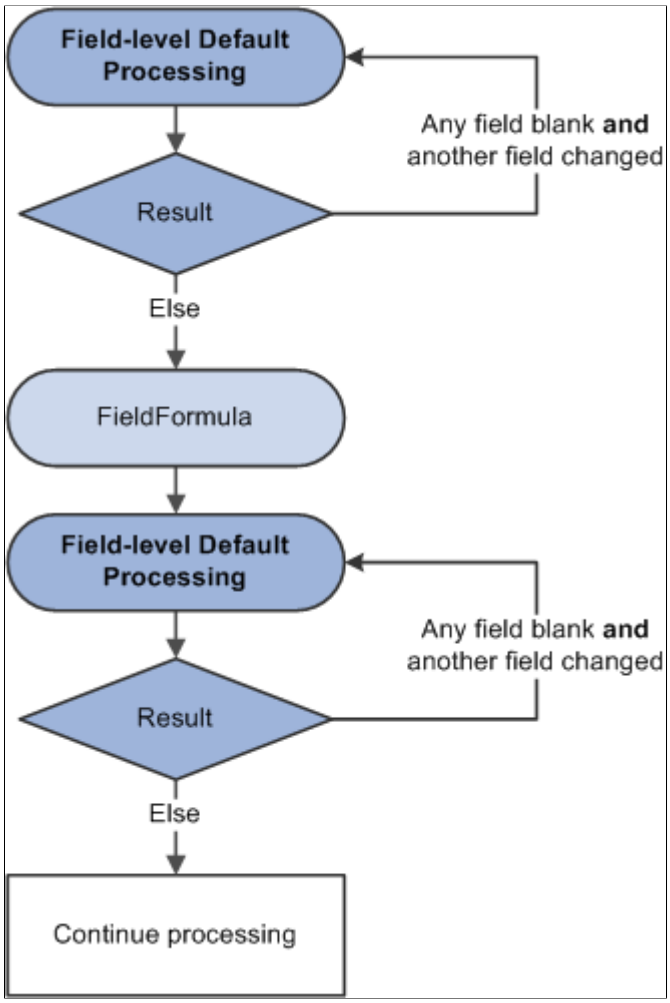
4. If the FieldFormula Event changed anything, field-level default processing is performed again, in case FieldFormula PeopleCode changed a field value to blank, or changed something that causes default processing to now assign a value to a field that was previously left blank.

   Because there should not be any FieldFormula PeopleCode, this is unlikely to affect the development process or performance.

5. If any field is still blank and any other field in the component has changed, field-level default processing is repeated.

Image: Default processing on component level

The following flowchart explains the logic of Default processing on component level.



Search Processing in Update Modes

If a user selects any of the update action modes (Update, Update/Display All, or Correction), the Component Processor begins update mode search processing, which includes the following steps:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

1. The SearchInit PeopleCode event is initiated, which triggers any SearchInit PeopleCode associated with the record field or the component search record, on the keys or alternate search keys in the component search record.

   This enables you to control the search page field values or the search page appearance programmatically, or to perform other processing prior to the appearance of the search page.

   Note: Set the search record for the component in the component properties.

   For example, the following program in SearchInit PeopleCode on the component search key record field EMPLID sets the search key page field to the user's employee ID, makes the page field unavailable for entry, and enables the user to modify the user's own data in the component:

   ```
   EMPLID = %EmployeeId;
   &Field = GetField(EMPLID).Enabled = False;
   AllowEmplIdChg(True);
   ```

   Note: Generally, the system search processing displays the search page. You can use the SearchInit event, and the SetSearchDialogBehavior function, to set the behavior of the search page before it is displayed. If SetSearchDialogBehavior is set to *Force display,* the dialog box is displayed even if all required keys have been provided. You can also set SetSearchDialogBehavior to *skip if possible.* In addition, you can force search processing to always occur by selecting Force Search Processing in the component definition properties in PeopleSoft Application Designer.

   ---

2. The search page and prompt list appear, in which the user can enter search keys or select an advanced search to enter alternate search keys.

   Note: Normally, the values in the search page are not set to default values. However, if the SearchDefault function was executed in SearchInit PeopleCode for any of the search key or alternate search fields, those fields in the dialog box are set to their system default values. No other default processing occurs (that is, the FieldDefault event is not initiated).

   ---

3. The user enters a value or partial value in the search page, and then clicks Search.

4. The SearchSave PeopleCode event is initiated, which triggers any SearchSave PeopleCode associated with the record field or the component search record, on the search keys or alternate search keys in the search record.

   This enables you to validate the user entry in the search page by testing the value in the search record field in PeopleCode and, if necessary, issuing an error or warning. If an error is executed in SearchSave, the user is sent back to the search page. If a warning is executed, the user can click OK to continue or click Cancel to return to the search page and enter new values.

   If partial values are entered, such that the Component Processor can select multiple rows, then the prompt list dialog box is filled, and the user can select a value. If key values from the search page are blank, or if the system cannot select any data based on the user entry in the search page, the system displays a message and redisplays the search page. If the values entered produce a unique value, the prompt list is not filled. Instead, the user is taken directly to the page.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: Normally, no system edits are applied when the user changes a field in the search page. However, if the SearchEdit property is executed for specific search page fields in SearchInit PeopleCode, the system edits are applied to those fields after the user changes a field and either leaves the field or clicks Search. In addition, the SearchEdit property can also be set in metadata for the record field definition.
If the user entry in the field fails the system edits, the system displays a message, highlights the field in question, and returns the user to the dialog box. The FieldEdit and SaveEdit PeopleCode events are not initiated. The SearchSave event is not initiated after values are selected from the search list. To validate data entered in the search page, use the Component PreBuild event.

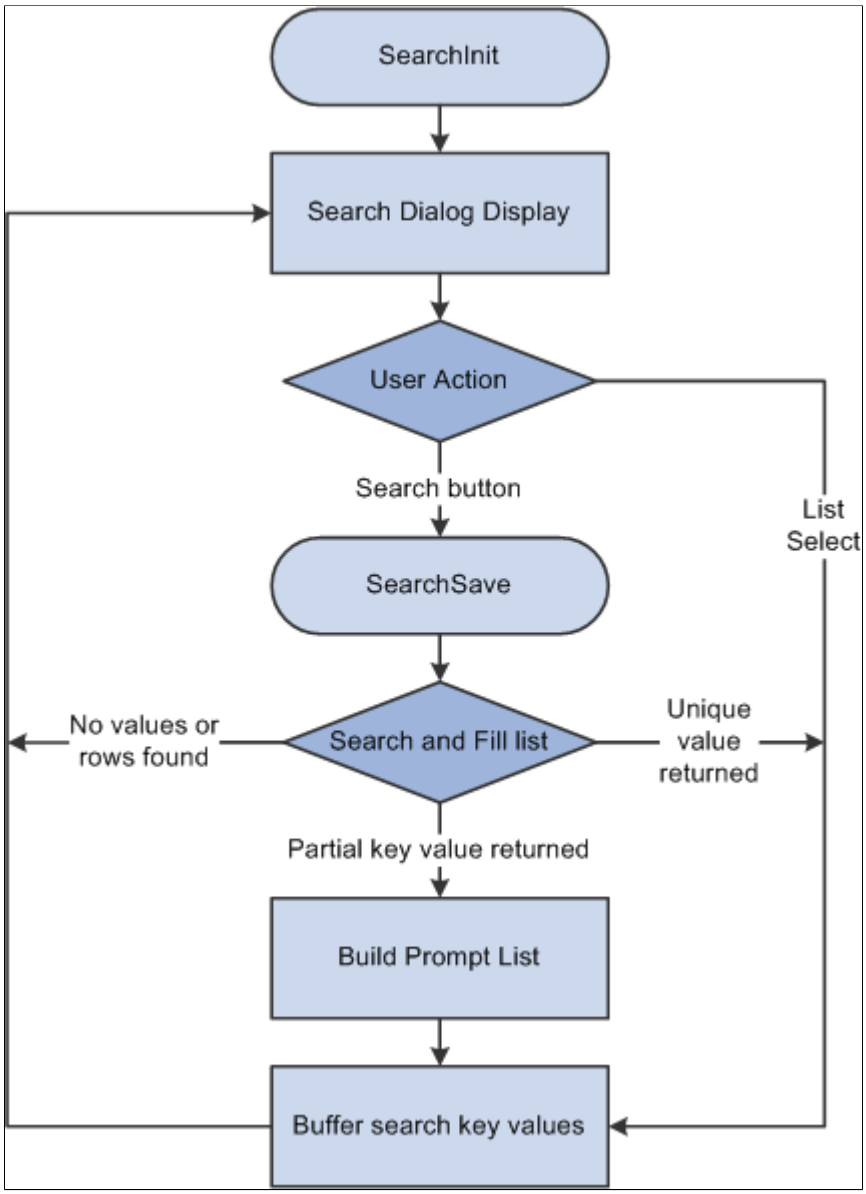
5. The Component Processor buffers the search key values.

   If the user then opens another component while this component is active, the Component Processor uses the same search key values and bypasses the search page.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Search processing logic in update mode

The following flowchart shows this logic. (It does not show the effects of executing the SearchDefault and SearchEdit Field class properties.)



Note: You can use the IsSearchDialog built-in function to create PeopleCode that runs only during search processing. To create processes that run only in a specific action mode, use the %Mode system variable. This could be useful in code that is part of a library function and that is invoked in places other than from the search page. It could also be used in PeopleCode associated with a record field that appears in pages and in the search page.

Related Links

"SetSearchDialogBehavior" (PeopleTools 8.59: PeopleCode Language Reference)

"SearchDefault" (PeopleTools 8.59: PeopleCode API Reference)

"%Mode" (PeopleTools 8.59: PeopleCode Language Reference)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Search Processing in Add Modes

When a user opens a component in add or data-entry modes, the following actions occur:

1. The Component Processor runs default processing on the high-level keys that appear in the Add or Data Entry dialog box.
2. The Component Processor initiates the RowInit event, which triggers any RowInit PeopleCode associated with the record field or the component record, on the Add or Data Entry dialog box fields.
3. The Component Processor initiates the SearchInit event on dialog fields, which triggers any SearchInit PeopleCode associated with the record field or the component search record.

   This enables you to execute PeopleCode programs before the dialog box appears.
4. The Component Processor displays the Add or Data Entry dialog box.
5. If the user changes a dialog box field, and then leaves the field or clicks OK, the following actions occur:

   - In add mode only, a field modification processing sequence occurs.

     See Field Change Processing.
   - Default processing is run on the Add or Data Entry dialog box fields.

     Normally this does not have any effect, because the fields have a value.
6. When the user clicks OK in the dialog box, the SaveEdit event is initiated, which triggers any PeopleCode associated with the record field or the component record.
7. The Component Processor initiates the SearchSave event, which triggers any SearchSave PeopleCode associated with the record field or the component search record.

   This enables you to validate user entry in the dialog box. If an error is executed in SearchSave, the user is sent back to the Add or Data Entry dialog box. If a warning is executed, the user can click OK to continue or click Cancel to return to the dialog box and enter new values.
8. The Component Processor buffers the search key values and continues processing.

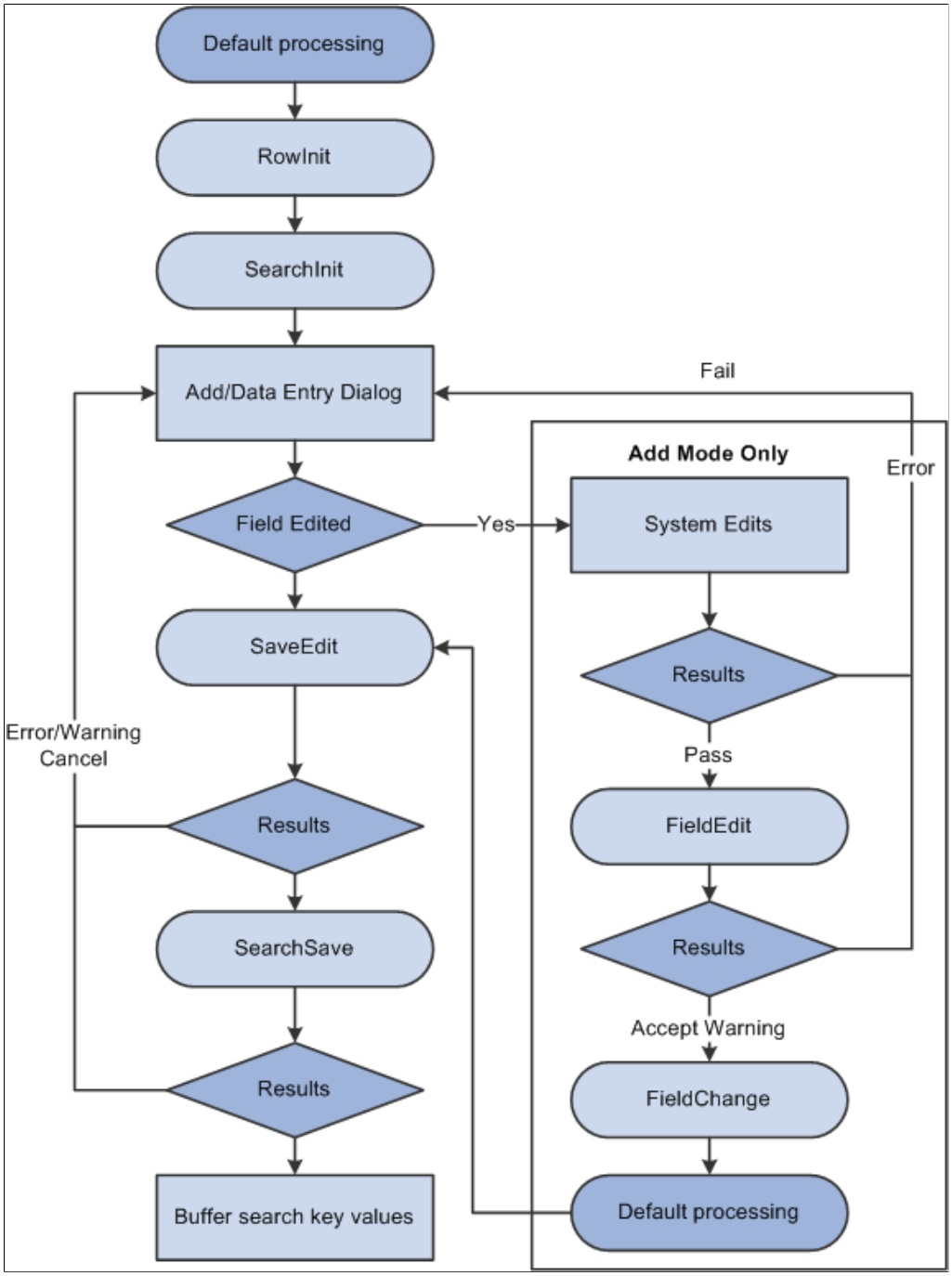
Note: If you compare the following diagram with search processing in update modes, notice that the add modes are considerably more complex and involve more PeopleCode events. However, in practice, PeopleCode development is similar in both cases. PeopleCode that runs before the dialog box appears (for example, to control dialog box appearance or set values in the dialog box fields) generally is placed in the SearchInit event; PeopleCode that validates user entry in the dialog box is placed in the SearchSave event.

See Search Processing in Update Modes.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Search processing logic in add and data-entry modes

The following flowchart diagram shows the logic of Search processing in add and data-entry modes.



Note: You can use the IsSearchDialog function to create PeopleCode that runs only during search processing. To create processes that run only in a specific action mode, use the %Mode system variable. This could be useful in code that is part of a library function and that is invoked in places other than from the search page. It could also be used in PeopleCode associated with a record field that appears in pages and in the search page.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"IsSearchDialog" (PeopleTools 8.59: PeopleCode Language Reference)
"%Mode" (PeopleTools 8.59: PeopleCode Language Reference)

Component Build Processing in Update Modes

After the Component Processor has saved the search keys values for the component, it uses the search key values to select rows of data from the database server using a SQL Select statement. After the rows are retrieved, the Component Processor performs these actions:

1. Performs row select processing, in which rows of data that have already been selected from the database server can be filtered before they are added to the component buffer.

   See Row Select Processing.

2. Initiates the PreBuild event, which triggers any PreBuild PeopleCode associated with the component record, enabling you to set global or component scope variables that can be used later by PeopleCode located in other events.

   The PreBuild event is also used to validate data entered in the search page, after a prompt list is displayed.

   Note: If a PreBuild PeopleCode program issues an error or warning, the user is returned to the search page. If there is no search page, that is, the search record has no keys, a blank component page appears.

3. Performs default processing on all the rows and fields in the component.

   See Row Select Processing.

4. Initiates the RowInit event, which triggers any RowInit PeopleCode associated with the record field or the component record.

   The RowInit event enables you to programmatically initialize the values of non-blank fields in the component.

5. Initiates the PostBuild event, which triggers any PostBuild PeopleCode associated with the component record, enabling you to set global or component scope variables that can be used later by PeopleCode located in other events.

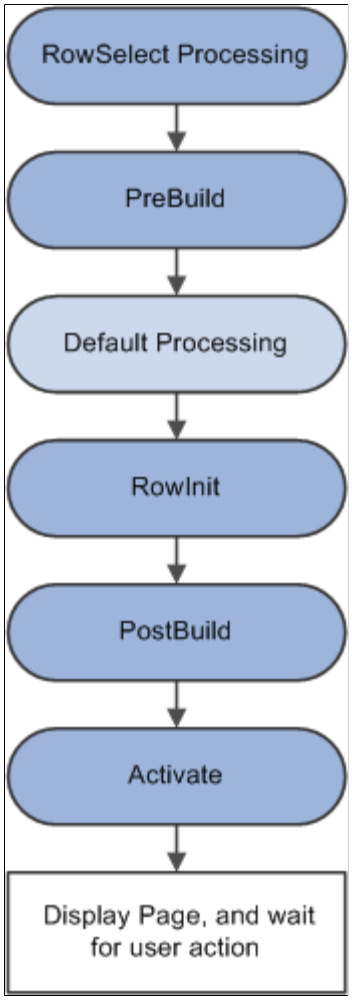
6. Initiates the Activate event, which triggers any Activate PeopleCode associated with the page about to be displayed, enabling you to programmatically control the display of that page.

7. Displays the component and waits for end-user action.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Component build processing in update modes

The following flowchart shows the logic of Component build processing in update modes.



Row Select Processing

Row select processing enables PeopleCode to filter out rows of data after they have been retrieved from the database server and before they are copied to the component buffers. Row select processing uses a SQL Select statement .

Row select processing is a subprocess of component build processing in add modes. It also occurs after a ScrollSelect or related function is executed.

See Component Build Processing in Add Modes.

Note: Instead of using row select processing, it is more efficient to filter out the rows using a search view, an effective-dated record, the Select method, or ScrollSelect or a related function, before the rows are sent to the browser.

In row select processing, the following actions occur:

1. The Component Processor checks for more rows to add to the component.

Copyright © 1988, 2021, Oracle and/or its affiliates.

2. The Component Processor initiates the RowSelect event, which triggers any RowSelect PeopleCode associated with the record field or component record.

   This enables PeopleCode to filter rows using the StopFetching and DiscardRow functions. StopFetching causes the system to add the current row to the component, and then to stop adding rows to the component. DiscardRow filters out a current row, and then continues the row select process.

3. If neither the StopFetching nor DiscardRow function is called, the Component Processor adds the rows to the page and checks for the next row.

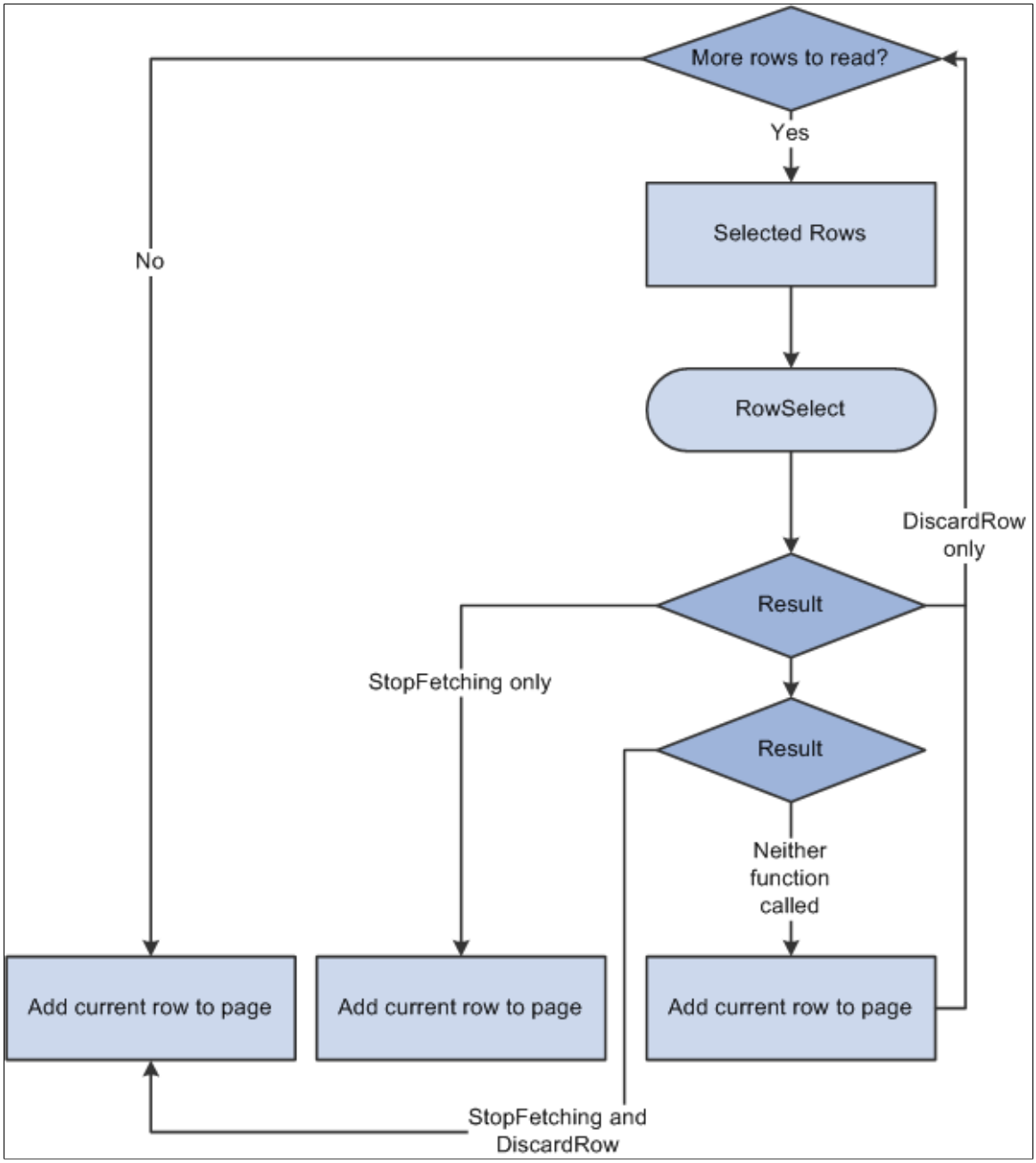
   The process continues until there are no more rows to add to the component buffers. If both StopFetching and DiscardRow are called, the current row is not added to the page, and no more rows are added to the page.

Note: In RowSelect PeopleCode, you can refer only to record fields on the record that is currently being processed, because the buffers are in the process of being populated. This means that the data might not be present.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: RowSelect processing logic

The following flowchart shows the logic of RowSelect processing in a Page.



Related Links

"StopFetching" (PeopleTools 8.59: PeopleCode Language Reference)

Component Build Processing in Add Modes

After search processing in add or data-entry modes, the Component Processor:

1. Initiates the PreBuild event.

Copyright © 1988, 2021, Oracle and/or its affiliates.

2. Runs default processing on all page fields.

   This enables you to set default fields programmatically using FieldDefault PeopleCode.

3. Initiates the RowInit event on all fields in the component, which triggers any RowInit PeopleCode associated with the record field or component record.

   This enables you to initialize the state of page controls, using RowInit PeopleCode, before the controls are displayed. (RowInit enables you to set the values of non-blank fields programmatically, whereas default processing is used to set blank fields to their default values.)

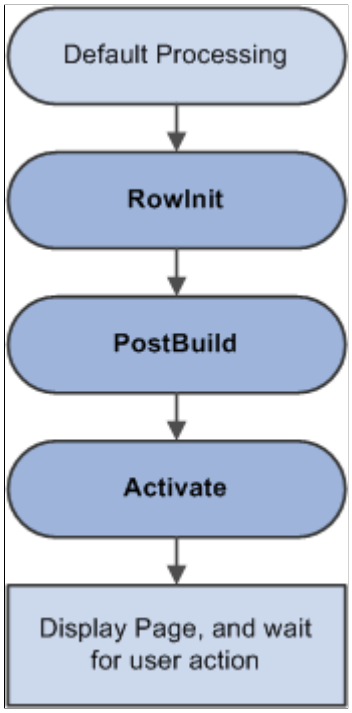
4. Initiates the PostBuild event, which triggers any PostBuild PeopleCode associated with the component record, enabling you to set global or component scope variables that can be used later by PeopleCode located in other events.

5. Initiates the Activate event, which triggers any Activate PeopleCode associated with the page about to be displayed, enabling you to programmatically control the display of that page.

6. Displays a new component, using the search keys obtained from the Add or Data Entry dialog box, with other fields set to their default values.

Image: Logic of component build processing in add modes

The following flowchart shows the logic of component build processing in add modes.



Field Change Processing

The field change processing sequence occurs after a user does any of the following:

- Changes the contents of a field, and then leaves the field (changes focus, for example, by tabbing out of the field).

Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: In certain, limited circumstances, tabbing out of a field is not recognized as a change of focus. This occurs when the next item in the tab order does not have an "on-focus handler." For example, the embedded help icon does not have an on-focus handler. In these limited circumstances, field change processing will not occur.

- Changes the state of a radio button or check box.
- Clicks a link or a button.

In this sequence, the following actions occur:

1. The Component Processor performs standard system edits.

   To reduce trips to the server, some processing must be done locally on the machine where the browser is located, while some is performed on the server.

   Standard system edits can be done either in the browser, utilizing local JavaScript code, or on the application server. The following table outlines where these system edits are done.

System Edits	***Location of Execution***
Checking data type	Browser
Formatting	Application server or browser
Updating current or history record	Application server
Effective date	Application server
Effective date or sequence	Application server
New effective date in range	Application server
Duplicate key	Application server
Current level is not effective-dated but one of its child scroll areas is	Application server
Required field	Browser
Date range	Browser
Prompt table	Application server
Translate table	Browser
Yes/no table	Depends on the field type. Browser if the field is a check box. Application server if the field is an edit box and the values are *Y* or *N*.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: Default processing for the field can be done in the browser only if the default value is specified as a constant in the record field properties. If the field contains a default, these defaults occur only upon component initialization. Then, if a user replaces a default value with a blank, the field is not initialized again. The required fields check is not performed on derived work fields when you press Tab to move out of a field.

If the data fails the system edits, the Component Processor displays an error message and highlights the field in the system color for errors (usually red).

2. If the field passes the system edits, Component Processor initiates the FieldEdit PeopleCode event, which triggers any FieldEdit PeopleCode associated with the record field or the component record field.

   This enables you to perform additional data validation in PeopleCode. If an Error statement is called in any FieldEdit PeopleCode, the Component Processor treats the error as it does a system edit failure; a message is displayed, and the field is highlighted. If a Warning statement is executed in any FieldEdit PeopleCode, a warning message appears, alerting the user to a possible problem, but the system accepts the change to the field.

3. If the field change is accepted, the Component Processor writes the change to the component buffer, then initiates the FieldChange event, which triggers any FieldChange PeopleCode associated with the record field or the component record field.

   This event enables you to add processes other than validation initiated by the changed field value, such as changes to page appearance or recalculation of values in other page fields. An Error or Warning statement in any FieldChange PeopleCode causes a runtime error.

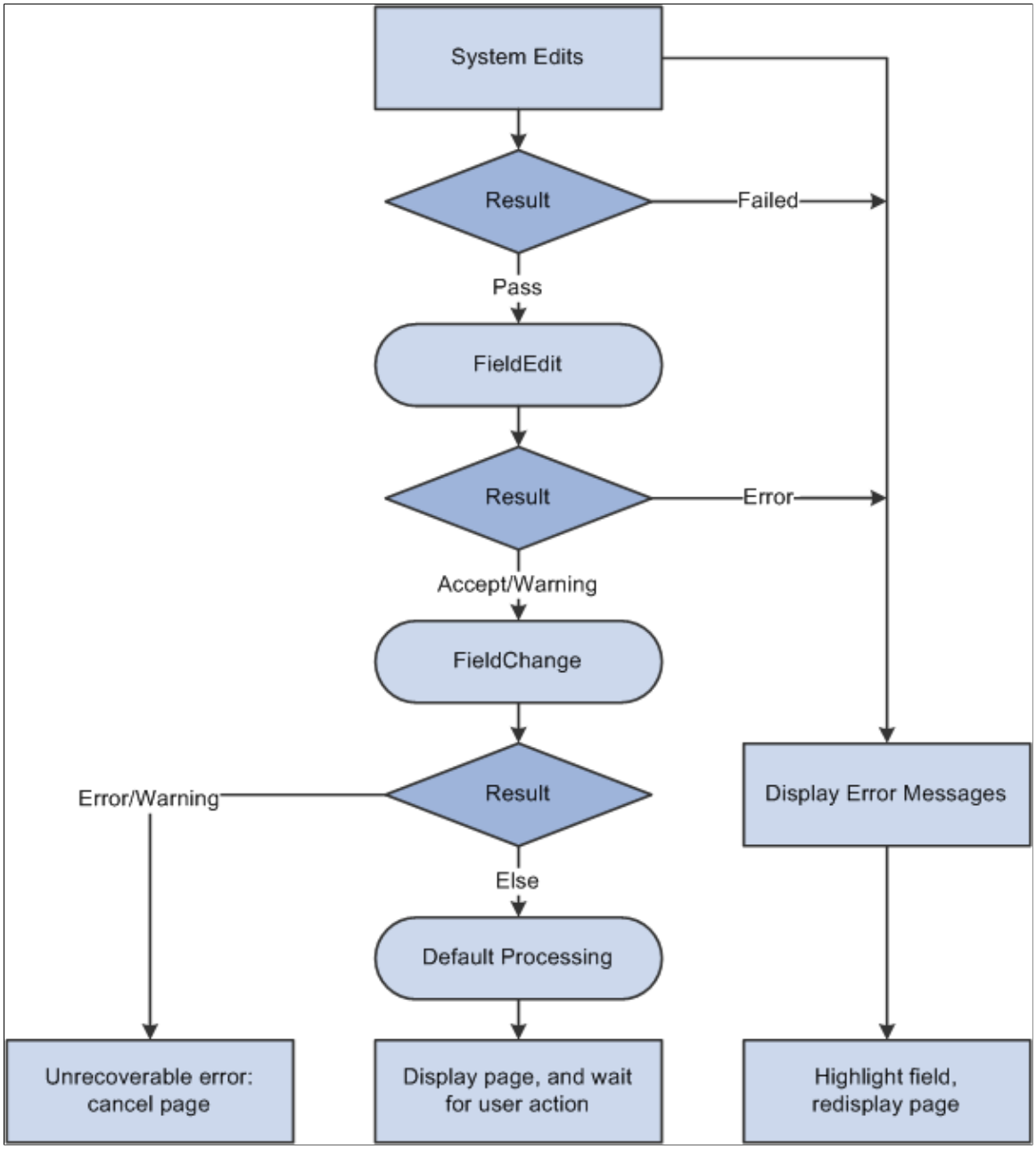
   Important! Do not use Error or Warning statements in FieldChange PeopleCode. All data validation should be performed in FieldEdit PeopleCode.

   After FieldChange processing, Component Processor runs default processing on all page fields, then redisplays the page. If the user has changed the field value to a blank, or if SetDefault or a related function is executed, and the changed field has a default value specified in the record field definition or any FieldDefault PeopleCode, the field is initialized again to the default value.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Logic of field modification processing

The following flowchart shows the logic of field modification processing in a Page.



Row Insert Processing

Row insert processing occurs when:

- A user requests a row insert in a scroll area by pressing Alt+7, by clicking the Insert Row button, or by clicking the New button.
- A PeopleCode RowInsert function or a InsertRow method requests a row insert.

In either case, the Component Processor performs these actions:

Copyright © 1988, 2021, Oracle and/or its affiliates.

1. Inserts a new row of data into the active scroll area.

   If the scroll area has a dependent scroll area, the system inserts a single new row into the blank scroll area, and the system continues until it reaches the lowest-level scroll area.

2. Initiates the RowInsert PeopleCode event, which triggers any RowInsert PeopleCode associated with the record field or the component record.

   This event processes fields only on the inserted row and any dependent rows that were inserted on lower-level scroll areas.

3. Runs default processing on all component fields.

   Normally this affects only the inserted row fields and fields on dependent rows, because other rows already have undergone default processing.

4. Initiates the RowInit PeopleCode event, which triggers any RowInit PeopleCode associated with the record field or the component record.

   This event affects fields only on the inserted row and any dependent rows that were inserted.

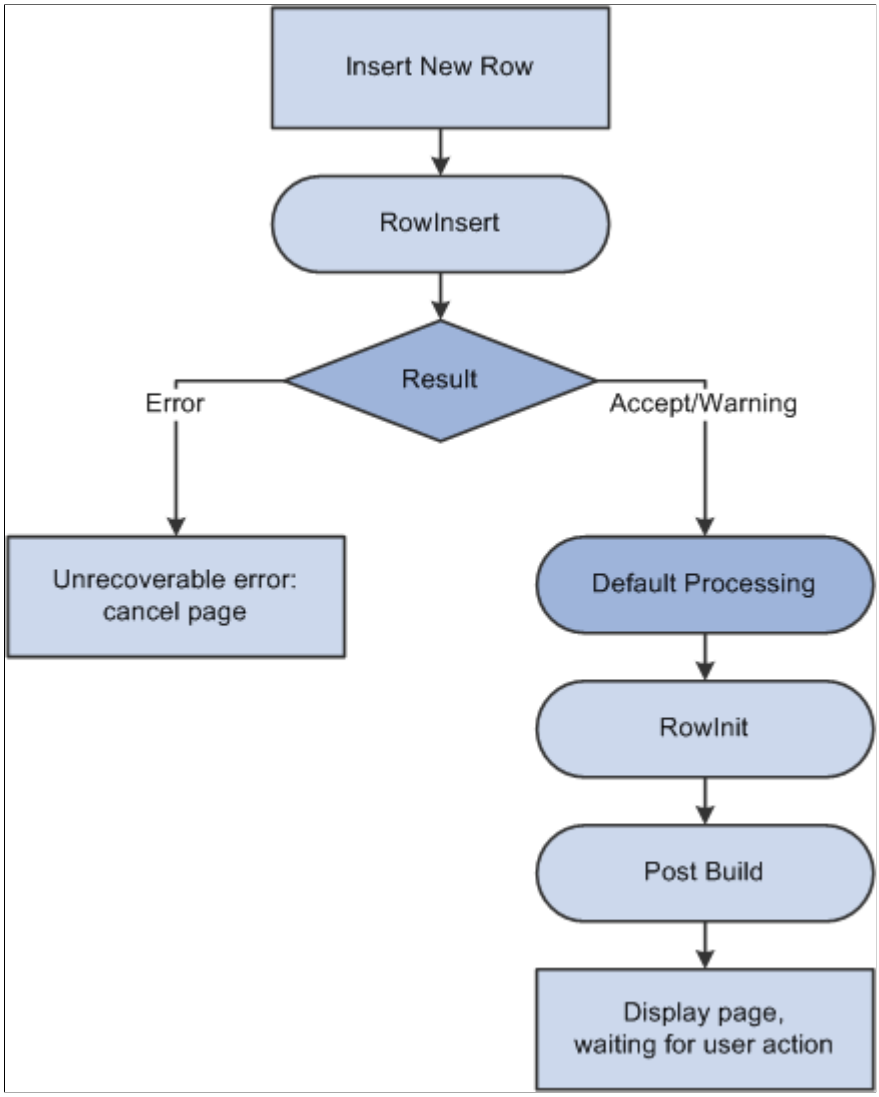
5. Redisplays the page and waits for user action.

Important! Do not use Error or Warning statements in RowInsert PeopleCode. All data validation should be performed in FieldEdit or SaveEdit PeopleCode.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Logic of row insert processing

The following flowchart shows the logic of row insert processing where in a PeopleCode RowInsert function or a InsertRow method requests a row insert.



Note: If none of the data fields in the new row are changed after the row has been inserted (either programmatically or by the user), the new row is not inserted into the database when the page is saved.

Row Delete Processing

Row delete processing occurs when:

- A user requests a row delete in a scroll area by pressing Alt+8, by clicking the Delete Row button, or by clicking the Delete button.
- A PeopleCode RowDelete function or a DeleteRow method requests a row delete.

In either case, these actions occur:

Copyright © 1988, 2021, Oracle and/or its affiliates.

1. The Component Processor initiates the RowDelete PeopleCode event, which triggers RowDelete PeopleCode associated with the record field or the component record.

   This event processes fields on the deleted row and any dependent child scroll areas. RowDelete PeopleCode enables you to check for conditions and control whether a user can delete the row. An Error statement displays a message and prevents the user from deleting the row. A Warning statement displays a message alerting the user about possible consequences of the deletion, but permits deletion of the row.

2. If the deletion is rejected, the page is redisplayed after the error message.

3. If the deletion is accepted, the row, and any child scroll areas dependent on the row, are flagged as deleted.

   The row no longer appears in the page, but it is not physically deleted from the buffer and can be accessed by PeopleCode all the way through the SavePostChange event (note, however, that SaveEdit PeopleCode is not run on deleted rows).

4. The Component Processor runs default processing on all component fields.

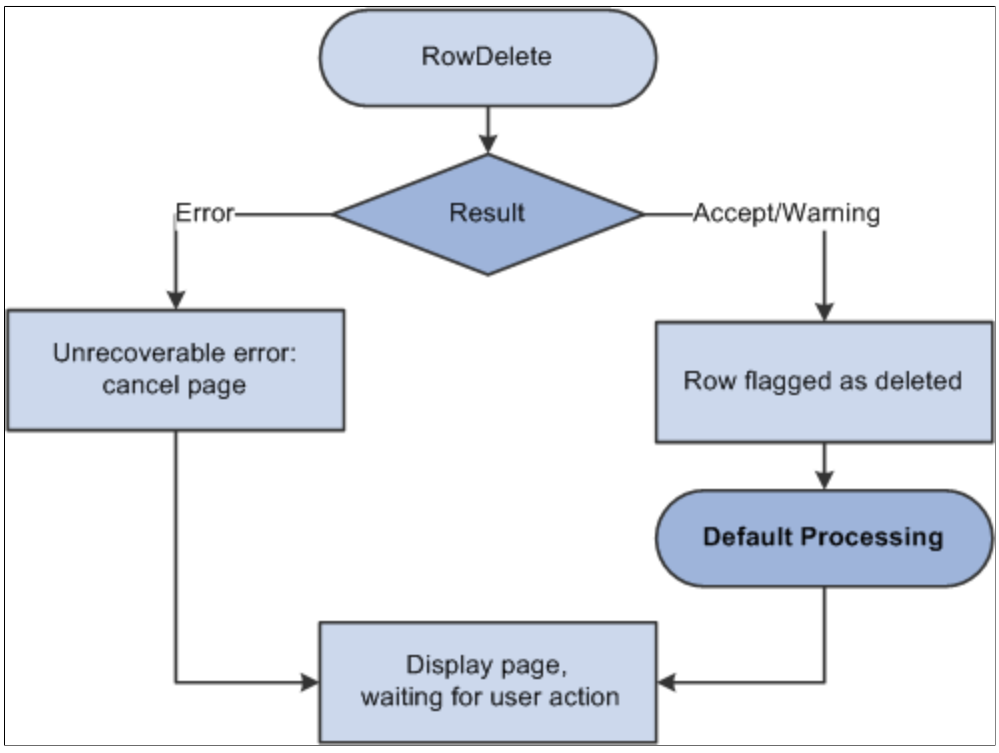
5. The Component Processor redisplays the page and waits for a user action

---

Note: PeopleCode programs are triggered on rows flagged as deleted in SavePreChange and SavePostChange PeopleCode. Use the IsDeleted row class property to test whether a row has been flagged as deleted. You can also access rows flagged as deleted by looping through the rows of a scroll area using a For loop delimited by the value returned by the RowCount rowset property.

Image: Logic of row delete processing

The following flowchart shows the logic of row delete processing.



Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"IsDeleted" (PeopleTools 8.59: PeopleCode API Reference)
"RowCount" (PeopleTools 8.59: PeopleCode API Reference)
"For" (PeopleTools 8.59: PeopleCode Language Reference)

Buttons

When a user presses a button, this initiates the same processing as changing a field. Typically, PeopleCode programs started by button are placed in the FieldChange event.

Related Links

Field Change Processing

Prompts

If the Allow Search Events for Prompt Dialogs check box is selected for the Record Field properties for a search key on a prompt table record, the search processing events are enabled for that field. When the user selects the prompt icon, the SearchInit event for that field executes before the search dialog displays. When the user selects the Look Up button on a prompt dialog the SearchSave event for the field executes.

Search event processing on prompt dialogs can affect performance. Oracle recommends that you limit the use of search events in prompt dialogs to simple tasks such as showing and hiding fields or character manipulation. Do not use the search events on prompt dialogs for complex functions such as AddKeyListItem, ClearKeyList, ClearSearchDefault, ClearSearchEdit, IsSearchDialog, SetSearchDefault, SetSearchDialogBehavior, or SetSearchEdit, and so on.

By default, Allow Search Events for Prompt Dialogs is off, in which case no PeopleCode event is initiated as a result of prompts.

No PeopleCode events are initiated as a result of the user returning to the search page or displaying a calendar. This process is controlled automatically by the system.

Note: When the value of a field is changed using a prompt, the standard field modification processing occurs.

Related Links

"Setting Record Field Use Properties" (PeopleTools 8.59: Application Designer Developer's Guide)
Field Change Processing
Search Processing in Update Modes

Pop-Up Menu Display

To display a pop-up menu, a user can click the pop-up button, either next to a field or at the bottom of a page (if the page has a pop-up menu associated with it.) The user can open a standard pop-up menu on a page field if no pop-up menu has been defined by an application developer for that page field.

The PrePopup PeopleCode event initiates only if the user opens a pop-up menu defined by an application developer on a page field. It does not initiate for a pop-up menu attached to the page background.

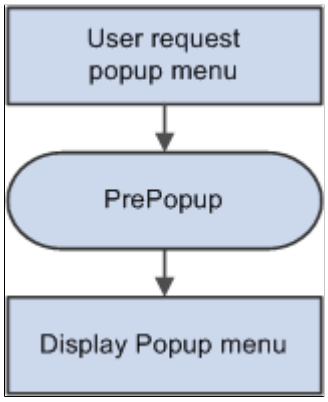
Copyright © 1988, 2021, Oracle and/or its affiliates.

The PrePopup PeopleCode event enables you to disable, check, or hide menu items in the pop-up menu.

PrePopup PeopleCode menu item operations (such as HideMenuItem, EnableMenuItem, and so on) work with pop-up menus attached to a grid, not a field in a grid, only if the PrePopup PeopleCode meant to operate on that pop-up menu resides in the record field that is attached to the first column in the grid. It does not matter if the first field is visible or hidden.

Image: Logic of PrePopup even processing

The following flowchart shows the logic of PrePopup event processing, which enables the user to disable, check, or hide menu items in the pop-up menu.

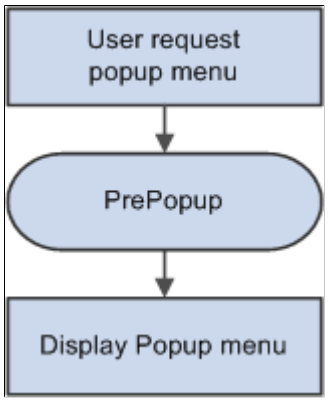


Selected Item Processing

Selected item processing occurs when a user selects a menu item from a pop-up menu. This initiates the ItemSelected PeopleCode event, which is a menu PeopleCode event.

Image: Logic of selected item processing

The following flowchart shows the logic of selected item processing, this occurs when a user selects a menu item from a pop-up menu.



Save Processing

A user can direct the system to save a component by clicking Save or by pressing Alt+1.

Copyright © 1988, 2021, Oracle and/or its affiliates.

An application can prompt the user to save a component when the Next or List button is clicked, or when a new action or component is selected. If the user clicks Save after being prompted, save processing begins.

The following actions occur in save processing:

1. The Component Processor initiates the SaveEdit PeopleCode event, which triggers any SaveEdit PeopleCode associated with a record field or a component record.

   This enables you to cross-validate page fields before saving, checking consistency among the page field values. An Error statement in SaveEdit PeopleCode displays a message and then redisplays the page, stopping the save. A Warning statement enables the user to cancel save processing by clicking Cancel, or to continue with save processing by clicking OK.

2. The Component Processor initiates the SavePreChange event, which triggers any SavePreChange PeopleCode associated with a record field, a component record, or a component.

   SavePreChange PeopleCode enables you to process data after validation and before the database is updated.

3. The Component Processor initiates the Workflow event, which triggers any Workflow PeopleCode associated with a record field or a component.

   Workflow PeopleCode should be used only for workflow-related processing (TriggerBusinessEvent and related functions).

4. The Component Processor updates the database with the changed component data, performing any necessary SQL Insert, Update, and Delete statements.

5. The Component Processor initiates the SavePostChange PeopleCode event, which triggers any SavePostChange PeopleCode associated with a record field, a component record, or a component.

   You can use SavePostChange PeopleCode for processing that must occur after the database update, such as updates to other database tables not in the component buffer.

6. The Component Processor issues a SQL Commit statement to the database server.

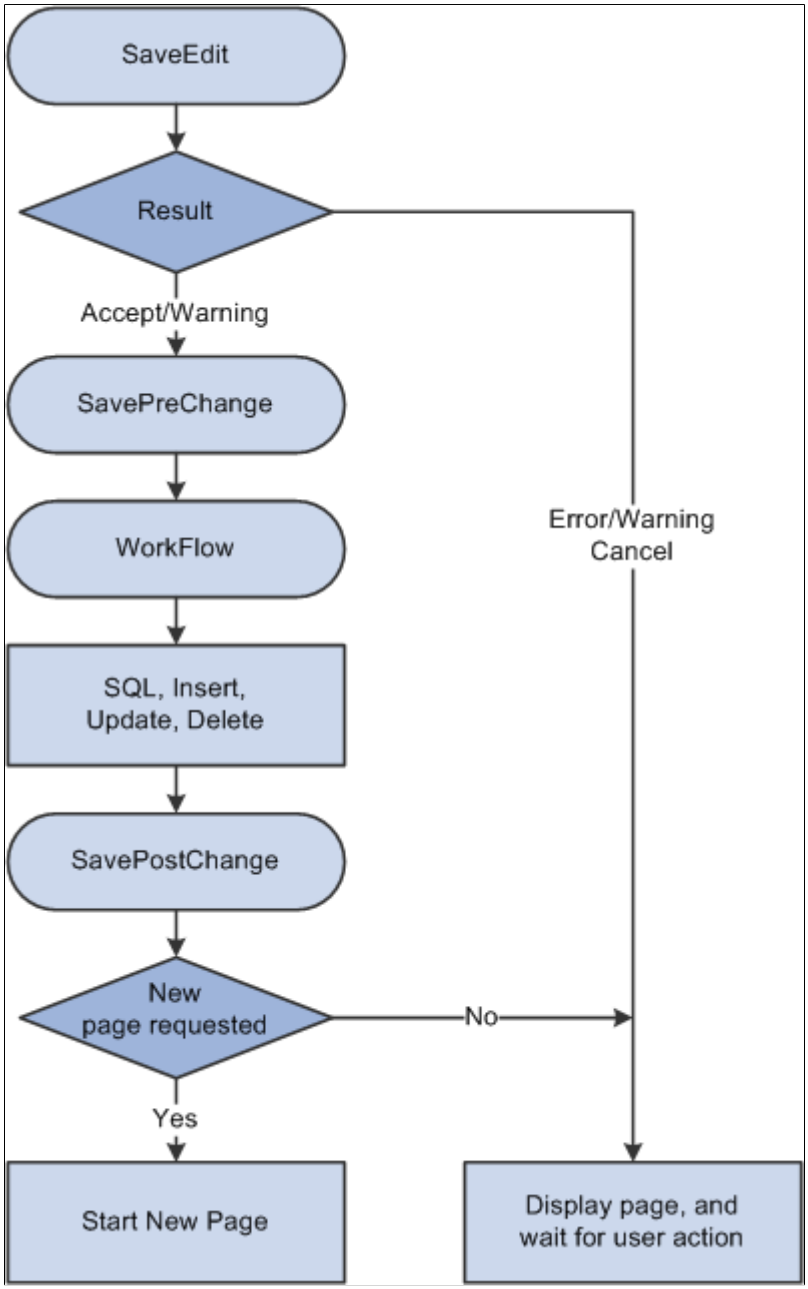
7. The Component Processor redisplays the component.

Important! Never use an Error or Warning statement in any save processing event other than SaveEdit. Perform all component data validation in SaveEdit.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Logic of save processing

The following flow chart shows the logic of save processing sequence.



PeopleSoft Pure Internet Architecture Processing Considerations

Keep the following points in mind concerning the PeopleSoft Pure Internet Architecture:

- If a user changes a field that field has nothing to cause a trip to the server, then default processing and FieldFormula PeopleCode do not run.

  These processes only run when another event causes a trip to the server.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Other fields that depend on the first field using FieldFormula or default PeopleCode are not updated until the next time a server trip occurs.

- In applications that run on the PeopleSoft portal, external, dynamic link information must be placed in RowInit PeopleCode.

  If it is placed in FieldChange PeopleCode, it will not work.

Deferred Processing Mode

When a component runs in deferred processing mode, trips to the server are reduced. When deploying some pages in the browser, you may want the user to be able to input data with minimal interruption or trips to the server. Each trip to the server can slow down your application. By specifying a component as deferred processing mode, you can achieve better performance.

PeopleSoft applications use AJAX (Asynchronous JavaScript and XML) technology to limit server trips and perform partial page refreshes. AJAX uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information, rather than whole pages, from the server. AJAX technology makes Internet applications smaller, faster, and more user-friendly.

With a partial page refresh, the browser refreshes the entire page only when the user navigates to a new page. Any server trips triggered by PeopleCode functions, such as FieldChange and FieldEdit, or for related display fields do not redraw the entire page; the refresh updates only the changed fields. Because of AJAX technology, much of the communication with the server happens in the background. Users continue to work uninterrupted during this background processing.

AJAX and partial page refreshes allow you to more freely design interactive applications and deploy interactive fields. Nevertheless, the default configuration for components, pages, and fields remains deferred mode. Oracle still recommends that you leverage deferred processing mode to limit network traffic. Note that:

- Deferred processing set at the component level can be overridden at either the page field or page level.
- Conversely, interactive processing set at the component level cannot be overridden at either the page field or page level.
- Similarly, interactive processing set at the page level cannot be overridden at the page field level.

The characteristics of deferred processing mode are:

1. Field modification processing is deferred.

   No field modification processing is done in the browser. FieldEdit and FieldChange PeopleCode, as well as other edits, such as required field checks, formats, and so on, do not run until a specific user action occurs. Several actions cause field modification processing to execute, for example, clicking a button or link, navigating to another page in the component, and saving the page. The following actions do not cause field processing:

   - Clicking an external link.
   - Clicking a list (performing a search).

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Clicking a process button.

  Deferred processing mode affects the appearance of pages in significant ways. For example, related processing is not done when the user presses Tab to move out of a field. Avoid related fields for components that use this mode.

2. Drop-down list box values are static while the page appears in the browser.

   Drop-down list box values are generated on the application server when generating the HTML for the page.

   If translate values are used to populate the drop-down list box, and the current record contains an effective date, that date is static while the page is displayed. This means the drop-down list box values may become out of date.

   If prompt table values are used to populate the drop-down list box, the high-order key field values for the prompt table are static while the page is displayed. This means the drop-down list box values may become out of date.

   Avoid interdependencies in drop-down lists used on pages executed in deferred mode, because the lists may quickly become out of date.

3. No field modification processing is done during prompt button processing.

   When the user clicks a prompt button, a trip is made to the application server (if values were not already downloaded) to select the search results from the database and to generate the HTML for the prompt dialog box. During this trip to the application server, field modification processing for the field being prompted is not performed, because this may cause an error message for another field on the page, and this error may confuse the user. When deferred changes are made to other fields, field modification processing for these fields is done before prompting. The field modification for the prompted field is done after returning from the prompt page. While the system displays the page, the high-order key field values for the prompt table should be static or not require field modification processing. Display-only drop-down list box, radio button, and check box fields do not require field modification processing. Field values that do not require field modification processing are temporarily written to the component buffer, without any field modification processing being performed on them, including FieldEdit and FieldChange PeopleCode. The system restores the original state of the page processor before returning to the browser.

4. Field modification processing executes in field layout order.

   The entire field modification processing sequence executes in field layout order for each field. If a field passes the system edits and FieldEdit PeopleCode, the field value is written to the component buffer. If an error occurs, field modification processing stops, and the system generates new HTML for the page, with the field in error highlighted and sent to the browser.

5. PeopleCode dependencies between fields on the page do not work as expected.

   Avoid PeopleCode dependencies between fields on pages displayed in deferred processing mode. Also, avoid FieldChange PeopleCode that changes the display.

   The following are examples of PeopleCode dependencies between fields on the page and the application server's action. In the following examples, field A comes before field B, which comes before field C.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Field A has FieldChange PeopleCode that hides field B or it makes unavailable for entry.

  The value in field B of the page that was submitted from the browser is discarded.

- Field B has FieldChange PeopleCode that hides field A or makes it unavailable for entry.

  The change made by the user for field A, if any, remains in the component buffer.

- Field A has FieldChange PeopleCode that changes the value in the component buffer for field B.

  If the value in field B of the page that was submitted from the browser passes the system edits and FieldEdit PeopleCode, it is written to the component buffer, overriding the change made by field A's FieldChange PeopleCode.

- Field B has FieldChange PeopleCode that changes the value in the component buffer for field A.

  The change made by field B's FieldChange PeopleCode overrides the change made by the user to field A, if any.

- Field A has FieldChange PeopleCode that unhides field B or makes it available for entry.

  Field B has the value that was already in the component buffer. If the user requests a different page or finishes, the user may not have the opportunity to enter a value into field B, and therefore the value may not be correct.

- Field B has FieldChange PeopleCode that changes the value in the component buffer for field A, but field C has FieldChange PeopleCode that hides field B or makes it unavailable for entry.

  The change made by field B's FieldChange PeopleCode, a field that is now hidden or unavailable for entry, overrides the change made by the user to field A, if any.

  Avoid such dependencies by moving FieldChange PeopleCode logic from individual fields to save processing for the component or FieldChange PeopleCode on a PeopleCode command button.

6. Not all buttons cause field modification processing to execute.

   External links, list (search), and process buttons do not cause field modification processing to execute.

7. You can use a PeopleCode command button to cause field modification processing to execute.

   An application can include a button for the sole purpose of causing field modification processing to execute. The result is a new page showing any display changes that resulted from field modification processing.

   In addition, if the user clicks the Refresh button, or presses Alt + 0, deferred processing is executed.

   Note: The Refresh button does not refresh the page from the database. It simply causes a server trip so any deferred PeopleCode changes get processed. If the page has no deferred changes or the deferred changes do not cause any errors or other changes on the page, it may appear to the user as if nothing happened.

   ---

8. A scroll button (link) causes field modification processing to execute.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: Even for a field in interactive mode, a field change does not automatically result in a trip back through the component build stream. In PeopleSoft systems, whether in interactive or deferred mode, if a user changes a field but there is nothing to cause a trip to the server on that field, default processing and FieldFormula PeopleCode do not run. They run only when another action or event causes a trip to the server. In other words, if a field is changed but there is no FieldEdit or FieldChange PeopleCode on the field, then record defaults, FieldDefault PeopleCode, and FieldFormula PeopleCode do not run until another action causes a trip to the server.

Related Links

"Understanding Pages" (PeopleTools 8.59: Applications User's Guide)

"Improving Online Performance" (PeopleTools 8.59: Application Designer Developer's Guide)

PeopleCode Events

This section discusses:

- Activate event.
- FieldChange event.
- FieldDefault event.
- FieldEdit event.
- FieldFormula event.
- ItemSelected event.
- PostBuild event.
- PreBuild event.
- PrePopup event.
- RowDelete event.
- RowInit event.
- RowInsert event.
- RowSelect event.
- SaveEdit event.
- SavePostChange event.
- SavePreChange event.
- SearchInit event.
- SearchSave event.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Workflow event.

Note: The term *PeopleCode type* is still frequently used, but it does not fit into the PeopleTools object-based, event-driven metaphor. The term *PeopleCode event* should now be used instead. However, it's often convenient to qualify a class of PeopleCode programs triggered by a specific event with the event name; for example, PeopleCode programs associated with the RowInit events are collectively referred to as *RowInit PeopleCode.*

Activate Event

The Activate event is initiated each time that a page is activated, including when a page is first displayed by a user, or if a user presses Tab between different pages in a component. Each page has its own Activate event.

Activate PeopleCode associated with a popup page execute after the page activate event for the main page. When fields on the main page change and trigger updates on the popup page the page activate event for the popup page is executed.

The Activate event segregates PeopleCode that is related to a specific page from the rest of the application's PeopleCode. Place PeopleCode related to page display or page processing, such as enabling a field or hiding a scroll area, in this event. Also, you can use this event for security validation: if an user does not have clearance to view a page in a component, you would put the code for hiding the page in this event.

Note: PeopleSoft builds a page grid one row at a time. Because the Grid class applies to a complete grid, you cannot attach PeopleCode that uses the Grid class to events that occur before the grid is built; the earliest event you can use is the Activate event. The Activate event is not associated with a specific row and record at the point of execution. This means you cannot use functions such as GetRecord, GetRow, and so on, which rely on context, without specifying more context.

Activate PeopleCode can only be associated with pages.

This event is valid only for pages that are defined as standard or secondary. This event is not supported for subpages.

Note: If your application uses the MessageBox built-in function in the Activate event with a message from the message catalog that's defined as type Error, Warning or Cancel, all component processing stops with an error message to that effect. If the message has a type of Message, processing does not stop.

Related Links

Component Build Processing in Update Modes
Component Build Processing in Add Modes

FieldChange Event

Use FieldChange PeopleCode to recalculate page field values, change the appearance of page controls, or perform other processing that results from a field change other than data validation. To validate the contents of the field, use the FieldEdit event. See FieldEdit Event.

Copyright © 1988, 2021, Oracle and/or its affiliates.

FieldChange PeopleCode can be associated with record fields and component record fields. The FieldChange event applies to the field and row that just changed. For a definition of what constitutes a *field change,* see Field Change Processing.

FieldChange PeopleCode is often paired with RowInit PeopleCode. In these RowInit/FieldChange pairs, the RowInit PeopleCode checks values in the component and initializes the state or value of page controls accordingly. FieldChange PeopleCode then rechecks the values in the component during page execution and resets the state or value of page controls.

To take a simple example, suppose you have a derived/work field called PRODUCT, the value of which is always the product of page field A and page field B. When the component is initialized, you would use RowInit PeopleCode to initialize PRODUCT equal to A × B when the component starts up or when a new row is inserted. You could then attach FieldChange PeopleCode programs to both A and B which also set PRODUCT equal to A × B. Whenever a user changes the value of either A or B, PRODUCT is recalculated.

FieldDefault Event

The FieldDefault PeopleCode event enables you to programmatically set fields to default values when they are initially displayed. This event is initiated on all page fields as part of many different processes; however, it triggers PeopleCode programs only when the following conditions are all True:

- The page field is still blank after applying any default value specified in the record field properties.

  This is True if there is no default specified, if a null value is specified, or if a 0 is specified for a numeric field.

- The field has a FieldDefault PeopleCode program.

In practice, FieldDefault PeopleCode normally sets fields by default when new data is being added to the component; that is, in Add mode and when a new row is inserted into a scroll area.

If a field value is changed, whether through PeopleCode or by a 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 events. If a value is set in FieldDefault or FieldFormula, 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.

You must attach FieldDefault PeopleCode to the field where the default value is being populated.

Note: An error or warning issued from FieldDefault PeopleCode causes a runtime error.

FieldDefault PeopleCode can be associated with record fields and component record fields.

Related Links

Default Processing

 Copyright © 1988, 2021, Oracle and/or its affiliates.

FieldEdit Event

Use FieldEdit PeopleCode to validate the contents of a field, supplementing standard system edits. If the data does not pass the validation, the PeopleCode program should display a message using the Error statement, which redisplays the page, displaying an error message and turning the field red.

To permit the field edit but alert the user to a possible problem, use a Warning statement instead of an Error statement. A Warning statement displays a warning dialog box with OK and Explain buttons. It permits field contents to be changed and continues processing as usual after the user clicks OK.

If the validation must check for consistency across page fields, then use SaveEdit PeopleCode instead of FieldEdit.

FieldEdit PeopleCode can be associated with record fields and component record fields. The FieldEdit event applies to the field and row that just changed. For a definition of what constitutes a *field change,* see Field Change Processing.

FieldFormula Event

The FieldFormula event is not currently used. Because FieldFormula PeopleCode initiates in many different contexts and triggers PeopleCode on every field on every row in the component buffer, it can seriously degrade application performance. Use RowInit and FieldChange events rather than FieldFormula.

If a field value is changed, whether through PeopleCode or by a user, the IsChanged property for the row is usually set to True. However, if a value is set in FieldDefault or FieldFormula, 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.

Note: In PeopleSoft Pure Internet Architecture, if a user changes a field but that field has nothing to cause a trip to the server, then default processing and FieldFormula PeopleCode do not run. They only run when another event causes a trip to the server.

As a matter of convention, FieldFormula is now often used in FUNCLIB_ (function library) record definitions to store shared functions. However, you can store shared functions in any PeopleCode event.

FieldFormula PeopleCode is only associated with record fields.

ItemSelected Event

The ItemSelected event is initiated whenever a user selects a menu item from a pop-up menu. In pop-up menus, ItemSelected PeopleCode executes in the context of the page field from where the pop-up menu is attached, which means that you can freely reference and change page fields, just as you could from a button.

Note: This event, and all its associated PeopleCode, does not initiate if run from a component interface.

ItemSelected PeopleCode is only associated with pop-up menu items.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

Selected Item Processing

PostBuild Event

The PostBuild event is initiated after all the other component build events have been initiated. This event is often used to hide or unhide pages. It is also used to set component variables.

PostBuild PeopleCode is only associated with components.

PreBuild Event

The PreBuild event is initiated before the rest of the component build events. This event is often used to hide or unhide pages. It is also used to set component variables.

Note: If a PreBuild PeopleCode program issues an error or warning, the user is returned to the search page. If the search record has no keys, a blank component page appears.

Also use the PreBuild event to validate data entered in a search page after a prompt list is displayed. For example, after a user selects key values on a search, the PreBuild PeopleCode program runs, catches the error condition, and issues an error message. The user receives and acknowledges the error message. The component is canceled (because of the error), and the user is returned to the search page. PreBuild PeopleCode is only associated with components.

PrePopup Event

The PrePopup event is initiated just before the display of a pop-up menu.

You can use PrePopup PeopleCode to control the appearance of the pop-up menu.

Note: This event, and all its associated PeopleCode, does not initiate if run from a component interface.

PrePopup PeopleCode can be associated with record fields and component record fields.

Related Links

Pop-Up Menu Display

RowDelete Event

The RowDelete event is initiated whenever a user attempts to delete a row of data from a page scroll area. Use RowDelete PeopleCode to prevent the deletion of a row (using an Error or Warning statement) or to perform any other processing contingent on row deletion. For example, you could have a page field called Total on scroll area level zero whose value is the sum of all the Extension page fields on scroll area level one. If the user deleted a row on scroll area level one, you could use RowDelete PeopleCode to recalculate the value of the Total field.

The RowDelete event triggers PeopleCode on any field on the row of data that is being flagged as deleted.

Note: RowDelete does not trigger programs on derived/work records.

Copyright © 1988, 2021, Oracle and/or its affiliates.

RowDelete PeopleCode can be associated with record fields and component records.

Deleting All Rows from a Scroll Area

When the last row of a scroll area is deleted, a new, dummy row is automatically added. As part of the RowInsert event, RowInit PeopleCode is run on this dummy row. If a field is changed by RowInit (even if it's left blank), the row is no longer new, and therefore is not reused by any of the ScrollSelect functions or the Select method. In this case, you may want to move your initialization code from the RowInit event to FieldDefault.

Related Links

Row Delete Processing
Using Errors and Warnings in RowDelete Events

RowInit Event

The RowInit event is initiated the first time that the Component Processor encounters a row of data. Use it to set the initial state of component controls during component build processing and row insert processing. The RowInit event also occurs after a Select or SelectAll Rowset method, or a ScrollSelect or related function, is executed.

Note: Generally, if none of the fields in the new row are changed after the row is inserted (either by a user pressing Alt+7 or programmatically) when the page is saved, the new row is not inserted into the database. However, if the ChangeOnInit rowset class property is set to False, you can set values for fields a new row in RowInsert or RowInit PeopleCode, and the row will not be saved.

RowInit is not field-specific. It triggers PeopleCode on all fields and on all rows in the component buffer.

Do not use Error or Warning statements in RowInit PeopleCode. They cause a runtime error.

RowInit PeopleCode is often paired with FieldChange PeopleCode. In these RowInit/FieldChange pairs, the RowInit PeopleCode checks values in the component and initializes the state or value of page controls accordingly. FieldChange PeopleCode then rechecks the values in the component during page execution and resets the state or value of page controls.

For a simple example, suppose you have a derived/work field called PRODUCT, the value of which is always the product of page field A and page field B. When the component is initialized, use RowInit PeopleCode to initialize PRODUCT equal to A × B when the component starts up or when a new row is inserted. You could then attach FieldChange PeopleCode programs to both A and B, which also sets PRODUCT equal to A × B. Whenever a user changes the value of either A or B, PRODUCT is recalculated.

RowInit PeopleCode can be associated with record fields and component records.

RowInit Exceptions

In certain rare circumstances, the Component Processor does not run RowInit PeopleCode for some record fields. The Component Processor runs RowInit PeopleCode when it loads the record from the database. However, in some cases, the record can be initialized entirely from the keys for the component. When this happens, RowInit PeopleCode is not run.

For RowInit to not run, the following must all be True:

Copyright © 1988, 2021, Oracle and/or its affiliates.

- The record is at level zero.
- Every record field that is present in the data buffers is also present in the keys for the component.

  The Component Processor determines if the field is required by the component. In practice, this usually means that the field is associated with a page field, possibly hidden, for some page of the component. It could also mean that the field is referenced by some PeopleCode program that is attached to an event on some other field of the component.
- Every record field that is present in the data buffers is display-only.

RowInit not running is not considered to be an error. The purpose of RowInit PeopleCode is to complete initialization of data on the row after it has been read from the database. Because the data in this special circumstance is coming from the keylist, it was already initialized correctly by whatever processing produced the keylist. More general initialization of the component should be done in PostBuild PeopleCode, not RowInit.

Related Links

Component Build Processing in Add Modes

"ChangeOnInit" (PeopleTools 8.59: PeopleCode API Reference)

RowInsert Event

When a user adds a row of data, the Component Processor generates a RowInsert event. You should use RowInsert PeopleCode for processing specific to the insertion of new rows. Do not put PeopleCode in RowInsert that already exists in RowInit, because a RowInit event always initiates after the RowInsert event, which will cause your code to be run twice.

Note: Generally, if none of the fields in the new row are changed after the row has been inserted (either by a user pressing Alt+7 or programmatically), when the page is saved, the new row is not inserted into the database. However, if the ChangeOnInit rowset class property is set to False, you can set values for fields a new row in RowInsert or RowInit PeopleCode, and the row won't be saved.

The RowInsert event triggers PeopleCode on any field on the inserted row of data.

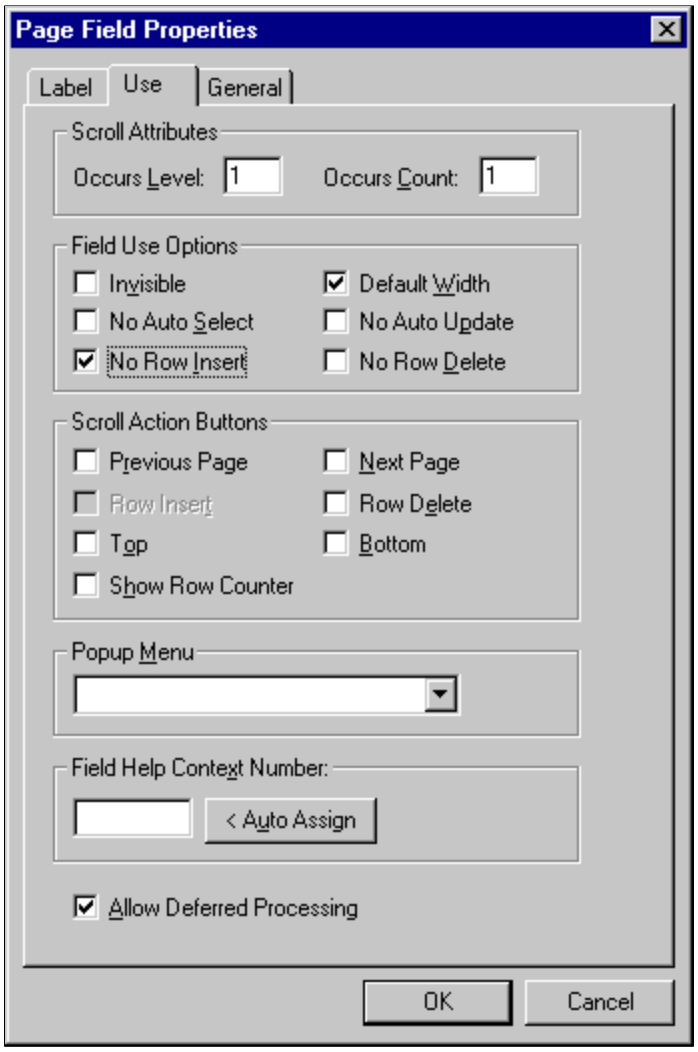
Do not use a warning or error in RowInsert.

You can prevent a user from inserting rows into a scroll area by selecting the No Row Insert check box in the scroll bar’s page field properties, as shown in the following illustration. However, you cannot prevent row insertion conditionally.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Setting row insert properties in page field properties for a scroll bar

This example illustrates the fields and controls on the Setting row insert properties in page field properties for a scroll bar. You can find definitions for the fields and controls later on this page.



Note: RowInsert does not trigger PeopleCode on derived/work fields.

RowInsert PeopleCode can be associated with record fields and component records.

Related Links

Row Insert Processing

"ChangeOnInit" (PeopleTools 8.59: PeopleCode API Reference)

RowSelect Event

The RowSelect event is initiated at the beginning of the component build process in any of the update action modes (Update, Update/Display All, Correction). RowSelect PeopleCode is used to filter out rows of data as they are being read into the component buffer. This event also occurs after a ScrollSelect or related function is executed.

Copyright © 1988, 2021, Oracle and/or its affiliates.

A DiscardRow function in RowSelect PeopleCode causes the Component Processor to skip the current row of data and continue to process other rows. A StopFetching statement causes the Component Processor to accept the current row of data, and then stop reading additional rows. If both statements are executed, the program skips the current row of data, and then stops reading additional rows.

PeopleSoft applications rarely use RowSelect, because it's inefficient to filter out rows of data after they've already been selected. Instead, screen out rows of data using search record views and effective-dated tables, which filter out the rows before they're selected. You could also use a ScrollSelect or related function to programmatically select rows of data into the component buffer.

In previous versions of PeopleTools, the Warning and Error statements were used instead of DiscardRow and StopFetching. Warning and Error statements still work as before in RowSelect, but their use is discouraged.

Note: In RowSelect PeopleCode, you can refer to record fields only on the record that is currently being processed. This event, and all its associated PeopleCode, does not initiate if run from a component interface.

RowSelect PeopleCode can be associated with record fields and component records.

Related Links

Row Select Processing

SaveEdit Event

The SaveEdit event is initiated whenever a user attempts to save the component. You can use SaveEdit PeopleCode to validate the consistency of data in component fields. Whenever a validation involves more than one component field, you should use SaveEdit PeopleCode. If a validation involves only one page field, use FieldEdit PeopleCode.

SaveEdit is not field-specific. It triggers associated PeopleCode on every row of data in the component buffers except rows flagged as deleted.

An Error statement in SaveEdit PeopleCode displays a message and redisplays the component without saving data. A Warning statement enables the user to click OK and save the data, or to click Cancel and return to the component without saving.

Use the SetCursorPos function to set the cursor position to a specific page field following a warning or error in SaveEdit, to show the user the field (or at least one of the fields) that is causing the problem. Make sure to call SetCursorPos before the error or warning, because these may terminate the PeopleCode program.

SaveEdit PeopleCode can be associated with record fields and components.

Related Links

Save Processing

"SetCursorPos" (PeopleTools 8.59: PeopleCode Language Reference)

 Copyright © 1988, 2021, Oracle and/or its affiliates.

SavePostChange Event

After the Component Processor updates the database, it initiates the SavePostChange event. You can use SavePostChange PeopleCode to update tables not in your component using the SQLExec built-in function.

An error or warning in SavePostChange PeopleCode causes a runtime error. Avoid errors and warnings in this event.

The system issues a SQL Commit statement after SavePostChange PeopleCode completes successfully.

If you are executing Workflow PeopleCode, keep in mind that if the Workflow PeopleCode fails, SavePostChange PeopleCode is not executed. If your component has both Workflow and SavePostChange PeopleCode, consider moving the SavePostChange PeopleCode to SavePreChange or Workflow.

If you are doing messaging, your Publish PeopleCode should go into this event.

SavePostChange does not execute if there is an error during the save. For example, if there is a data conflict error because another user updated the same data at the same time, SavePostChange does not execute.

Important! Never issue a SQL Commit or Rollback statement manually from within a SQLExec function. Let the Component Processor issue these SQL commands.

SavePostChange PeopleCode can be associated with record fields, components, and component records.

Related Links

Save Processing

"SQLExec" (PeopleTools 8.59: PeopleCode Language Reference)

SavePreChange Event

The SavePreChange event is initiated after SaveEdit completes without errors. SavePreChange PeopleCode provides one final opportunity to manipulate data before the system updates the database; for instance, you could use SavePreChange PeopleCode to set sequential high-level keys. If SavePreChange runs successfully, a Workflow event is generated, and then the Component Processor issues appropriate Insert, Update, or Delete SQL statements.

SavePreChange PeopleCode is not field-specific: it triggers PeopleCode on all fields and on all rows of data in the component buffer.

SavePreChange PeopleCode can be associated with record fields, components, and component records.

Related Links

Save Processing

SearchInit Event

The SearchInit event is generated just before a search, add, or data-entry dialog box is displayed. SearchInit triggers associated PeopleCode in the search key fields of the search record. This enables you to control processing before a user enters values for search keys in the dialog box. In some cases, you may

Copyright © 1988, 2021, Oracle and/or its affiliates.

want to set the value of the search dialog fields programmatically. For example, the following program in SearchInit PeopleCode on the component search key record field EMPLID sets the search key page field to the user's employee ID, makes the page field unavailable for entry, and enables the user to modify the user's own data in the component:

```
EMPLID = %EmployeeId;
Gray (EMPLID);
AllowEmplIdChg(True);
```

You can activate system defaults and system edits in the search page by calling SetSeachDefault and SetSearchEdit in SearchInit PeopleCode. You can also control the behavior of the search page, either forcing it to appear even if all the required keys have been provided, or by skipping it if possible, with the SetSeachDialogBehavior function. You can also force search processing to always occur by selecting the Force Search Processing check box in the component properties in PeopleSoft Application Designer.

Note: This event, and all its associated PeopleCode, does not initiate if run from a component interface.

SearchInit PeopleCode can be associated with record fields on search records and prompt table records and on component search records and component prompt table records.

SearchInit with Prompt Dialogs

Beginning with PeopleTools 8.50, you can put PeopleCode on the SearchInit and SearchSave events on the search keys of prompt table records. SearchInit and SearchSave events will only execute if the Allow Search Events for Prompt Dialogs check box was selected for the search key's record field properties in Application Designer. By default Allow Search Events for Prompt Dialogs is off.

Note: Search processing with prompt dialogs can affect performance. Oracle recommends that you limit the use of PeopleCode with prompt dialogs.

SearchInit PeopleCode Function Restrictions

You cannot use the following functions in SearchInit PeopleCode:

- DoModal, DoModalComponent, DoModalX, and DoModalXComponent
- Transfer, TransferExact, TransferNode, TransferPage, and TransferPortal

Related Links

Prompts
"SetSearchDefault" (PeopleTools 8.59: PeopleCode Language Reference)
Search Processing in Update Modes
Search Processing in Add Modes

SearchSave Event

SearchSave PeopleCode is executed for all search key fields on a search, add, or data-entry dialog box after a user clicks Search. This enables you to control processing after search key values are entered, but before the search based on these keys is executed. A typical use of this feature is to provide cross-field edits for selecting a minimum set of key information. This event is also used to force a user to enter a value in at least one field, even if it's a partial value, to help narrow a search for tables with many rows.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: SearchSave is not initiated when values are selected from the search list. To validate data entered in the search page, use the Component PreBuild event.

You can use Error and Warning statements in SearchSave PeopleCode to send the user back to the search page if the user entry does not pass validations implemented in the PeopleCode.

Note: This event, and all its associated PeopleCode, is not initiated if run from a component interface.

SearchSave PeopleCode can be associated with record fields and component search records.

Note: Do not use the %Menu system variable in this event. You may get unexpected results.

SearchSave with Prompt Dialogs

Beginning with PeopleTools 8.50, you can put PeopleCode on the SearchInit and SearchSave events on the search keys of prompt table records. SearchInit and SearchSave events will only execute if the Allow Search Events for Prompt Dialogs check box is selected for the search key's record field properties in Application Designer. By default Allow Search Events for Prompt Dialogs is off.

Note: Search processing with prompt dialogs can affect performance. Oracle recommends that you limit the use of PeopleCode with prompt dialogs.

Related Links

Prompts
Search Processing in Update Modes
Search Processing in Add Modes

Workflow Event

Workflow PeopleCode executes immediately after the SavePreChange event and before the database update that precedes the SavePostChange event. The Workflow event segregates PeopleCode related to workflow from the rest of the application's PeopleCode. Only PeopleCode related to workflow (such as TriggerBusinessEvent) should be in workflow programs. Your program should deal with the Workflow event only after any SavePreChange processing is complete.

Workflow PeopleCode is not field-specific: it triggers PeopleCode on all fields and on all rows of data in the component buffer.

WorkFlow PeopleCode can be associated with record fields and components.

Related Links

Save Processing
"Writing Workflow PeopleCode" (PeopleTools 8.59: Workflow Technology)

Copyright © 1988, 2021, Oracle and/or its affiliates.

PeopleCode Execution in Pages with Multiple Scroll Areas

Components with multiple levels can have multiple rows of data from multiple primary record definitions. You must know the order in which the system processes buffers for this data, because it applies PeopleCode in the same order.

The Component Processor uses a depth-first algorithm to process rows in multiple-scroll-area pages, starting with a row at level zero, drilling down to dependent rows on lower levels, and then working up the hierarchy until the system has processed all the dependent rows of the last row on the highest level.

Scroll Level One

When pages have only one scroll bar, the Component Processor processes record definitions at scroll level zero, and then all rows of data at scroll level one.

Data is retrieved for all rows with a single Select statement, and then it is merged with buffer structures.

Scroll Level Two

With scroll bars at multiple scroll levels, the system processes a single row of data at scroll level one, and then it processes all subordinate rows of data at scroll level two. After processing all subordinate data at scroll level two, it processes the next row for scroll level one, and all the subordinate data for that row. The system continues in this fashion until all data is processed.

Scroll Level Three

The Component Processor uses the same method for processing subordinate data at scroll level three. Data is retrieved for all rows with a single Select statement, and then merged with buffer structures. The Component Processor processes a single row of data at scroll level two, and it processes all subordinate data at scroll level three. After processing all subordinate data at scroll level three, it processes the next row for scroll level two and all the subordinates data for that row. The system continues in this fashion until all data is processed..

Related Links

Understanding Component Buffer Structure and Contents

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 7

PeopleCode and PeopleSoft Pure Internet Architecture

Considerations Using PeopleCode in PeopleSoft Pure Internet Architecture

Consider the following points when writing PeopleCode programs for PeopleSoft Pure Internet Architecture:

- To help your application run efficiently, avoid using field-level PeopleCode events (FieldEdit and FieldChange).

  Each field-level PeopleCode program requires a trip to the application server.

  The majority of PeopleCode programs run on the application server as part of the component build and save process. Do not hesitate to use PeopleCode for building and saving components.

- If a user changes a field but nothing on that field will cause a trip to the server, then default processing and FieldFormula PeopleCode do not run.

  This processing occurs only when another event causes a trip to the server.

  Other fields that depend on the first field using FieldFormula or default PeopleCode are not updated until the next time a server trip occurs.

- In applications that run on the PeopleSoft portal, external dynamic link information must be placed in RowInit PeopleCode.

  If external dynamic link information is placed in FieldChange PeopleCode, it will not work.

- Trips to the server are reduced when a component runs in deferred processing mode.

  Each trip to the server results in the page being completely refreshed on the browser, which may cause the display to flicker. It can also slow down your application. Deferred processing mode results in better performance.

Related Links

Deferred Processing Mode

Using PeopleCode with PeopleSoft Pure Internet Architecture

This section discusses how to:

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Use internet scripts.
- Use the field object Style property.
- Use the HTML area.
- Use HTML definitions and the GetHTMLText function.
- Use HTML definitions and the GetJavaScriptURL method.
- Use PeopleCode to populate key fields in search dialog boxes

Using Internet Scripts

An internet script is a specialized PeopleCode function that generates dynamic web content. Internet scripts interact with web clients (browsers) using a request-response paradigm based on HTTP.

Related Links

"Understanding Internet Script Classes" (PeopleTools 8.59: PeopleCode API Reference)

Using the Field Object Style Property

In PeopleSoft Application Designer, on the Use tab of the page definition properties, you can associate a page with a style sheet component.

The style sheet has several classes of styles defined for it. You can edit each style class to change the font, the color, the background, and so on. Then, you can dynamically change the style of a field using the Style field class property. The style sheet does not change, only the style class associated with that field changes.

The following example changes the style class of a field depending on a value entered by the user. This code is in the FieldChange event.

```
Local Field &field;

&field = GetField();

If TESTFIELD1 = 1 Then;
   &field.Style = "PSHYPERLINK";
End-If;

If TESTFIELD1 = 2 Then;
   &field.Style = "PSIMAGE";
End-If;
```

Image: Field with PSHYPERLINK style

The following examples show the fields with different styles:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Field with PSIMAGE style

The below mentioned is an image of a field with PSIMAGE style.

Related Links

"Understanding Style Sheets and Style Classes" (PeopleTools 8.59: Application Designer Developer's Guide)

"Understanding the Field Class" (PeopleTools 8.59: PeopleCode API Reference)

Using the HTML Area

Two methods are used to populate an HTML area control. Both require accessing the HTML area in the PeopleSoft Application Designer. One method is to select Constant on the HTML tab of the HTML page field properties dialog and enter HTML directly into the page field dialog.

The other method is to select Value on the HTML tab of the HTML page field properties dialog and associate the control with a record field. At runtime, populate that field with the text that you want to appear in the HTML area.

If you are using an HTML area to add form controls to a page, you can use GetParameter request class method in PeopleCode to get the user input from those controls.

Note: When you associate an HTML area control with a field, make sure the field is long enough to contain the data you want to pass to it. For example, if you associate an HTML area control with a field that is only 10 characters long, only the first 10 characters of your text will appear.

The following code populates an HTML area with a simple bulleted list. This code is in the RowInit event of the record field associated with the HTML control.

```
Local Field &HTMLField;

&HTMLField = GetField();
&HTMLField.Value = "<ul><li>Item one</li><li>Item two</li></ul>";
```

The following code is in the FieldChange event of a button. It populates an HTML area (associated with the record field CHART_DATA.HTMLAREA) with a simple list.

```
Local Field &HTMLField;

&HTMLField = GetRecord(Record.CHART_DATA).HTMLAREA;
&HTMLField.Value = "<ul><li>Item one</li><li>Item two</li></ul>";
```

The following code populates an HTML area (associated with the record DERIVED_HTML and the field HTMLAREA) with the output of the GenerateTree function:

```
DERIVED_HTML.HTMLAREA = GenerateTree(&TREECTL);
```

The following tags are unsupported by the HTML area control:

- Body
- Frame

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Frameset
- Form
- Head
- HTML
- Meta
- Title

Related Links

Using the GenerateTree Function
"Understanding HTML Definitions" (PeopleTools 8.59: Application Designer Developer's Guide)

Using HTML Definitions and the GetHTMLText Function

If you are using the same HTML text in more than one place or if it is a large, unwieldy string, you can create an HTML definition in PeopleSoft Application Designer, and then use the GetHTMLText function to populate an HTML area control.

The following is the HTML string to create a simple table:

```
<P>
<TABLE>

  <TR bgColor=#008000>
    <TD>
      <P><FONT color=#f5f5dc face="Arial, Helvetica, sans-serif"
      size=2>message 1 </FONT></P></TD></TR>
  <TR bgColor=#0000cd>
    <TD>
      <P><FONT color=#00ffff face="Arial, Helvetica, sans-serif"
      size=2>message 2</FONT></P></TD></TR>
</TABLE>
</P>
```

This HTML is saved to an HTML definition called TABLE_HTML.

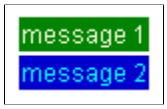
This code is in the RowInit event of the record field associated with the HTML area control:

```
Local Field &HTMLField;

&HTMLField = GetField();
&string = GetHTMLText(HTML.TABLE_HTML);
&HTMLField.Value = &string;
```

This code produces the following:

Image: HTML definition example

This example illustrates the fields and controls on the HTML definition example. You can find definitions for the fields and controls later on this page.



 Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"GetHTMLText" (PeopleTools 8.59: PeopleCode Language Reference)

Using HTML Definitions and the GetJavaScriptURL Method

HTML definitions can contain JavaScript programs in addition to HTML. If you have an HTML definition that contains JavaScript, use the GetJavaScriptURL Response method to access and execute the script.

This example assumes the existence in the database of a HTML definition called HelloWorld_JS that contains some JavaScript:

```
Function IScript_TestJavaScript()

   %Response.WriteLine("<script src= " |
%Response.GetJavaScriptURL(HTML.HelloWorld_JS) | "></script>");

End-Function;
```

Related Links

"GetJavaScriptURL" (PeopleTools 8.59: PeopleCode API Reference)

"Understanding HTML Definitions" (PeopleTools 8.59: Application Designer Developer's Guide)

Using PeopleCode to Populate Key Fields in Search Dialog Boxes

In a PeopleSoft Pure Internet Architecture application, you typically want users to directly access their own data. To facilitate this, you may want to use SearchInit PeopleCode to populate standard key fields in search page fields and then make the fields unavailable for entry. You might assign the search key field a default value based on the user ID or alias the user entered when signing in.

You must also call the AllowEmplIdChg function, which enables users to change their own data. This function takes a single Boolean parameter in which you pass True to allow employees to change their own data.

Here is a simple example of such a SearchInit program, using %EmployeeId to identify the user:

```
EMPLID = %EmployeeId;

Gray (EMPLID);

AllowEmplIdChg(True);
```

Calling DLL Functions on the Application Server

To support processes running on an application server, you can declare and call functions compiled in Microsoft Windows DLLs and in UNIX shared libraries (or shared objects, depending on the specific UNIX platform). You can do this either with a special PeopleCode declaration, or using the business interlink framework.

When you call out to a DLL using PeopleCode, on Microsoft Windows NT application servers, the DLL file has to be on the path. On UNIX application servers, the shared library file must be on the library path (as defined for the specific UNIX platform).

Copyright © 1988, 2021, Oracle and/or its affiliates.

The PeopleCode declaration and function call syntax remains unchanged. For example, the following PeopleCode could be used to declare and call a function LogMsg in an external library Testdll.dll on a Microsoft Windows client or a Windows application server, or a libtestdll.so on an UNIX application server. The UNIX shared library's extension varies by the specific UNIX platform.

```
Declare Function LogMsg Library "testdll" (string, string)
       Returns integer;

&res = LogMsg("\temp\test.log", "This is a test");
```

Sample Cross-Platform External Test Function

The following section describes and includes the C source code for a sample cross-platform test function: LogMsg. It is a basic function that opens a log file and appends a line to it. If you compile the code using a C++ compiler, the functions must be declared using external C to ensure C-language linkage.

The sample program also contains an interface function (LogMsg_intf). Prior to PeopleTools 8.52, this interface function was required for all non-Microsoft Windows environments. The interface function references a provided header file, pcmext.h. The interface function is passed type codes that can be optionally used for parameter checking.

Starting with PeopleTools 8.52, the interface function is required for the following environments only:

- HP-UX Itanium
- Solaris x86_64
- z/Linux

For all the other PeopleTools-supported UNIX platforms, the functions from the UNIX shared libraries can be used directly. To maintain backward compatibility, the interface functions are also supported on these UNIX platforms.

In the following sample test program, the interface function is compiled only when compiling for the specified non-Windows environments.

```
/*
 * Simple test function for calling from PeopleCode.
 * This is passed two strings, a file name and a message.
 * It creates the specified file and writes the message
 * to it.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifdef _WINDOWS
#define DLLEXPORT __declspec(dllexport)
#define LINKAGE __stdcall
#else
#define DLLEXPORT
#define LINKAGE
#endif

DLLEXPORT int LINKAGE LogMsg(char * fname, char * msg);

/*****************************************************
*  PeopleCode External call test function.           *
*                                                    *
*  Parameters are two strings (filename and message) *
*  Result is 0 if error, 1 if OK                     *
*                                                    *
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
*                                                     *
*  To call this function, the following PeopleCode is   *
*  used                                               *
*                                                     *
*  Declare Function LogMsg Library "testdll"          *
*  (string, string)                                   *
*       Returns integer;                              *
*                                                     *
*  &res = LogMsg("\temp\test.log", "This is a test");   *
*                                                     *
*****************************************************/

DLLEXPORT int LINKAGE LogMsg(char * fname, char * msg)
{
    FILE *fp;

    fp = fopen(fname, "a");         /* append */
    if (fp == NULL) return 0;

    fprintf(fp, "%s\n", msg);
    fclose(fp);
    return 1;
}


#ifndef _WINDOWS

/*****************************************************
*  Interface function.                                *
*                                                     *
*  This is not needed for Windows....                 *
*                                                     *
*****************************************************/

#include  "pcmext.h"
#include  "assert.h"

/* This interface function is required only for the following platforms:
   - HP-UX Itanium
   - Solaris x86_64
   - z/Linux
*/

void LogMsg_intf(int nParam, void ** ppParams, EXTPARAMDESC * pDesc)
{
    int    rc;

    /* Some error checking */
    assert(nParam == 2);
    assert(pDesc[0].eExtType == EXTTYPE_STRING
       && pDesc[1].eExtType == EXTTYPE_STRING
       && pDesc[2].eExtType == EXTTYPE_INT);

    rc = LogMsg((char *)ppParams[0],
         (char *)ppParams[1]);
    *(int *)ppParams[2] = rc;

}

#endif
```

Updating the Installation and PSOPTIONS Tables

When an application updates either the PSOPTIONS or the Installation table it must call UpdateSysVersion from the SavePreChange PeopleCode event. This way, updates take effect at the next

Copyright © 1988, 2021, Oracle and/or its affiliates.

page load. Otherwise, the change does not take effect at the client workstation until the user signs out and signs back in.

Important! Only a database administrator or the equivalent should change these tables.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 8

Using Methods and Built-In Functions

Understanding Restrictions on Method and Function Use

This section discusses:

- Think-time functions.
- WinMessage and MessageBox functions.
- Program execution with fields not in the data buffer.
- Errors and warnings.
- DoSave function.
- Record class database methods.
- SQL class methods and functions.
- Component interface restricted functions.
- SearchInit PeopleCode function restrictions.
- CallAppEngine function.
- ReturnToServer function.
- GetPage function.
- GetGrid function.
- Publish method.
- SyncRequest method.

Think-Time Functions

Think-time functions suspend processing either until the user has taken some action (such as clicking a button in a message box) or until an external process has run to completion (for example, a remote process).

Avoid think-time functions in the following PeopleCode events:

- SavePreChange.
- Workflow.
- RowSelect.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- SavePostChange.
- Any PeopleCode event that executes as a result of a ScrollSelect, ScrollSelectNew, RowScrollSelect, or RowScrollSelectNew function call.
- Any PeopleCode event that executes as a result of a Rowset class Select or SelectNew method.

Violation of this rule can result in application failure.

The following are think-time functions:

- Calls to an external DLL.
- DoCancel.
- DoModal and DoModalComponent.
- Exec (this is think-time only when synchronous).
- File attachment functions AddAttachment, DetachAttachment, MAddAttachment, and ViewAttachment.
- CropImage and InsertImage.
- Object functions, such as CreateObject, ObjectDoMethod, ObjectSetProperty, and ObjectGetProperty (these are think-time only when the object requires user action).
- Prompt.
- RemoteCall.
- RevalidatePassword.
- WinExec (think-time only when synchronous).
- WinMessage and MessageBox (depending on the *style* parameter).

WinMessage and MessageBox Functions

The WinMessage and MessageBox functions sometimes behave as think-time functions, depending on the value passed in the function's *style* parameter, which controls, among other things, the number of buttons displayed in the message dialog box.

Note: The *style* parameter is ignored if the message has any severity other than Message.

Here is the syntax of both functions:

```
MessageBox(style, title, message_set, message_num, default_txt [, paramlist])

WinMessage(message [, style] [, title])
```

Note: The WinMessage function is supported for compatibility with previous releases of PeopleTools. New applications should use MessageBox instead.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

If the *style* parameter specifies more than one button, the function behaves as a think-time function and is subject to the same restrictions as other think-time functions (that is, it should never be used from SavePreChange through SavePostChange PeopleCode, or in RowSelect).

If the *style* parameter specifies a single button (that is, the OK button), then the function can be called in any PeopleCode event.

Note: In the Microsoft Windows client, MessageBox dialog boxes include an Explain button to display more detailed information stored in the message catalog. The presence of the Explain button has no bearing on whether a message box behaves as a think-time function.

The *style* parameter is optional in WinMessage. If *style* is omitted, WinMessage displays OK and Cancel buttons, which causes the function to behave as a think-time function. To avoid this situation, always pass an appropriate value in the WinMessage *style* parameter.

The following table shows the values that can be passed in the *style* parameter. To calculate the value to pass, make one selection from each category in the table, then add the selections.

Category	*Value*	*Constant*	*Meaning*
Buttons	0	%MsgStyle_OK	The message box contains one button: OK.
Buttons	1	%MsgStyle_OKCancel	The message box contains two buttons: OK and Cancel.
Buttons	2	%MsgStyle_ AbortRetryIgnore	The message box contains three buttons: Abort, Retry, and Ignore.
Buttons	3	%MsgStyle_YesNoCancel	The message box contains three buttons: Yes, No, and Cancel.
Buttons	4	%MsgStyle_YesNo	The message box contains two buttons: Yes and No.
Buttons	5	%MsgStyle_RetryCancel	The message box contains two buttons: Retry and Cancel.

Note: The following values for *style* can only be used in the Microsoft Windows client. They have no affect in PeopleSoft Pure Internet Architecture.

Category	*Value*	*Constant*	*Meaning*
Default Button	0	%MsgDefault_First	The first button is the default.
Default Button	256	%MsgDefault_Second	The second button is the default.
Default Button	512	%MsgDefault_Third	The third button is the default.
Icon	0	%MsgIcon_None	None

Copyright © 1988, 2021, Oracle and/or its affiliates.

Category	*Value*	*Constant*	*Meaning*
Icon	16	%MsgIcon_Error	A stop-sign icon appears in the message box.
Icon	32	%MsgIcon_Query	A question-mark icon appears in the message box.
Icon	48	%MsgIcon_Warning	An exclamation-point icon appears in the message box.
Icon	64	%MsgIcon_Info	An icon consisting of a lowercase letter *i* in a circle appears in the message box.

Related Links

"MessageBox" (PeopleTools 8.59: PeopleCode Language Reference)

"WinMessage" (PeopleTools 8.59: PeopleCode Language Reference)

Program Execution with Fields Not in the Data Buffer

Under certain conditions, when you access a field that is not in the data buffer, a portion of your PeopleCode program is skipped. The skip occurs when:

- The reference is in the Import Manager.
- The reference is from the FieldDefault or FieldFormula events.

After the call to the invalid field, execution skips to the next top-level statement. Top-level statements are not nested inside other statements. The start of a PeopleCode program is a top-level statement. Nesting begins with the first conditional statement (such as While or If) or the first function call.

For example, if your code is executing in a function and inside an If … then … end-if statement, and it runs into the skip conditions, the next statement executed is the one after the End-if statement, still inside the function.

Errors and Warnings

Errors and warnings should not be used in FieldDefault, FieldFormula, RowInit, FieldChange, RowInsert, SavePreChange, WorkFlow, and SavePostChange PeopleCode events. An error or warning in these events causes a runtime error that forces cancellation of the component.

Related Links

"Warning" (PeopleTools 8.59: PeopleCode Language Reference)

"Error" (PeopleTools 8.59: PeopleCode Language Reference)

DoSave Function

DoSave can be used in the following PeopleCode events only: FieldEdit, FieldChange, or ItemSelected (for menu items in popup menus only).

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"DoSave" (PeopleTools 8.59: PeopleCode Language Reference)

Record Class Database Methods

You use the following record class methods to update the database:

- Delete
- Insert
- Save
- Update

Only use these methods in the following events (events that allow database updates):

- SavePreChange
- WorkFlow
- SavePostChange
- FieldChange
- Application Engine PeopleCode action

Related Links

"Understanding Record Class" (PeopleTools 8.59: PeopleCode API Reference)

SQL Class Methods and Functions

Use the SQL class to update the database. Use these functions and methods only in the following events (events that allow database updates):

- SavePreChange
- WorkFlow
- SavePostChange
- FieldChange
- Application Engine PeopleCode action

Component Interface Restricted Functions

PeopleCode events and functions that relate exclusively to the page interface (the 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.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- 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.

- Save in the middle of a transaction.

  DoSave cannot be used.

- The page Activate event cannot be used.

When executed using a component interface, these functions do nothing and return a default value. In addition, using the Transfer function terminates the current PeopleCode program.

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;
```

Related Links

"Generating PeopleCode Templates to Access Component Interfaces" (PeopleTools 8.59: Component Interfaces)

SearchInit PeopleCode Function Restrictions

You cannot use the following functions in SearchInit PeopleCode:

- DoModal, DoModalComponent, DoModalX, and DoModalXComponent
- Transfer, TransferExact, TransferNode, TransferPage, and TransferPortal

CallAppEngine Function

Use the CallAppEngine function only in events that allow database updates, because, generally, if you are calling Application Engine, you intend to perform database updates. This category of events includes the following PeopleCode events:

- SavePreChange (Page)
- SavePostChange (Page)
- Workflow

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- FieldChange

CallAppEngine cannot be used in a Application Engine PeopleCode action. If you need to access one Application Engine program from another Application Engine program, use the CallSection action.

Related Links

"Running Application Engine Programs" (PeopleTools 8.59: Application Engine)

ReturnToServer Function

The ReturnToServer function returns a value from a PeopleCode application messaging program to the publication or subscription server. You would use this in either your publication or subscription routing code, not in one of the standard Component Processor events.

Related Links

"Understanding Managing Messages" (PeopleTools 8.59: Integration Broker)

GetPage Function

The GetPage function cannot be used until after the Component Processor has loaded the page. You should not use this function in an event prior to the PostBuild event.

Related Links

Component Processor Behavior

"GetPage" (PeopleTools 8.59: PeopleCode Language Reference)

GetGrid and GetAnalyticGrid Functions

PeopleSoft builds a grid one row at a time. Because the grid and AnalyticGrid classes apply to a complete grid, you cannot use either the GetGrid or GetAnalyticGrid functions in an event prior to the Activate event.

Related Links

Component Processor Behavior

"GetGrid" (PeopleTools 8.59: PeopleCode Language Reference)

"Understanding the Analytic Calculation Engine Classes" (PeopleTools 8.59: PeopleCode API Reference)

Publish Method

If you are using PeopleSoft Integration Broker, your sending PeopleCode should go in the SavePostChange event, for either the record or the component.

Related Links

"Understanding Managing Messages" (PeopleTools 8.59: Integration Broker)

Copyright © 1988, 2021, Oracle and/or its affiliates.

SyncRequest Method

If you are using PeopleSoft Integration Broker, your SyncRequest PeopleCode should go in the SavePostChange event, for either the record or the component.

Related Links

"Understanding Managing Messages" (PeopleTools 8.59: Integration Broker)

Implementing Modal Transfers

This section provides an overview of modal transfers and discusses how to implement modal transfers.

Understanding Modal Transfers

When you use modal transfers to transfer from one component (the *originating* component) to another component (the *modal* component), the user must click the OK or Cancel buttons on the modal component before returning to the originating component.

Modal transfers provide some control over the order in which the user fills in pages, which is useful where data in the originating component can be derived from data entered by the user into the modal component.

Limit use of this feature, as it forces users to complete interaction with the modal page before returning to the main component.

Note: Modal transfers cannot be initiated from SearchInit PeopleCode.

A modal component resembles a Microsoft Windows modal dialog box. It displays three buttons: OK, Cancel, and Apply. No toolbars or windows are available while the modal component has the focus. The OK button saves changes to the modal component and returns the user to the originating component. The Apply button saves changes to the modal component without returning to the originating component. The Cancel button returns the user to the originating component without saving changes to the modal component.

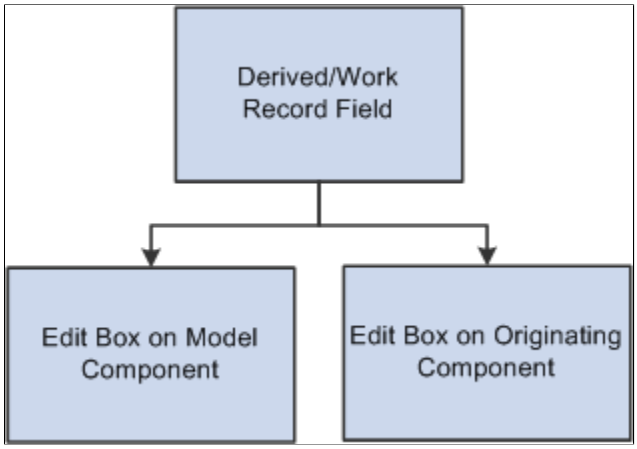
Modal components are generally smaller than the page from which they are invoked. Remember that OK and Cancel buttons are added at runtime, thus increasing the size of the pages.

The originating component and the modal component share record fields in a derived/work record called a *shared work record.* The derived/work fields of this record provide the two components with an area in memory where they can share data. Edit boxes in both components are associated with the same derived/work field, so that changes made to this field in the originating component are reflected in the modal component, and vice versa. The following diagram illustrates this shared memory:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Edit boxes on the originating and modal components share the same data

The following diagram explains that the Edit boxes on the originating and modal components share the same data.



Edit boxes associated with the same derived/work fields must be placed at level zero in both the originating component and the modal component.

You can use the shared fields to:

- Pass values assigned to the search keys in the modal component search record.

  If these fields are missing or invalid, the search page appears, enabling the user to enter search keys.

- Pass other values from the originating component to the modal component.
- Pass values back from the modal component to the originating component.

Modeless Windows

In addition to modal secondary windows, you can create modeless secondary windows. A modeless window is different from a modal window launched by any of the DoModal* functions in that its state is separate from that of the parent, launching component. When a modeless window is launched from a classic component using the TransferModeless function, the modeless window does not mask the parent window, which allows the user to update the modeless and parent window from the same browser session at the same time.

However, when a modeless window is launched from a fluid component using either the ViewContentURLModeless or ViewURLModeless functions, the parent window is masked and cannot be accessed until the modeless window is dismissed. In this respect and from the user's perspective, a modeless window launched from a fluid component operates similar to a modal window; but programmatically, its state is separate from that of the parent, launching component.

Similar to modal, secondary windows opened by any of the DoModal* functions, the modeless window does not include the browser title bar, browser menus, and the browser tool bars, status bar, and tool icons associated with most browser windows.

Note: While the title bar of a modeless window includes an X (or cancel) button, it cannot include any custom buttons.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Important! Only one active child modeless window can be open at one time. Therefore, after opening a modeless child window from the parent, you cannot open a second modeless child window from that modeless window. However, you can open a modal window from that modeless window.

Related Links

"TransferModeless" (PeopleTools 8.59: PeopleCode Language Reference)
"ViewContentURLModeless" (PeopleTools 8.59: PeopleCode Language Reference)
"ViewURLModeless" (PeopleTools 8.59: PeopleCode Language Reference)

Implementing Modal Transfers

Any component accessible through an application menu system can be accessed using a modal transfer. However, to implement a modal transfer, you must modify pages in both the originating component and the modal component. After these modifications are complete, you can implement the modal transfer using the DoModalComponent function from a page in the originating component.

Before beginning this process, you should answer the following questions:

- Should the originating component provide search key values for the modal component?

  If so, what are the search keys? (Check the modal component's search record.)

- Does the originating component need to pass any data to the modal component?

  If so, what record fields are needed to store this data?

- Does the modal component need to pass any data back to the originating component?

  If so, what record fields are needed to store this data?

To implement a modal transfer:

1. Create derived/work record fields for sharing data between the originating and modal components.

   Create a new derived/work record or open an existing derived/work record. If suitable record fields exist, you can use them; otherwise create new record fields for any data that needs to be shared between the components. These can be search keys for the modal component, data to pass to the modal component, or data to pass back to the originating component.

2. Add derived work fields to the level-zero area of the originating component.

   Add one edit box for each of the derived/work fields that you need to share between the originating and modal components to the level-zero area of the page from which the transfer will take place. You probably want to make the edit boxes invisible.

3. Add the same derived work fields to the level-zero area of the modal component.

   Add one edit box for each of the edit boxes that you added in the previous step to the level-zero area of the page to which you are transferring. You probably want to make the edit boxes invisible.

4. Add PeopleCode to pass values into the derived/work fields in the originating component.

   To provide search key values or pass data to the modal page, write PeopleCode that assigns appropriate values to the derived/work fields before DoModalComponent is called.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

For example, if the modal component search key is PERSONAL_DATA.EMPLID, you could place the following assignment statement in the derived/work field's RowInit event:

```
EMPLID = PERSONAL_DATA.EMPLID
```

You also might assign these values in the same program where DoModalComponent is called.

5. Add PeopleCode to access and change the derived/work fields in the modal component.

   No PeopleCode is required to pass search key values during the search. However, if other data has been passed to the modal component, you may need PeopleCode to access and use the data. You may also need to assign new values to the shared fields so that they can be used by the originating component.

   It is possible that the component was accessed through the menu system and not through a modal transfer. To write PeopleCode that runs only in the component when it is running modally, use the IsModalComponent function:

```
If IsModalComponent() Then
   /* PeopleCode for modal execution only.  */
End-If
```

6. Add PeopleCode to access changed derived/work fields in the originating component.

   If the modal component has altered the data in the shared work fields, you can write PeopleCode to access and use the data after DoModalComponent has executed.

Note: You can use the EndModalComponent function as a programmatic implementation of the OK and Cancel buttons.

Related Links

"Transfer and Modal Functions" (PeopleTools 8.59: Fluid User Interface Developer’s Guide)

"IsModal" (PeopleTools 8.59: PeopleCode Language Reference)

"EndModalComponent" (PeopleTools 8.59: PeopleCode Language Reference)

Implementing the Multi-Row Insert Feature

Enabling the multi-row insert feature in grids or scroll areas can reduce response times for transactions that usually require entering many rows of data. With the multi-row feature, users specify the number of rows to add to a grid or scroll area, and empty rows appear for data entry.

This feature cannot be used with effective-dated grids or scroll areas. In addition, the feature may not apply if the entire row is populated using PeopleCode, especially if the data is copied from prior rows. If the feature does apply in this case, the default value of the ChangeOnInit property can be used (the default value is True, which means any PeopleCode updates done in the RowInit or RowInsert events set the IsChanged and IsNew properties to True).

To use the multi-row insert feature:

1. Specify deferred mode processing.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The multi-row feature reduces transaction times by eliminating excess server trips. To take full advantage of this feature, the transaction should be set to execute in deferred mode. Deferred mode should be set for the component, all pages in the component, and all fields on those pages.

2. Enable the multi-row feature.

   For each grid or scroll area where appropriate, select the Allow Multi-row Insert check box under the Use tab in the grid or scroll area property sheet.

3. Add ChangeOnInit PeopleCode.

   Setting the ChangeOnInit property for a rowset to False enables PeopleCode to modify data in the rowset during RowInit and RowInsert events without flagging the rows as changed. This ensures that only user changes cause the affected row to be saved.

   Note: Each rowset that is referenced by a grid or scroll area with the multi-row feature enabled should have the ChangeOnInit property for the rowset set to False. This includes lower-level rowsets. In addition, this property must be set prior to any RowInsert or RowInit PeopleCode for the affected row.

4. Empty rows at save.

   After a transaction is saved, any empty rows are discarded before the page is redisplayed to the user. An empty row means that the user did not access the data because PeopleCode or record defaults may have been used to initialize the row for the initial display.

   Note: PeopleCode save processing (SaveEdit and SavePreChange) PeopleCode executes for all rows in the buffer (including the empty ones). Therefore, SaveEdit and SavePreChange PeopleCode should be coded so that it is executed only if the field contains data, or if the row properties IsNew and IsChanged are both True. An alternative method is adding PeopleCode in the first save program in the component, to explicitly delete any row based on the IsNew and IsChanged properties. If you choose this method, then rows should be deleted from the bottom of the data buffer to the top (last row first).

Related Links

Deferred Processing Mode
"ChangeOnInit" (PeopleTools 8.59: PeopleCode API Reference)

Using the ImageReference Field

To associate an image definition with a field at runtime, the field has to be of type ImageReference. An example of this is referencing a red, yellow, or green light on a page, depending on the context.

To change the image value of an ImageReference field:

1. Create a field of type ImageReference.
2. Create the images you want to use.

   These images must be saved in PeopleSoft Application Designer as image definitions.

3. Add the field to a record that will be accessed by the page.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

4. Add an image control to the page and associate the image control with the ImageReference field.
5. Assign the field value.

   Use the keyword **Image** to assign a value to the field. For example:

```
Local Record &MyRec;
Global Number &MyResult;

&MyRec = GetRecord();
If &MyResult Then
   &MyRec.MyImageField.Value = Image.THUMBSUP;
Else
   &MyRec.MyImageField.Value = Image.THUMBSDOWN;
End-If;
```

Related Links

"Creating New Field Definitions" (PeopleTools 8.59: Application Designer Developer's Guide)

Inserting Rows Using PeopleCode

When inserting rows using PeopleCode, you can either use the Insert method with a record object or create a SQL Insert statement using the SQL object. If you do a single insert, use the Record Insert method. If you are in a loop and, therefore, calling the insert more than once, use the SQL object. The SQL object uses dedicated cursors and, if the database you are working with supports it, bulk insert.

A *dedicated cursor* means that the SQL gets compiled only once on the database, so PeopleTools looks for the meta-SQL only once. This can increase performance.

For *bulk insert,* inserted rows are buffered and sent to the database server only when the buffer is full or a commit occurs. This reduces the number of round-trips to the database. Again, this can increase performance.

The following is an example of using the Record Insert method:

```
&REC = CreateRecord(Record.GREG);
&REC.DESCR.Value = "Y" | &I;
&REC.EMPLID.Value = &I;
&REC.Insert();
```

The following is an example using a SQL object to insert rows:

```
&SQL = CreateSQL("%INSERT(:1)");
&REC = CreateRecord(Record.GREG);
&SQL.BulkMode = True;
For &I = 1 to 10
   &REC.DESCR.Value = "Y" | &I;
   &REC.EMPLID.Value = &I;
   &SQL.Execute(&REC);
End-For;
```

Related Links

"Insert" (PeopleTools 8.59: PeopleCode API Reference)

"Understanding SQL Class" (PeopleTools 8.59: PeopleCode API Reference)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Using OLE Functions

This section provides an overview of OLE functions and discusses how to:

- Use the Object data type.
- Share a single object instance.
- Use the Exec and WinExec functions.

Understanding OLE Functions

OLE automation is a Microsoft Windows protocol that enables one application to control another's operation. The applications communicate by means of an OLE object. One of the applications (called the automation server) makes available an OLE object that the second application (the client application) can use to send commands to the server application. The OLE object has methods associated with it, each of which corresponds to an action that the server application can perform. The client runs the methods, which cause the server application to perform the specified actions.

PeopleCode includes a set of functions that enable your PeopleCode program to be an OLE client. You can connect to any application that's registered as an OLE automation server and invoke its methods.

Note: Differences in Microsoft Windows applications from one release to the next (that is, properties becoming methods or vice versa) can cause problems with the ObjectGetProperty, ObjectSetProperty and ObjectDoMethod functions.

See the documentation for the OLE-automated application.

Related Links

"Functions by Category" (PeopleTools 8.59: PeopleCode Language Reference)

Using the Object Data Type

To support OLE, PeopleCode has a special data type, Object, which it uses for OLE objects. The purpose of the Object data type is to hold OLE objects during the course of a session so that you can run its methods. You cannot store Object data for any extended period of time.

Important! Object is a valid data type for variables, but not for record fields. Because OLE objects are by nature temporary, you cannot store Object data in a record field, including work record fields.

Some OLE object methods return data to the client. You can use such methods to get data from the automation server, if the method returns the data in a PeopleCode-supported data type. If the method returns data in an spreadsheet, for example, you cannot accept the data, because PeopleCode does not support spreadsheets.

Sharing a Single Object Instance

When you need the services of an OLE automation server, you create an instance of its OLE object, using the CreateObject function. After you have the object, you can run its methods as often as you like. You do not need to create a new instance of the object each time.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

In a typical scenario, you have a PeopleSoft component that needs to access Microsoft Excel or Word, or some other automation server, perhaps one you have created yourself. Various PeopleCode programs associated with the component must run OLE object methods.

Rather than create a new instance of the OLE object in each PeopleCode program, you should create one instance of the OLE object in a PeopleCode program that runs when the component starts (such as RowInit) and assign it to a global variable. Then, any PeopleCode program can reference the object and invoke its methods.

Using the Exec and WinExec Functions

The WinExec and Exec built-in functions provide another way to start another application from PeopleCode. Unlike the OLE functions, however, Exec and WinExec do not enable you to control what actions the application takes after you start it. You can start the application, and if you use the synchronous option you can find out when it closes, but you cannot affect its course or receive any data in return.

WinExec is appropriate in two situations:

- When you want to start an application and continue processing.
- When you have a short, unvarying process that you want to run, such as copying a file.

The Exec function, unlike WinExec and the OLE functions, is not Microsoft Windows-specific. You can run it on an application server to call an executable on the application server platform, which in PeopleTools release 7 and later can be either Windows NT or UNIX.

Important! If you use the WinExec function with its synchronous option, the PeopleCode program (and the PeopleSoft application) remain paused until the called program is complete. If you start a program that waits for user input, such as Notepad, the application appears to hang until the user closes the called program. The synchronous option also imposes limits on the PeopleCode.

Related Links

"Exec" (PeopleTools 8.59: PeopleCode Language Reference)
"WinExec" (PeopleTools 8.59: PeopleCode Language Reference)

Using the Select and SelectNew Methods

This section provides an overview of the Select method and discusses how to use the Select method.

Understanding the Select and SelectNew Methods

The Select and SelectNew methods, like the ScrollSelect functions, enable you to control the process of selecting data into a page scroll area. The Select method selects rows from a table or view and adds the rows to either a rowset or a row. Let’s call the record definition of the table or view that it selected from the *select record.* Let’s call the primary database record of the top-level rowset object executing the method the *default scroll record.*

Copyright © 1988, 2021, Oracle and/or its affiliates.

The select record can be the same as the default scroll record, or it can be a different record definition that has the same key fields as the default scroll record. If you define a select record that differs from the default scroll record, you can restrict the number of fields loaded into the buffers by including only the fields you actually need.

You can use these methods only with a rowset. A rowset can be thought of as a page scroll area.

A level zero rowset starts at the top level of the page, level zero, and contains all the data in the component buffers. A child rowset is contained by an upper-level rowset, also called the parent rowset. For example, a level one rowset could be considered the child rowset of a level zero, or parent, rowset. Or a level two rowset could be the child rowset of a level one rowset. The data contained in a child rowset depends on the row of the parent rowset.

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.

The Select method 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.

The Select method also accepts an optional SQL string that can contain a Where clause restricting the number of rows selected into the scroll area. The SQL string can also contain an Order By clause, enabling you to sort the rows.

The Select and SelectNew methods generate an SQL Select statement at runtime, based on the fields in the select record and the Where clause passed to them in the function call. This gives Select and SelectNew a significant advantage over the SQLExec function: they enable you to change the structure of the select record without affecting the PeopleCode program, unless the field affected is referred to in the Where clause string. This can make the application easier to maintain.

Also, if you use one of the meta-SQL constructs or shortcuts in the Where clause, such as %KeyEqual or %List, even if a field has changed, you do not have to change your code.

Unlike the ScrollSelect functions, neither Select or SelectNew allow you to operate in turbo mode.

Note: In addition to these methods, the SelectByKey record class method enables you to select into a record object. If you're only interested in selecting a single row of data, consider this method instead.

Related Links

Understanding Data Buffer Access

Using the Select Method

The syntax of the Select method is:

```
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.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

This syntax does the following:

- Specifies an optional child rowset into which to read the selected rows.
- Specifies the select record from which to select rows.
- Passes a string containing a SQL Where clause to restrict the selection of rows or an Order By clause to sort the rows, or both.

Specifying Child Rowsets

The first part of the Select syntax specifies a child rowset into which rows are selected. This parameter is optional.

If you do not specify any child rowsets in *paramlist*, Select selects from a SQL table or view specified by *selrecord* into the rowset object executing the method. For example, suppose you've instantiated a level one rowset &BUS_EXPENSES_PER. The following would select into this rowset:

```
Local Rowset &BUS_EXPENSES_PER;

&BUS_EXPENSES_PER = GetRowset(SCROLL.BUS_EXPSNESE_PER);
&BUS_EXPENSES_PER.Select(RECORD.BUS_EXPENSE_VW,
"WHERE SETID = :1 and CUST_ID = :2", SETID, CUST_ID);
```

If the rowset executing the method is a level zero rowset, and you specify the Select method without specifying any child rowsets with *paramlist,*, the method reads only a single row, because only one row is allowed at level zero.

Note: For developers familiar with previous releases of PeopleCode: In this situation, the Select method is acting like the RowScrollSelect function.

If you specify a child rowset in *paramlist,* the Select method 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.

In the following example, rows are selected into a child rowset BUS_EXPENSE_DTL, matching level-one keys, and with the charge amount equal to or exceeding 200, sorting by that amount:

```
Local Record &REC_EXP;
Local Rowset &BUS_EXPENSE_PER;

&REC_EXP = GetRecord(RECORD.BUSINESS_EXPENSE_PER;
&BUS_EXPENSE_PER = GetRowset(SCROLL.BUS_EXPSNESE_PER);
&BUS_EXPENSE_PER.Select(SCROLL.BUS_EXPENSE_DTL,
RECORD.BUS_EXPENSE_DTL, "WHERE %KeyEqual(:1) AND EXPENSE_AMT
>= 200 ORDER BY EXPENSE_AMT", &REC_EXP);
```

Specifying the Select Record

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 executing the method, or it can be a different record definition that has compatible fields.

The select record must be defined in PeopleSoft Application Designer and be a built SQL table or view (using Build, Project), unless the select record is the same record as the primary database record associated with the rowset.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The select record 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.

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 work station by including only the fields you actually need.

The Where Clause

The Select method 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 SelectNew 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.

To avoid errors, the Where clause should explicitly select matching key fields on parent and child rows. You do this using the %KeyEqual meta-SQL.

Select Like RowScrollSelect

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.

Note: For developers familiar with previous releases of PeopleCode: In this situation, the Select method is acting like the RowScrollSelect function.

If you qualify the lower-level rowset so that it only returns one row, it acts like the RowScrollSelect method.

```
&RSLVL1 = GetRowset(SCROLL.PHYSICAL_INV);
&RSLVL2 = &RSLVL1(&PHYSICAL_ROW).GetRowset(SCROLL.PO_RECEIVED_INV);
&REC2 = &RSLVL2.PO_RECEIVED_INV;
If &PO_ROW = 0 Then
   &RSLVL2.Select(PO_RECEIVED_INV, "WHERE %KeyEqual(:1)
and qty_available > 0", &REC2);
End-if;
```

Related Links

"%KeyEqual" (PeopleTools 8.59: PeopleCode Language Reference)

Using Standalone Rowsets

This section provides an overview of standalone rowsets and discusses how to:

- Use the Fill rowset method.
- Use the CopyTo rowset method.
- Add child rowsets.
- Use standalone rowsets to write a file.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Use standalone rowsets to read a file.

Understanding Standalone Rowsets

Standalone rowsets are not associated with a component or page. Use them to work on data that is not associated with a component or page buffer. In earlier releases, this was done using derived work records. You still must build work pages.

Note: Standalone rowsets are not connected to the Component Processor, so there are no database updates when they are manipulated. Delete and insert actions on these types of rowsets are not automatically applied at save time.

As with any PeopleTools object, the scope of standalone rowsets can be Local, Global, or Component. Consider the following code:

```
Local Rowset &MYRS;

&MYRS = CreateRowset(RECORD.SOMEREC);
```

This code creates a rowset with SOMEREC as the level zero record. The rowset is unpopulated. Functionally, it is the same as an array of rows.

Using the Fill Method

The Fill method fills the rowset by reading records from the database, by first flushing out all the contents of the rowset. A Where clause must be provided to get all the relevant rows.

```
Local Rowset &MYRS;
Local String &EMPLID;

&MYRS = CreateRowset(RECORD.SOMEREC);
&EMPLID = '8001';

&MYRS.Fill("where EMPLID = :1", &EMPLID);
```

Use the Fill method with standalone rowsets, created using the CreateRowset function. Do not use Fill with component buffer rowsets.

Using the CopyTo Method

The CopyTo method copies like-named fields from a source rowset to a destination rowset. To perform the copy, it uses like-named records for matching, unless specified. It works on any rowset except the Application Engine state records. The following is an example:

```
Local Rowset &MYRS1, MYRS2;
Local String &EMPLID;

&MYRS1 = CreateRowset(RECORD.SOMEREC);
&MYRS2 = CreateRowset(RECORD.SOMEREC);

&EMPLID = '8001';

&MYRS1.Fill("where EMPLID = :1", &EMPLID);
&MYRS1.CopyTo(&MYRS2);
```

After running the previous code segment, &MYRS2 contains that same data as &MYRS1. Both &MYRS1 and &MYRS2 were built using like-named records.

Copyright © 1988, 2021, Oracle and/or its affiliates.

To use the CopyTo method where there are no like-named records, you must specify the source and destination records. The following code copies only like-named fields:

```
Local Rowset &MYRS1, MYRS2;
Local String &EMPLID;

&MYRS1 = CreateRowset(RECORD.SOMEREC1);
&MYRS2 = CreateRowset(RECORD.SOMEREC2);

&EMPLID = '8001';

&MYRS1.Fill("where EMPLID = :1", &EMPLID);
&MYRS1.CopyTo(&MYRS2, RECORD.SOMEREC1, RECORD.SOMEREC2);
```

Adding Child Rowsets

The first parameter of the CreateRowset method determines the top-level structure. If you pass the name of the record as the first parameter, the rowset is based on a record. You can also base the structure on a different rowset. In the following example, &MYRS2 inherits the structure of &MYRS1:

```
Local Rowset &MYRS1, MYRS2;

&MYRS1 = CreateRowset(RECORD.SOMEREC1);
&MYRS2 = CreateRowset(&MYRS1);
```

To add a child rowset, suppose the following records describe a relationship. The structure is made up of three records:

- PERSONAL_DATA
- BUS_EXPENSE_PER
- BUS_EXPENSE_DTL

To build rowsets with child rowsets, use code like the following:

```
Local Rowset &rsBusExp, &rsBusExpPer, &rsBusExpDtl;

&rsBusExpDtl = CreateRowset(Record.BUS_EXPENSE_DTL);
&rsBusExpPer = CreateRowset(Record.BUS_EXPENSE_PER, &rsBusExpDtl);
&rsBusExp = CreateRowset(Record.PERSONAL_DATA, &rsBusExpPer);
```

Another variation is

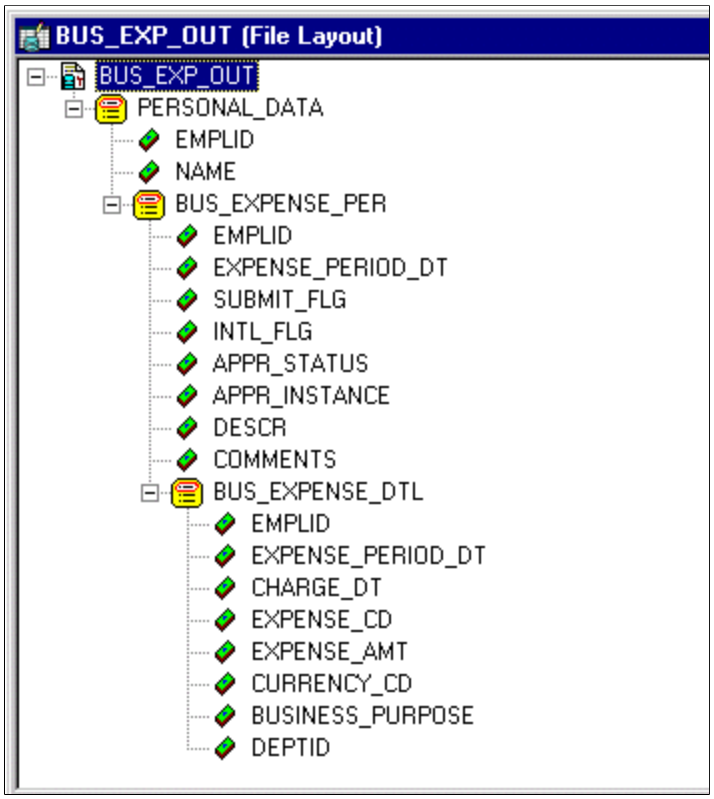
```
&rsBusExp = CreateRowset(Record.PERSONAL_DATA,
CreateRowset(Record.BUS_EXPENSE_PER,
CreateRowset(Record.BUS_EXPENSE_DTL)));
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Using Standalone Rowsets to Write a File

Image: File layout example

The following is an example of using standalone rowsets along with a file layout to write a file:



The following example writes a file using a file layout that contains parent-child records:

```
Local File &MYFILE;
Local Rowset &rsBusExp, &rsBusExpPer, &rsBusExpDtl;
Local Record &rBusExp, &rBusExpPer, &rBusExpDtl;
Local SQL &SQL1, &SQL2, &SQL3;

&rBusExp = CreateRecord(Record.PERSONAL_DATA);
&rBusExpPer = CreateRecord(Record.BUS_EXPENSE_PER);
&rBusExpDtl = CreateRecord(Record.BUS_EXPENSE_DTL);

&rsBusExp = CreateRowset(Record.PERSONAL_DATA,
CreateRowset(Record.BUS_EXPENSE_PER,
CreateRowset(Record.BUS_EXPENSE_DTL)));
&rsBusExpPer = &rsBusExp.GetRow(1).GetRowset(1);

&MYFILE = GetFile("c:\temp\BUS_EXP.out", "W", "UTF8", %FilePath_Absolute);
&MYFILE.SetFileLayout(FileLayout.BUS_EXP_OUT);

&EMPLID = "8001";

&SQL1 = CreateSQL("%selectall(:1) where EMPLID = :2", &rBusExp, &EMPLID);
&SQL2 = CreateSQL("%selectall(:1) where EMPLID = :2", &rBusExpPer, &EMPLID);

While &SQL1.Fetch(&rBusExp)
   &rBusExp.CopyFieldsTo(&rsBusExp.GetRow(1).PERSONAL_DATA);
   &I = 1;
   While &SQL2.Fetch(&rBusExpPer)
      &rBusExpPer.CopyFieldsTo(&rsBusExpPer(&I).BUS_EXPENSE_PER);
      &J = 1;
      &SQL3 = CreateSQL("%selectall(:1) where EMPLID = :2
and EXPENSE_PERIOD_DT = :3", &rBusExpDtl, &EMPLID,
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
&rsBusExpPer(&I).BUS_EXPENSE_PER.EXPENSE_PERIOD_DT.Value);
      &rsBusExpDtl = &rsBusExpPer.GetRow(&I).GetRowset(1);
      While &SQL3.Fetch(&rBusExpDtl)
         &rBusExpDtl.CopyFieldsTo(&rsBusExpDtl(&J).BUS_EXPENSE_DTL);
         &rsBusExpDtl.InsertRow(&J);
         &J = &J + 1;
      End-While;

      &rsBusExpPer.InsertRow(&I);
      &I = &I + 1;
   End-While;
   &MYFILE.WriteRowset(&rsBusExp);
End-While;
&MYFILE.Close();
```

The previous code generates the following output file.

```
AA8001       Schumacher,Simon
BB8001       06/11/1989YNA0        Customer Go-Live Celebration
CC8001       06/11/1989
             06/01/198908226.83             USDEntertain Clients   10100
BB8001       08/31/1989YNA0        Customer Focus Group Meeting
CC8001       08/31/198908/11/1989012401.58             USDCustomer Visit      10100⇒

CC8001       08/31/198908/12/198904250.48              USDCustomer Visit      10100⇒

CC8001       08/31/198908/12/198902498.34              USDCustomer Visit      10100⇒

BB8001       03/01/1998YYP0        Attend Asia/Pacific Conference
CC8001       03/01/199802/15/1998011200                USDConference          00001⇒

CC8001       03/01/199802/16/19980220000               JPYConference          00001⇒

BB8001       05/29/1998NNP0        Annual Subscription
CC8001       05/29/199805/29/199814125.93              USDSoftware, Inc.      10100⇒

BB8001       08/22/1998NNP0        Regional Users Group Meeting
CC8001       08/22/199808/22/19981045.69               USDDrive to Meeting    10100⇒

CC8001       08/22/199808/22/19980912.44               USDCity Parking        10100⇒

BB8001       12/12/1998NNP0        Customer Visit: Nevco
CC8001       12/12/199812/02/199801945.67              USDCustomer Feedback   00001⇒

CC8001       12/12/199812/02/19981010.54               USDTo Airport          00001⇒

CC8001       12/12/199812/03/19980610                  USDAirport Tax         00001⇒

CC8001       12/12/199812/03/199804149.58              USDCustomer Feedback   00001⇒

CC8001       12/12/199812/04/1998055.65                USDCheck Voicemail     00001⇒

CC8001       12/12/199812/04/19980988                  USDAirport Parking     00001⇒

CC8001       12/12/199812/04/199802246.95              USDCustomer Feedback   00001⇒

CC8001       12/12/199812/04/199803135.69              USDCustomer Feedback   00001⇒
```

Related Links

"Understanding File Layouts" (PeopleTools 8.59: Application Designer Developer's Guide)

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Using Standalone Rowsets to Read a File

The following code shows an example of reading in a file and inserting the rows into the database:

```
Local File &MYFILE;
Local Rowset &rsBusExp, &rsBusExpPer, &rsBusExpDtl;
Local Record &rBusExp, &rBusExpPer, &rBusExpDtl;
Local SQL &SQL1;

&rBusExp = CreateRecord(Record.PERSONAL_DATA);
&rBusExpPer = CreateRecord(Record.BUS_EXPENSE_PER);
&rBusExpDtl = CreateRecord(Record.BUS_EXPENSE_DTL);

&rsBusExp = CreateRowset(Record.PERSONAL_DATA,
CreateRowset(Record.BUS_EXPENSE_PER,
CreateRowset(Record.BUS_EXPENSE_DTL)));

&MYFILE = GetFile("c:\temp\BUS_EXP.out", "R", "UTF8", %FilePath_Absolute);
&MYFILE.SetFileLayout(FileLayout.BUS_EXP_OUT);

&SQL1 = CreateSQL("%Insert(:1)");

&rsBusExp = &MYFILE.ReadRowset();
While &rsBusExp <> Null;
   &rsBusExp.GetRow(1).PERSONAL_DATA.CopyFieldsTo(&rBusExp);
   &rsBusExpPer = &rsBusExp.GetRow(1).GetRowset(1);
   For &I = 1 To &rsBusExpPer.ActiveRowCount
      &rsBusExpPer(&I).BUS_EXPENSE_PER.CopyFieldsTo(&rBusExpPer);
      &rBusExpPer.ExecuteEdits(%Edit_Required);
      If &rBusExpPer.IsEditError Then
         For &K = 1 To &rBusExpPer.FieldCount
            &MYFIELD = &rBusExpPer.GetField(&K);
            If &MYFIELD.EditError Then
               &MSGNUM = &MYFIELD.MessageNumber;
               &MSGSET = &MYFIELD.MessageSetNumber;
            End-If;
         End-For;
      Else
         &SQL1.Execute(&rBusExpPer);
         &rsBusExpDtl = &rsBusExpPer.GetRow(&I).GetRowset(1);
         For &J = 1 To &rsBusExpDtl.ActiveRowCount
            &rsBusExpDtl(&J).BUS_EXPENSE_DTL.CopyFieldsTo(&rBusExpDtl);
            &rBusExpDtl.ExecuteEdits(%Edit_Required);
            If &rBusExpDtl.IsEditError Then
               For &K = 1 To &rBusExpDtl.FieldCount
                  &MYFIELD = &rBusExpDtl.GetField(&K);
                  If &MYFIELD.EditError Then
                     &MSGNUM = &MYFIELD.MessageNumber;
                     &MSGSET = &MYFIELD.MessageSetNumber;
                  End-If;
               End-For;
            Else
               &SQL1.Execute(&rBusExpDtl);
            End-If;
         End-For;
      End-If;
   End-For;
   &rsBusExp = &MYFILE.ReadRowset();
End-While;
&MYFILE.Close();
```

Using Errors and Warnings

For the most part, errors and warnings display messages to users informing them about invalid data. For this reason, they are almost always placed in FieldEdit or SaveEdit PeopleCode, or in SearchSave

Copyright © 1988, 2021, Oracle and/or its affiliates.

PeopleCode for validation during search processing. In conjunction with edits, errors stop processing, while warnings allow processing to continue. When errors and warnings appear in places other than FieldEdit or SaveEdit, their effects vary.

This section discusses how to:

- Use errors and warning syntax.
- Use errors and warnings in edit events.
- Use errors and warnings in RowSelect events.
- Use errors and warnings in RowDelete events.
- Use errors and warnings in other events.

Using Error and Warning Syntax

Errors and warnings require only a message that the Component Processor displays to users. You can code the message into the error or warning statement, or you can use the message catalog. Use the message catalog with the MsgGet, MsgGetExplainText, and similar functions.

Errors and warnings use the same syntax. For example:

```
Error MsgGet(11100, 180, "Message not found.");
Warning MsgGet(11100, 180, "Message not found.");
```

Using Errors and Warnings in Edit Events

You can use the following PeopleCode events for validation edits: FieldEdit and SaveEdit. The Component Processor applies FieldEdit when the user changes a field, and SaveEdit when the user saves a component. Errors and warnings in these events display a message. Most errors and warnings appear in these event types, although you can use errors and warnings elsewhere.

FieldEdit Event Errors

You can use either the record field or component record field event. The record field event for each record runs before the component record field event for that record.

An error in FieldEdit prevents the system from accepting the new value of a field. The Component Processor highlights the problem field. The user must either change the field back to its original value or to something else which does not trigger the error. A warning enables the Component Processor to accept the new data. The Component Processor does not highlight a field that has warnings.

SaveEdit Event Errors

You can use the record field or the component record event. All record field events for a record run before the component record events.

An error in SaveEdit prevents the system from saving any row of data. The Component Processor does not update the database for any field if one field has an error. Although the Component Processor displays an error message, it does not turn any field red. Unlike FieldEdit errors, SaveEdit errors can happen anywhere on a page or component, for any row of data. The data causing the error may appear on a

 Copyright © 1988, 2021, Oracle and/or its affiliates.

different page within the same group, or a row of data not currently displayed. If this is the case, the field in error is brought into view by the system.

A warning in SaveEdit also is applied to all data in the page or component, but the Component Processor will accept the data, if told to by the user. In a FieldEdit warning, the Component Processor displays a message box with the text and two buttons: OK and the standard Explain (the Explain button returns an explanation for the last message retrieved with the MsgGet function). In a SaveEdit warning, the message box contains an additional button, Cancel. OK accepts the data, overriding the warning and continuing the save process. Cancel ends the save process.

Because errors and warnings apply to all rows of data and all pages in a group, you must provide the user explicit information about what caused the error. Typically, you use the message catalog function to store messages and substitute variables into them. However, you can also facilitate this by concatenating in a field value. For example, if you have a stack of historical data on the page, you could use the following error statement:

```
Error ("The value exceeds the maximum on "|effdt|".");
```

Using Errors and Warnings in RowSelect Events

RowSelect PeopleCode filters out rows of data after the system applies search record criteria. It also can stop the Component Processor from reading additional rows of data.

Note: Errors and warnings should no longer be used in RowSelect processing; instead, use DiscardRow and StopFetching. The behavior of errors and warnings in RowSelect PeopleCode is retained for compatibility with previous releases of PeopleTools.

A warning causes the Component Processor to reject the current row, but the Component Processor continues reading more data. An error prevents more data coming into the page or component. The Component Processor accepts the row that causes the error, but does not read any more data. To reject the current row and stop loading additional rows, issue a warning and an error.

You must specify text for an error or warning, but the Component Processor does not display messages from RowSelect. You can still use the message text as a way of documenting the program.

Related Links

Understanding PeopleCode Programs and Events

"DiscardRow" (PeopleTools 8.59: PeopleCode Language Reference)

"StopFetching" (PeopleTools 8.59: PeopleCode Language Reference)

Using Errors and Warnings in RowDelete Events

When you delete a row of data, the system prompts you to confirm. If you confirm, any record field RowDelete PeopleCode runs, and any component record RowDelete PeopleCode also runs. Errors and warnings in RowDelete display a message box.

A warning from RowDelete presents two choices: accept the RowDelete (the OK button), or cancel the RowDelete (the Cancel button). An error from RowDelete PeopleCode prevents the Component Processor from removing that row of data from the page.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Using Errors and Warnings in Other Events

Do not put errors or warning in PeopleCode attached to the FieldDefault, FieldFormula, RowInit, FieldChange, RowInsert, SavePreChange, WorkFlow, and SavePostChange events. These event types activate processing that a user has no direct control over. However, the Component Processor may issue its own errors and warnings when it runs PeopleCode and encounters an unrecoverable error. The Component Processor cancels the transaction to avoid unpredictable results.

Related Links

"Warning" (PeopleTools 8.59: PeopleCode Language Reference)

"Error" (PeopleTools 8.59: PeopleCode Language Reference)

Using the RemoteCall Feature

This section provides an overview of RemoteCall components and discusses how to:

- Decide between RemoteCall and PeopleSoft Process Scheduler.
- Modify PeopleSoft Process Scheduler programs to run with RemoteCall.

Related Links

Think-Time Functions

"CallAppEngine" (PeopleTools 8.59: PeopleCode Language Reference)

Understanding RemoteCall Components

RemoteCall is a PeopleTools feature that enables executing a COBOL program remotely from within a PeopleSoft application. Remote calls are made using the RemoteCall PeopleCode function.

The purpose of the RemoteCall PeopleCode function is to allow PeopleCode programs to invoke PeopleSoft application COBOL programs.

In the application server configuration file, you can specify where the COBOL executables are located.

See "Remote Call Options" (PeopleTools 8.59: System and Server Administration).

The RemoteCall function is a synchronous call. The PeopleSoft system passes parameters to the remote program, and then waits while the program runs. When the remote program is done, it returns any results or status information to the invoking program, which then resumes execution. This means that RemoteCall is a think-time function. RemoteCall is designed for fast response time, and has an application programming interface (API) that provides programs with the response time needed for transaction processing. However, RemoteCall has no scheduling or multistep job capabilities. Each execution of RemoteCall is independent.

Note: As of PeopleTools 8, you can no longer use RemoteCall to execute an Application Engine program. Use the CallAppEngine function instead.

The RemoteCall PeopleTools feature consists of the following components:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- PeopleCode programs.

  This interface consists of the RemoteCall PeopleCode function, which is used from within PeopleCode programs to start a remote program and process results.

  Note: The PeopleCode program does not include any special code to specify where the remote program is executed.

- Remote program API.

  This is used by the remote COBOL program to receive or pass parameters and return status information.

- PeopleSoft RemoteCall service.

  The PeopleSoft application server, PSAPPSRV, advertises the RemoteCall service. The service receives requests from PeopleCode programs and starts the requested remote program. When the remote program is completed, PSAPPSRV passes the parameters and status code back to the PeopleCode program.

- Oracle Tuxedo.

  Oracle Tuxedo is a message-based transaction monitor for distributed applications. No direct Oracle Tuxedo calls need to be implemented in PeopleCode or remote programs.

PeopleCode Programs

You can execute the RemoteCall function from PeopleCode associated with any Component Processor event except SavePostChange, SavePreChange, Workflow, RowSelect, or in any PeopleCode event resulting from a ScrollSelect or related function call. However, remote programs that change data should not be run as part of a SaveEdit process, because the remote program may complete successfully even though an error occurs later in the save process.

To call a remote program that changes data, use FieldChange PeopleCode in a record field associated with a command button, or from a pop-up menu item.

Do not use RemoteCall if you expect the remote program to return a large amount of data to the client, because data is passed back only through the parameters of the PeopleCode API.

Authorization to run a remote program is like authorization to run a PeopleCode program. Because a remote program is started from PeopleCode, the user has authorization to use the page that executes the PeopleCode.

The remote program runs in a different unit of work from the page. A commit is issued by PeopleTools if needed on the client before RemoteCall is called. This means that, by default, the remote program does not know about any database changes unless the page is saved before the program is called. After the remote program starts, it runs to completion and commits or ends before returning to the page. In this way, the remote program and the page do not have locking contention. To ensure that the save has actually been done, use the DoSaveNow built-in function.

When using RemoteCall to execute a COBOL program, two types of errors can occur:

- PeopleTools errors.

Copyright © 1988, 2021, Oracle and/or its affiliates.

An error might occur in the application server domain while processing the remote call. These are treated as hard errors by PeopleCode. An error message box appears, and that PeopleCode program is terminated. In the case of a PeopleTools error, the remote program always either returns a code of zero or terminates with a message due to a system error.

- Application-specific errors.

  Any error information specific to the remote application must be passed back in regular data variables, and the application can process these in an application-specific way. If you have a status code on which the application depends, you should initialize it to an invalid value to be sure the COBOL program does return the status code.

Because the remote program is executed synchronously, users receive an hourglass icon and cannot do anything in the current window until the remote program completes. They could move to another window and do processing there, or they could open another PeopleSoft window. They cannot cancel the remote program after it starts. If the program does not terminate in a timely fashion (as determined by the RemoteCall timeout set with PeopleSoft Configuration Manager), RemoteCall attempts to terminate the process and returns an error indicating that the program was terminated.

Remote Program API

The remote program API provides the functions to get and put data between the network and the COBOL program. These functions are implemented in C, but are callable from COBOL through the PTPNETRT program. For an example, see the PTPNTEST.CBL program.

Note: If these APIs are called when the program is not running as a remote program, ACTION-GET and ACTION-PUT return an error. All other actions return without doing anything.

If an unexpected error is found, call PTPNETRT with ACTION-RESET, then with ACTION-PUT to send back any error status variables, then with ACTION-DONE to send the buffer.

PeopleSoft RemoteCall Service

The RemoteCall service serves as a bridge between the PeopleCode API and remote COBOL programs. RemoteCall is one of many services advertised by the application server domain, and can be configured as part of the standard domain setup and administration.

The client sends the RemoteCall service request, consisting of the connect information and the program name, as well as any other parameters for the program, to the application server. The RemoteCall service then executes the program and passes it the connect string.

RemoteCall Programming Guidelines

Keep the following points in mind when using RemoteCall:

- Do not use RemoteCall for long-running batch jobs.

  As a general rule, if you think execution will take more than 15 seconds, you should not be using RemoteCall, but should instead use PeopleSoft Process Scheduler.

- RemoteCall is meant for running jobs on the application server.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

It should not be used to invoke client-only programs. Support for local calling with RemoteCall is provided solely as a debugging and development aid. For client-only programs, use Declare Function, then call the external function from a library.

- If you do not want to modify an existing program, then pass only the program name and run control, and do not return any parameters.

  This way, the program requires few changes to run as a remote function.

Deciding Between RemoteCall and PeopleSoft Process Scheduler

COBOL application programs initiated by the RemoteCall service use the same COBOL application architecture used by PeopleSoft Process Scheduler. After being initiated by the dispatcher, COBOL application programs call the COBOL SQL API program, PTPSQLRT, to connect to the relational database management system to compile and execute SQL statements. You can design and implement COBOL programs to be understood by both PeopleSoft Process Scheduler and RemoteCall.

Follow these guidelines to select the optimal method for running a particular COBOL program:

- Use PeopleSoft Process Scheduler for asynchronous processes, or for processes that can be scheduled, are multistep, are long-running, or that require printed output.
- Use RemoteCall for synchronous processes that complete quickly (transaction processing types of processes).

Modifying PeopleSoft Process Scheduler Programs to Run with RemoteCall

To enable an existing program that runs under PeopleSoft Process Scheduler to run under RemoteCall as well, make the following changes:

- Include the PTCNETRT copy member.
- Include the PTCNCHEK member before the connection call to PTPSQLRT.
- Add the call to PTPNETRT ACTION-DONE just before the program terminates (after the call to disconnect from the database).

  This should be conditional on whether you are RUNNING-REMOTE-CALL.
- If you are running as a remote call, ensure that PROCESS-INSTANCE OF PRUNSTATUS is not set.

  Otherwise your calls to PTCPSTAT try to update the PSPRCSRQST table. This does not cause an error, but it is unnecessary processing.

This program can now run from PeopleSoft Process Scheduler or from RemoteCall. If a program has to pass parameters, it must have RemoteCall-specific ACTION-GET and ACTION-PUT calls.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 9

Using HTML Trees and the GenerateTree Function

Using the GenerateTree Function

This section provides an overview of HTML trees and discusses how to:

- Build HTML tree pages.
- Use HTML tree rowset records.
- Use tree actions (events).
- Initialize HTML trees.
- Process events passed from a tree to an application.
- Add mouse-over ability to HTML trees.
- Add visual selection node indicators.
- Specify override images.

Understanding HTML Trees

Use the GenerateTree function to display data in a tree format. The result of the GenerateTree function is an HTML string, which can appear in an HTML area control. The tree generated by GenerateTree is called an HTML tree.

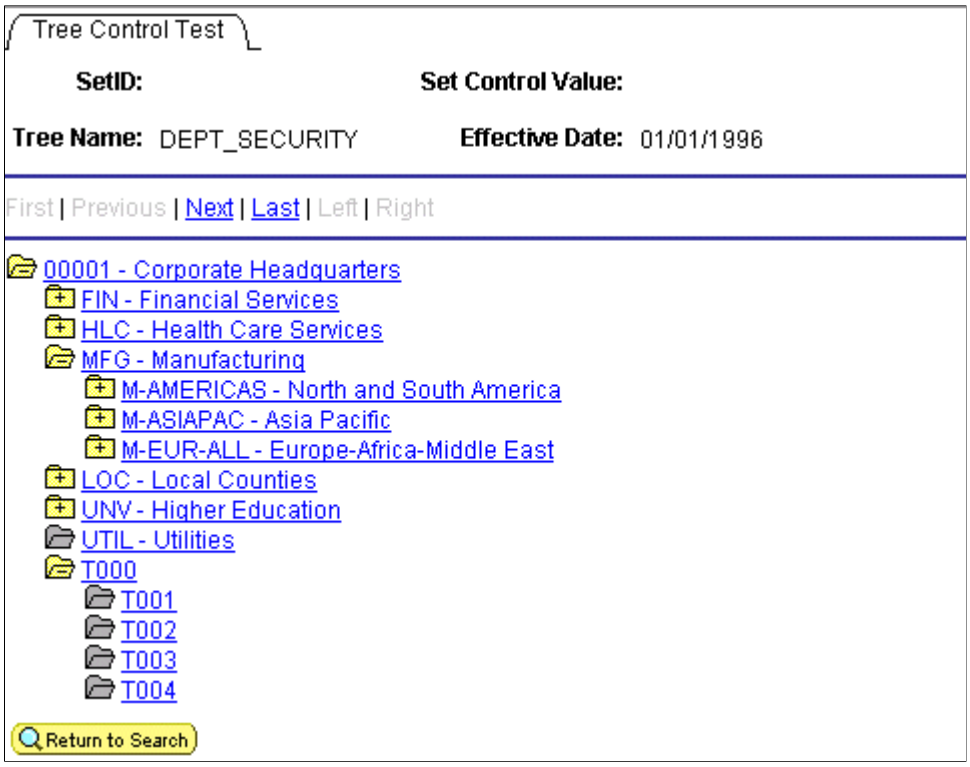
The GenerateTree function displays data from a rowset. You can populate this rowset using existing record data. You can also use the tree classes to display data from trees created using PeopleSoft Tree Manager.

To use this function, you must set up a page for displaying the data and populate a standalone rowset with the data to be displayed.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: HTML tree example

The following example shows an HTML tree. This is used to display data in tree format using GenerateTree function.



The positional links at the top of the page (First, Previous, Next, Last, Left, Right) enable the user to navigate around the tree. These links are automatically generated as part of the execution of GenerateTree.

When a node is collapsed, a plus sign appears on the node icon, and the node's children are hidden. When a node is expanded, all child nodes appear, and the icon displays a minus sign. Icons without a plus or minus sign are terminal nodes, which have no children and cannot be expanded or collapsed.

Building HTML Tree Pages

The page you use to display the HTML tree must contain:

- An HTML area used to display the HTML tree.
- A character field that has a page field name, is at least 46 characters long, and is invisible.

Note: The edit box should be invisible, but not display-only. An invisible edit box cannot be seen by the user, but it still has a buffer that can be written to. Page fields that have been specified as invisible do not need to be marked as Modifiable from HTML unless they are located on a page that is not active when GenerateTree is called. For example, if your application calls GenerateTree from one page and then saves the result in a field that is displayed by an HTML area on another page in the component, the associated event field must be marked both Invisible and Modifiable from HTML.

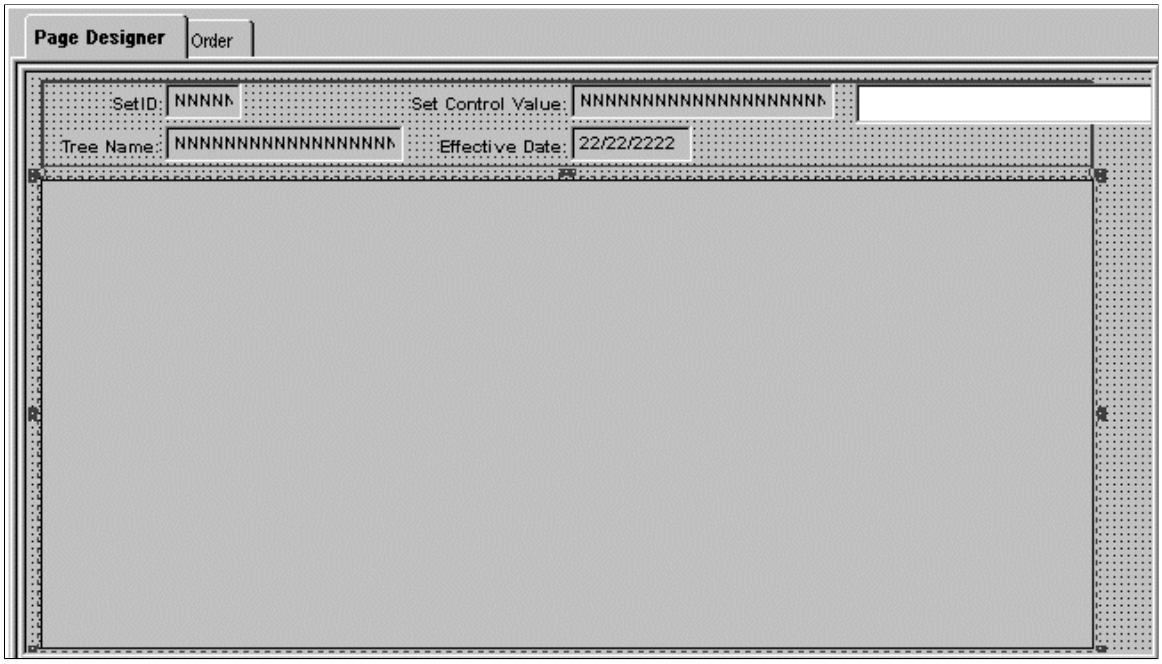
Events are sent to the application from the HTML tree using the invisible field. The events are processed by FieldChange PeopleCode that is attached to the invisible field.

This is an example page for an HTML tree:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Example of PeopleSoft Application Designer HTML tree page

This example illustrates the fields and controls on the Example of PeopleSoft Application Designer HTML tree page. You can find definitions for the fields and controls later on this page.



The large area that is selected in the example is the HTML area that displays the HTML tree. The HTML area is attached to the DERIVED_HTML.HTMLAREA field for this example.

The white edit box is the invisible field used to pass events from the HTML tree to the application. It is attached to the DERIVED_HTML.TREECTLEVENT field for this example.

The edit box must have a page field name. In this example, the page field name is TREECTLEVENT.

Using HTML Tree Rowset Records

The GenerateTree function takes a prebuilt and populated rowset as a parameter. This rowset must have a certain structure and contain certain fields. In the following examples, the rowset is standalone, that is, the rowset is created using the CreateRowset function. The fields necessary for the rowset are contained in the following record definitions:

- The header record TREECTL_HRD, containing the subrecord TREECTL_HDR_SBR.
- The node record TREECTL_NDE, containing the subrecord TREECTL_NDE_SBR.

The header record is the level zero record of the HTML tree rowset. It contains options for the HTML tree, such as the name of the collapsed node image, the height of the images, the number of pixels to indent each node, and so on.

The node record is the level one record of the HTML tree rowset. It contains the tree data and information about the data, such as the dynamic range leaf, the level, and so on.

The level one scroll area contains a row for each node or leaf in the tree data.

Copyright © 1988, 2021, Oracle and/or its affiliates.

To store additional application data with each node in the tree, you can incorporate the TREECTL_NDE_SBR into a record of your definition and use your record to define the HTML tree rowset.

For example, you might want to store application key values with each node record, so that when a user selects a node, you have the data you need to perform the action that you want.

This table describes the relevant fields in TREECTL_HDR_SBR:

Field	***Description***
PAGE_NAME	Name of the page that contains the HTML area and the invisible field used to process the HTML tree events.
PAGE_FIELD_NAME	Page field name of the invisible field used to process the HTML tree events.
PAGE_SIZE	Number of nodes or leaves to send to the browser at a time. Set to 0 to send all visible nodes or leaves to the browser. The default value is 0.
DISPLAY_LEVELS	Number of levels to display on the browser at a time. The default value is 8.
COLLAPSED_IMAGE	Collapsed node image name. The default value is PT_TREE_COLLAPSED.
EXPANDED_IMAGE	Expanded node image name. The default value is PT_TREE_EXPANDED.
END_NODE_IMAGE	End node image name. The default value is PT_TREE_END_NODE.
LEAF_IMAGE	Leaf image name. The default value is PT_TREE_LEAF.
IMAGE_WIDTH	Image width in pixels. All four images need to be the same width. The default value is 15 pixels.
IMAGE_HEIGHT	Image height in pixels. All four images need to be the same height. The default value is 12 pixels.
INDENT_PIXELS	Number of pixels to indent each level. The default value is 20 pixels.
TREECTL_VERSION	Version of the HTML tree. The default value is 812. Used with the DESCR_IMAGE field in the TREECTL_HDR_SBR record.

This table describes the relevant fields in TREECTL_NDE_SBR:

Field	***Description***
LEAF_FLAG	If this is a leaf, set to Y. The default value is N.
TREE_NODE	Node name.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Field	*Description*
DESCR	(Optional) Node description.
RANGE_FROM	The range from value of the leaf.
RANGE_TO	The range to value of the leaf.
DYNAMIC_FLAG	If this leaf has a dynamic range, set to Y. The default value is N.
ACTIVE_FLAG	Set to N for the node or leaf not to be a link. The default value is Y.
DISPLAY_OPTION	Set to N to display the name only. Set to D to display the description only. Set to B to display both the name and the description. Used for nodes only. The default value is B.
STYLECLASSNAME	Use to control the style of the link associated with the node or leaf. The default value is PSHYPERLINK.
PARENT_FLAG	If this node is a parent and its direct children are loaded now, set to Y. If this node is a parent and its direct children are loaded on demand, set to X. If this node is not a parent, set to N. The default value is N.
TREE_LEVEL_NUM	Set to the level of the node. The default value is 1.
LEVEL_OFFSET	If a child node is to appear more than one level to the right of its parent, specify the number of additional levels. The default value is 0.
DESCR_IMAGE	Use to display an image after the node or leaf image and before the name or description. The two images are separated by a space. The new image is not scaled. This field takes a string value, the name of an image definition created in PeopleSoft Application Designer. This field is only recognized if the TREECTL_VERSION field is greater than or equal to 812.
EXPANDED_FLAG	When the EXPANDED_FLAG of a node is set to Y, the GenerateTree function expects the immediate children of the node to be loaded into the &TREECTL rowset (such as in PostBuild), and GenerateTree generates HTML such that the node is expanded and its immediate children appear.

Related Links

"CreateRowset" (PeopleTools 8.59: PeopleCode Language Reference)

Using HTML Tree Actions (Events)

The GenerateTree function works with an HTML area control and an invisible field. When a user selects a node, expands a node, collapses a node, or uses one of the navigation links, that event (user action) is passed to the invisible field, and the FieldChange PeopleCode for the invisible field is executed.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The FieldChange PeopleCode example program (below) checks for expanding (or collapsing) a node and selecting a node by checking the first character in the invisible field. The following example checks for whether a node is selected:

```
If Left(TREECTLEVENT, 1) = "S" Then
```

In your application, you can check for the following user actions:

Event	*Description*
T*n*	Expand or collapse the node, whichever is the opposite of the previous state. *N* is the row number of the node in the TREECTL_NODE rowset.
X*n*	Expand the node, but load the children first. The children are loaded in PeopleCode, and then the event is passed to GenerateTree so that the HTML can be generated with the node expanded. *N* is the row number of the node in the TREECTL_NODE rowset.
F	Display the first page.
P	Display the previous page.
N	Display the next page.
L	Display the last page.
Q	Move the display left one level.
R	Move the display to the right one level.
S*n*	Select the node or leaf. *N* is the row number of the node or leaf in the TREECTL_NODE rowset.

Note: Drag-and-drop functionality is not supported in an HTML tree.

Initializing HTML Trees

For this example, the PeopleCode for initializing the HTML tree was put into the PostBuild event of the component that contained the page with the HTML area used with the HTML tree.

The PostBuild PeopleCode Example program is an example of how to initialize the HTML tree using the Tree classes and load only the root node into the HTML tree rowset.

The first time a user expands a node, the direct children of the node are loaded into the HTML tree rowset by the FieldChange PeopleCode Example program, shown in the following section. This chunking functionality enables the HTML tree to support trees of any size with good performance.

You cannot simply copy either the PostBuild or FieldChange PeopleCode example programs into your application. You must modify them to make them work with your data. You must make these changes to the PostBuild PeopleCode to initialize HTML trees:

1. Set the PAGE_NAME and PAGE_FIELD_NAME fields.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

The PAGE_NAME field contains the name of the page that contains the HTML area and the invisible field that processes HTML tree events. The PAGE_FIELD_NAME field is the page field name of the invisible field that is used to process the HTML tree events.

Note: The PAGE_FIELD_NAME field is the page field name of the invisible field, not the invisible field name.

---

2. Set tree-specific variables.

   The *&SET_ID, &USERKEYVALUE, &TREE_NAME, &TREE_DT,* and *&BRANCH_NAME* variables contain specific information about the tree. Set these values to the tree you want to open. In the example PeopleCode that follows, these variables are set as follows:

```
&SET_ID = PSTREEDEFN_VW.SETID;
&USERKEYVALUE = "";
&TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
&TREE_DT = PSTREEDEFN_VW.EFFDT;
&BRANCH_NAME = "";
```

3. Set the PAGE_SIZE field.

   If you do not want the page to expand vertically to display the tree, set the PAGE_SIZE to a number of rows that will fit inside the HTML area. If some vertical expansion is okay, but you do not want the page to get too large, set the PAGE_SIZE to whatever value you like. Set the PAGE_SIZE to 0 if you do not care how big the page gets.

4. Set the DISPLAY_LEVELS field to the number of levels that will fit inside the HTML area.

   If this field is set too large, wrapping may occur. Positional links at the top of the HTML area enable the user to navigate as the tree expands.

5. (Optional) Set the DISPLAY_OPTION field.

   The default for the DISPLAY_OPTION field is to display both the node name and the description. You can display just the node name or just the description. The values for this field are:

Field Value	***Description***
N	Display the name only.
D	Display the description only.
B	Display both the name and description.

6. (Optional) Set the STYLECLASSNAME field for the root node.

   The STYLECLASSNAME field controls the style of the link associated with a node or leaf. The default for the STYLECLASSNAME is *PSHYPERLINK*. If *PSHYPERLINK* is not the style you want to use, change this field value to the style you want.

7. Change the last line to assign the output of GenerateTree to the field attached to the HTML area that will display the tree.

Copyright © 1988, 2021, Oracle and/or its affiliates.

In the example that follows, the HTML area control is the DERIVED_HTML.HTMLAREA. You must specify the record and field name associated with the HTML area control on your page.

PostBuild PeopleCode Example

The PeopleCode for initializing the HTML tree for this example was put into the PostBuild event of the component that contained the page with the HTML area used with the HTML tree.

This example shows how to initialize the HTML tree using the tree classes and load only the root node into the HTML tree rowset:

```
Component Rowset &TREECTL;

&NODE_ROWSET = CreateRowset(Record.TREECTL_NODE);
&TREECTL = CreateRowset(Record.TREECTL_HDR, &NODE_ROWSET);

&TREECTL.InsertRow(1);
&REC = &TREECTL.GetRow(2).GetRecord(1);

/* Set the HDR options:

1) PAGE_NAME - Name of the page that contains the HTML Area
and the invisible field that will be used to process the HTML
tree events.
2) PAGE_FIELD_NAME - Page field name of the invisible field that
will be used to process the HTML tree events.
3) PAGE_SIZE - Number of nodes or leaves to send to the browser at
a time.
Set to 0 to send all of the visible nodes or leaves to the browser.
Default value: 0
4) DISPLAY_LEVELS - Number of levels to display on the browser at
a time.  Default value: 8
5) COLLAPSED_IMAGE - Collapsed node image name.
Default value: PT_TREE_COLLAPSED
6) EXPANDED_IMAGE - Expanded node image name.
Default value: PT_TREE_EXPANDED
7) END_NODE_IMAGE - End node image name.
Default value: PT_TREE_END_NODE
8) LEAF_IMAGE - Leaf image name.  Default value: PT_TREE_LEAF
9) IMAGE_WIDTH - Image width.
All four images need to be the same size.  Default value: 15
10) IMAGE_HEIGHT - Image height.  Default value: 12
11) INDENT_PIXELS - Number of pixels to indent each level.
Default value: 20
*/
&REC.GetField(Field.PAGE_NAME).Value = "TREECTL_TEST";
&REC.GetField(Field.PAGE_FIELD_NAME).Value = "TREECTLEVENT";
&REC.GetField(Field.PAGE_SIZE).Value = 15;
&REC.GetField(Field.DISPLAY_LEVELS).Value = 8;
&REC.GetField(Field.COLLAPSED_IMAGE).Value = "PT_TREE_COLLAPSED";
&REC.GetField(Field.EXPANDED_IMAGE).Value = "PT_TREE_EXPANDED";
&REC.GetField(Field.END_NODE_IMAGE).Value = "PT_TREE_END_NODE";
&REC.GetField(Field.LEAF_IMAGE).Value = "PT_TREE_LEAF";
&REC.GetField(Field.IMAGE_WIDTH).Value = 15;
&REC.GetField(Field.IMAGE_HEIGHT).Value = 12;
&REC.GetField(Field.INDENT_PIXELS).Value = 20;

&SET_ID = PSTREEDEFN_VW.SETID;
&USERKEYVALUE = "";
&TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
&TREE_DT = PSTREEDEFN_VW.EFFDT;
&BRANCH_NAME = "";

&MYSESSION = %Session;
&SRC_TREE = &MYSESSION.GetTree();
&RES = &SRC_TREE.OPEN(&SET_ID, &USERKEYVALUE, &TREE_NAME,
&TREE_DT, &BRANCH_NAME, False);
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
/* Just insert the root node into the &TREECTL Rowset.
If the root node has children, set the &PARENT_FLAG to 'X',
so that its children will be loaded on demand. */

&ROOT_NODE = &SRC_TREE.FindRoot();

If &ROOT_NODE.HasChildren Then
   &PARENT_FLAG = "X";
Else
   &PARENT_FLAG = "N";
End-If;

&NODE_ROWSET = &TREECTL.GetRow(2).GetRowset(1);
&NODE_ROWSET.InsertRow(1);
&REC = &NODE_ROWSET.GetRow(2).GetRecord(1);

/* Set the NODE values:

1) LEAF_FLAG - If this is a leaf set to "Y".  Default value: N
2) TREE_NODE - Node name.
3) DESCR - Node description.  (optional)
4) RANGE_FROM - Leaf's range from value.
5) RANGE_TO - Leaf's range to value.
6) DYNAMIC_FLAG - If this leaf has a dynamic range, set to "Y".
Default value: N
7) ACTIVE_FLAG - Set to "N" for the node or leaf not to be a link.
  Default value: Y
8) DISPLAY_OPTION - Set to "N" to display the name only.
Set to "D" to display the description only.
Set to "B" to display both the name and the description.
Only used for nodes.  Default value: B
9) STYLECLASSNAME - Used to control the style of the link
associated with the node or leaf.  Default value: PSHYPERLINK
10) PARENT_FLAG - If this node is a parent and its direct
children will be loaded now, set to "Y".  If this node is a
parent and its direct children are to be loaded on demand,
set to "X".  Default value: N
11) TREE_LEVEL_NUM - Set to the node's level.  Default value: 1
12) LEVEL_OFFSET - If a child node is to be displayed more than
one level to the right of its parent, specify the number of
additional levels.  Default value: 0
*/
&REC.GetField(Field.LEAF_FLAG).Value = "N";
&REC.GetField(Field.TREE_NODE).Value = &ROOT_NODE.NAME;
&REC.GetField(Field.DESCR).Value = &ROOT_NODE.DESCRIPTION;
&REC.GetField(Field.RANGE_FROM).Value = "";
&REC.GetField(Field.RANGE_TO).Value = "";
&REC.GetField(Field.DYNAMIC_FLAG).Value = "N";
&REC.GetField(Field.ACTIVE_FLAG).Value = "Y";
&REC.GetField(Field.DISPLAY_OPTION).Value = "B";
&REC.GetField(Field.STYLECLASSNAME).Value = "PSHYPERLINK";
&REC.GetField(Field.PARENT_FLAG).Value = &PARENT_FLAG;
&REC.GetField(Field.TREE_LEVEL_NUM).Value = 1;
&REC.GetField(Field.LEVEL_OFFSET).Value = 0;

&SRC_TREE.Close();
DERIVED_HTML.HTMLAREA = GenerateTree(&TREECTL);
```

Processing Events Passed from a Tree to an Application

To modify the FieldChange PeopleCode to load the direct children of the node into the HTML trees, use the following FieldChange PeopleCode to process the events passed from an HTML tree to an application. The code that processes the load children event loads the direct children of a node the first time the node is expanded by the user. Changes that you must make to the FieldChange PeopleCode are as follows.

Copyright © 1988, 2021, Oracle and/or its affiliates.

1. Globally change TREECTLEVENT to the name of the invisible field used to process the events.
2. Set the tree-specific variables.

   The *&SET_ID, &USERKEYVALUE, &TREE_NAME, &TREE_DT,* and *&BRANCH_NAME* variables contain specific information about the tree. Set these values to the tree you want to open. In the example PeopleCode that follows, they are set like this:

   ```
   &SET_ID = PSTREEDEFN_VW.SETID;
   &USERKEYVALUE = "";
   &TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
   &TREE_DT = PSTREEDEFN_VW.EFFDT;
   &BRANCH_NAME = "";
   ```

3. (Optional) Set the DISPLAY_OPTION field.

   The default for the DISPLAY_OPTION field is to display both the node name and the description. You can display just the node name or just the description. The values for this field are:

Field Value	***Description***
N	Display the name only.
D	Display the description only.
B	Display both the name and description.

4. (Optional) Set the STYLECLASSNAME field for the root node.

   The STYLECLASSNAME field controls the style of the link associated with a node or leaf. The default for the STYLECLASSNAME is PSHYPERLINK. If PSHYPERLINK is not the style you want to use, change this field value to the style you want.

5. Change the assignment of the output of every GenerateTree call to the field attached to the HTML area that will display the tree.

   In this example, the HTML area control is the DERIVED_HTML.HTMLAREA. You must specify the record and field name associated with the HTML area control on your page.

6. Change the code that processes the select event to perform the action you want when the user selects a node or leaf.

   This section is marked as Process Select Event in the following code sample.

FieldChange PeopleCode Example

The following is the PostBuild PeopleCode example:

```
Component Rowset &TREECTL;

/* process load children event */
If Left(TREECTLEVENT, 1) = "X" Then
   &ROW = Value(Right(TREECTLEVENT, Len(TREECTLEVENT) - 1)) + 1;
   &NODE_ROWSET = &TREECTL.GetRow(2).GetRowset(1);
   &PARENT_REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);
   &PARENT_LEVEL = &PARENT_REC.GetField(Field.TREE_LEVEL_NUM).Value;
   &ROW = &ROW + 1;

   &SET_ID = PSTREEDEFN_VW.SETID;
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
   &USERKEYVALUE = "";
   &TREE_NAME = PSTREEDEFN_VW.TREE_NAME;
   &TREE_DT = PSTREEDEFN_VW.EFFDT;
   &BRANCH_NAME = "";

   &MYSESSION = %Session;
   &SRC_TREE = &MYSESSION.GetTree();
   &RES = &SRC_TREE.OPEN(&SET_ID, &USERKEYVALUE, &TREE_NAME,
&TREE_DT, &BRANCH_NAME, False);

   /* Find the parent node and expand the tree one level below
the parent.  Insert just the direct children of the parent node
into the &TREECTL Rowset.  If any of the child nodes have
children, set their PARENT_FLAG to 'X', so that their children
are loaded on demand. */

   &PARENT_NODE = &SRC_TREE.FindNode(&PARENT_REC.
GetField(Field.TREE_NODE).Value, "");
   If &PARENT_NODE.HasChildren Then
      &PARENT_NODE.Expand(2);

      If &PARENT_NODE.HasChildLeaves Then
         /* Load the child leaves into the &TREECTL Rowset. */
         &FIRST = True;
         &CHILD_LEAF = &PARENT_NODE.FirstChildLeaf;
         While &FIRST Or
               &CHILD_LEAF.HasNextSib
            If &FIRST Then
               &FIRST = False;
            Else
               &CHILD_LEAF = &CHILD_LEAF.NextSib;
            End-If;
            If &CHILD_LEAF.Dynamic = True Then
               &RANGE_FROM = "";
               &RANGE_TO = "";
               &DYNAMIC_RANGE = "Y";
            Else
               &RANGE_FROM = &CHILD_LEAF.RangeFrom;
               &RANGE_TO = &CHILD_LEAF.RangeTo;
               &DYNAMIC_RANGE = "N";
            End-If;

            &NODE_ROWSET.InsertRow(&ROW - 1);
            &REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);

            /* Set the NODE values:

1) LEAF_FLAG - If this is a leaf set to "Y".  Default value: N
2) TREE_NODE - Node name.
3) DESCR - Node description.  (optional)
4) RANGE_FROM - Leaf's range from value.
5) RANGE_TO - Leaf's range to value.
6) DYNAMIC_FLAG - If this leaf has a dynamic range, set to "Y".
Default value: N
7) ACTIVE_FLAG - Set to "N" for the node or leaf not to be a link.
  Default value: Y
8) DISPLAY_OPTION - Set to "N" to display the name only.
Set to "D" to display the description only.
Set to "B" to display both the name and the description.
Only used for nodes.  Default value: B
9) STYLECLASSNAME - Used to control the style of the link
associated with the node or leaf.  Default value: PSHYPERLINK
10) PARENT_FLAG - If this node is a parent and its direct
children will be loaded now, set to "Y".  If this node is a
parent and its direct children are to be loaded on demand,
set to "X".  Default value: N
11) TREE_LEVEL_NUM - Set to the node's level.  Default value: 1
12) LEVEL_OFFSET - If a child node is to be displayed more than
one level to the right of its parent, specify the number of
additional levels.  Default value: 0
*/
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
            &REC.GetField(Field.LEAF_FLAG).Value = "Y";
            &REC.GetField(Field.TREE_NODE).Value = "";
            &REC.GetField(Field.DESCR).Value = "";
            &REC.GetField(Field.RANGE_FROM).Value = &RANGE_FROM;
            &REC.GetField(Field.RANGE_TO).Value = &RANGE_TO;
            &REC.GetField(Field.DYNAMIC_FLAG).Value =
&DYNAMIC_RANGE;
            &REC.GetField(Field.ACTIVE_FLAG).Value = "Y";
            &REC.GetField(Field.DISPLAY_OPTION).Value = "B";
            &REC.GetField(Field.STYLECLASSNAME).Value =
"PSHYPERLINK";
            /* Leaves never have children. */
            &REC.GetField(Field.PARENT_FLAG).Value = "N";
            &REC.GetField(Field.TREE_LEVEL_NUM).Value =
&PARENT_LEVEL + 1;
            &REC.GetField(Field.LEVEL_OFFSET).Value = 0;

            &ROW = &ROW + 1;
         End-While;
      End-If;

      If &PARENT_NODE.HasChildNodes Then
         /* Load the child nodes into the &TREECTL Rowset. */
         &FIRST = True;
         &CHILD_NODE = &PARENT_NODE.FirstChildNode;
         While &FIRST Or
               &CHILD_NODE.HasNextSib
            If &FIRST Then
               &FIRST = False;
            Else
               &CHILD_NODE = &CHILD_NODE.NextSib;
            End-If;
            If &CHILD_NODE.HasChildren Then
               &PARENT_FLAG = "X";
            Else
               &PARENT_FLAG = "N";
            End-If;

            /* If the tree uses strict levels, set the
&LEVEL_OFFSET to the number of levels that the child node is to
the right of its parent minus 1. */
            If &SRC_TREE.LevelUse = "S" Then
               &LEVEL_OFFSET = &CHILD_NODE.LevelNumber -
&PARENT_NODE.LevelNumber - 1;
            Else
               &LEVEL_OFFSET = 0;
            End-If;

            &NODE_ROWSET.InsertRow(&ROW - 1);
            &REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);
            &REC.GetField(Field.LEAF_FLAG).Value = "N";
            &REC.GetField(Field.TREE_NODE).Value = &CHILD_NODE.Name;
            &REC.GetField(Field.DESCR).Value =
&CHILD_NODE.Description;
            &REC.GetField(Field.RANGE_FROM).Value = "";
            &REC.GetField(Field.RANGE_TO).Value = "";
            &REC.GetField(Field.DYNAMIC_FLAG).Value = "N";
            &REC.GetField(Field.ACTIVE_FLAG).Value = "Y";
            &REC.GetField(Field.DISPLAY_OPTION).Value = "B";
            &REC.GetField(Field.STYLECLASSNAME).Value =
"PSHYPERLINK";
            &REC.GetField(Field.PARENT_FLAG).Value = &PARENT_FLAG;
            &REC.GetField(Field.TREE_LEVEL_NUM).Value =
&PARENT_LEVEL + 1;
            &REC.GetField(Field.LEVEL_OFFSET).Value = &LEVEL_OFFSET;

            &ROW = &ROW + 1;
         End-While;
      End-If;

      /* change the parent's PARENT_FLAG from 'X' to 'Y' */
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
      &PARENT_REC.GetField(Field.PARENT_FLAG).Value = "Y";

      HTMLAREA = GenerateTree(&TREECTL, TREECTLEVENT);
   End-If;

   &SRC_TREE.Close();
Else

   /* Process select event. */

   /* As an example, just display the selected node name or
leaf range as a MessageBox. */

   If Left(TREECTLEVENT, 1) = "S" Then
      &ROW = Value(Right(TREECTLEVENT, Len(TREECTLEVENT) - 1)) + 1;
      &NODE_ROWSET = &TREECTL.GetRow(2).GetRowset(1);
      &REC = &NODE_ROWSET.GetRow(&ROW).GetRecord(1);
      If &REC.GetField(Field.LEAF_FLAG).Value = "N" Then
         MessageBox(0, "", 0, 0, "The selected node is %1.",
&REC.GetField(Field.TREE_NODE).Value);
      Else
         If &REC.GetField(Field.DYNAMIC_FLAG).Value = "N" Then
            If &REC.GetField(Field.RANGE_FROM).Value =
&REC.GetField(Field.RANGE_TO).Value Then
               &TEMP = "[" | &REC.GetField(Field.RANGE_FROM).
Value | "]";
            Else
               &TEMP = "[" | &REC.GetField(Field.RANGE_FROM).
Value | " - " | &REC.GetField(Field.RANGE_TO).Value | "]";
            End-If;
         Else
            &TEMP = "[ ]";
         End-If;
         MessageBox(0, "", 0, 0, "The selected leaf is %1.", &TEMP);
      End-If;
   Else
      /* process all other events */
      HTMLAREA = GenerateTree(&TREECTL, TREECTLEVENT);
   End-If;
End-If;

/* done processing the event, so clear it */
TREECTLEVENT = "";
```

Related Links

Using HTML Tree Actions (Events)

Adding Mouse-Over Ability to HTML Trees

To add mouse-over ability to HTML tree elements, you must add fields to the TREECTL_HDR_SBR record and PeopleCode to the program to set the values and the images.

1. Add the following fields to the TREECTL_HDR_SBR (tree control header subrecord) record.
   - COLLAPSED_MSGNUM
   - COLLAPSED_MSGSET
   - END_NODE_MSGNUM
   - END_NODE_MSGSET
   - EXPANDED_MSGNUM

Copyright © 1988, 2021, Oracle and/or its affiliates.

- EXPANDED_MSGSET
- LEAF_NODE_MSGNUM
- LEAF_NODE_MSGSET

2. Add the following PeopleCode to set the message set and number for the mouse-over text:

```
&REC.GetField(Field.EXPANDED_MSGSET).Value = 2;
&REC.GetField(Field.EXPANDED_MSGNUM).Value = 903;
&REC.GetField(Field.COLLAPSED_MSGSET).Value = 2;
&REC.GetField(Field.COLLAPSED_MSGNUM).Value = 904;
&REC.GetField(Field.END_NODE_MSGSET).Value = 2;
&REC.GetField(Field.END_NODE_MSGNUM).Value = 905;
&REC.GetField(Field.LEAF_MSGSET).Value = 2;
&REC.GetField(Field.LEAF_MSGNUM).Value = 906;
```

3. Add the following fields to the TREECTL_NDE_SBR record:

- DESCR_MSGNUM
- DESCR_MSGSET

4. Add PeopleCode to set the DESCR_MSGNUM and DESCR_MSGSET fields.

   These two fields should be set to the correct message number and message set values that contain the text to be used as the mouse-over text.

Adding Visual Selection Node Indicators

Image: Example of selected node style

Sometimes, users need a visual indicator, such as a different color or style, to indicate which node is selected. This example shows a selected node style:

To add selected node highlighting:

1. Add the field NODESELECTEDSTYLE to the TREECTL_HDR_SBR record.
2. Add PeopleCode to set the NODESELECTEDSTYLE field to provide the highlighting effect.

   The NODESELECTEDSTYLE field takes the name of a style class.

   The following example uses the PSTREENODESELECTED style:

```
&REC.GetField(Field.NODESELECTEDSTYLE).Value = "PSTREENODESELECTED";
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

You can set the style of the selected node when processing the select event.

Note: You also must reset the style of the previous selected node when processing the select event. To find the previous selected node, you can search the node rowset looking for a node with a STYLECLASSNAME equal to the style you set for selected nodes. Alternatively, you can keep a global variable with the index of the node in the rowset. If you keep an index variable, however, you may have to update the index when processing the load children event.

Specifying Override Images

You specify different images to represent the nodes in a tree by using the TREECTL_NODE record.

To specify override images:

1. Add the following fields to the tree control node record:
   - OVERRIDE_IMAGE
   - OVERRIDE_MSGSET
   - OVERRIDE_MSGNUM
2. Add PeopleCode to use the override values when writing tree control node records.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 10

Working With File Attachments

Understanding the File Attachment Functions

This section provides an overview of:

- PeopleCode built-in file attachment functions.
- File attachment architecture.
- File attachment storage locations.
- Storage location URLs.

PeopleCode Built-in File Attachment Functions

All file attachments are performed using PeopleCode built-in functions, such as AddAttachment, ViewAttachment, GetAttachment, and so on. These functions operate on and transfer files to and from supported storage locations: database tables, FTP sites, HTTP, and Oracle Content and Experience Cloud (CEC) repositories.

PeopleCode provides built-in file attachment functions that are organized into three categories:

- End user upload/download:
  - AddAttachment

    Use the AddAttachment function to upload one file from an end user specified location (local storage or cloud storage) to a specified storage location.

    See "AddAttachment" (PeopleTools 8.59: PeopleCode Language Reference).
  - MAddAttachment

    Use the MAddAttachment function to upload one or more files from an end user specified location (local storage or cloud storage) to a specified storage location.

    See "MAddAttachment" (PeopleTools 8.59: PeopleCode Language Reference).
  - DetachAttachment

    Use the DetachAttachment function to download a file from its source storage location and save it locally on the end user machine. The file is sent to the browser with appropriate HTTP headers to cause the browser to display a save as dialog box to the user.

    See "DetachAttachment" (PeopleTools 8.59: PeopleCode Language Reference).
  - ViewAttachment

Copyright © 1988, 2021, Oracle and/or its affiliates.

Use the ViewAttachment function to download a file from its source storage location and open it locally on the end user machine.

See "ViewAttachment" (PeopleTools 8.59: PeopleCode Language Reference).

- ShareAttachment

  Use the ShareAttachment function to share files in Oracle Content and Experience Cloud (CEC).

  See "ShareAttachment" (PeopleTools 8.59: PeopleCode Language Reference).

- UnshareAttachment

  Use the UnshareAttachment function to revoke sharing access to files in Oracle Content and Experience Cloud (CEC).

  See "UnshareAttachment" (PeopleTools 8.59: PeopleCode Language Reference).

- Application server upload/download:
  - PutAttachment

    Use the PutAttachment function to upload a file from the file system of the application server to the specified storage location.

    See "PutAttachment" (PeopleTools 8.59: PeopleCode Language Reference).

  - GetAttachment

    Use the GetAttachment function to download a file from its source storage location to the file system of the application server.

    See "GetAttachment" (PeopleTools 8.59: PeopleCode Language Reference).

- Storage location maintenance:
  - CleanAttachments

    Use the CleanAttachments function to remove orphan files (files with no corresponding file reference) from specified tables used as storage locations in the current database.

    See "CleanAttachments" (PeopleTools 8.59: PeopleCode Language Reference).

  - CopyAttachments

    Use the CopyAttachments function to copy all files with file references from one storage location to another. The files to be copied can be limited to those referenced in specific file reference tables.

    See "CopyAttachments" (PeopleTools 8.59: PeopleCode Language Reference).

  - DeleteAttachment

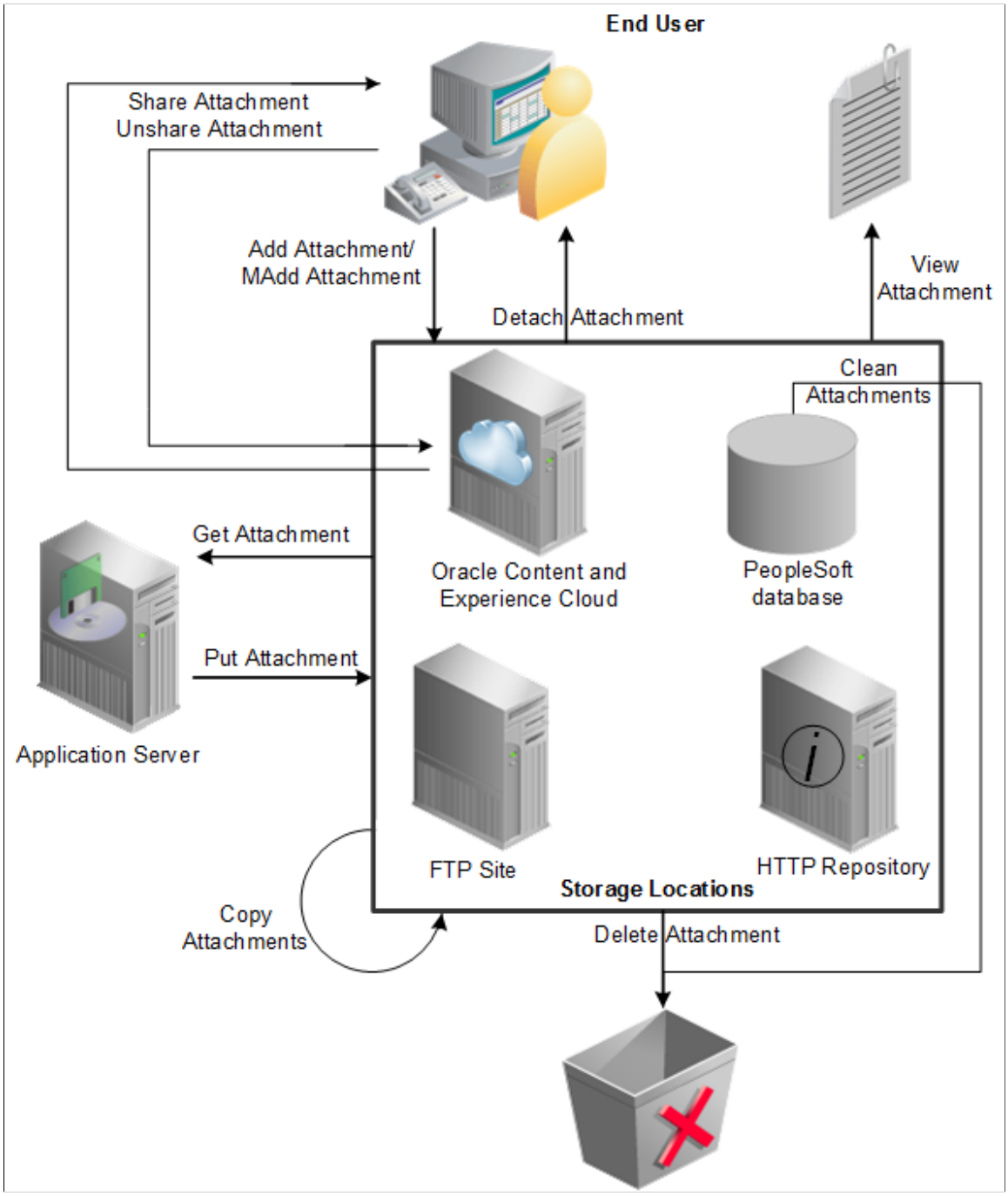
    Use the DeleteAttachment function to delete a file from the specified storage location.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

See "DeleteAttachment" (PeopleTools 8.59: PeopleCode Language Reference).

Image: PeopleCode file attachment functions

The following diagram illustrates the operation of the PeopleCode file attachment functions:



Because these functions abstract the storage of the attachments, you can use any defined storage location. The location to be used is determined by the URL passed as the first parameter to the invoked attachment function.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

Understanding File Attachment Storage Locations
Understanding URL Strings Versus URL Objects

Understanding the File Attachment Architecture

File attachments are supported by using PeopleCode built-in functions that implement the transfer of a file to or from a storage location. Using the PeopleCode functions, files can be transferred back and forth from the end user machine to the storage location (by way of the web server and application server) or transferred back and forth from the application server file system to the storage location.

The following steps depict the process of transferring a file with the AddAttachment function:

1. The file is streamed from the browser to the servlet on the web server using a standard HTML form construct.

   Optionally, if virus scan is enabled, the stream in the HTTP servlet request is scanned by the virus scan engine.

   See Setting Up Virus Scanning.

   Note: This transfer can be performed securely in an encrypted fashion if the web server uses Secure Sockets Layer (SSL) to communicate to the browser.

   Note: When the user selects a file for uploading, file size is not checked until after the file is transferred to the web server. Once the file gets to the web server the file size is compared to the value of the AddAttachment function's MaxSize parameter. The transfer is terminated if the file size exceeds this parameter.

2. As the file stream is received by the web server, the file is streamed from the web server to the application server in one-megabyte chunks until there is no streamed data available on the web server.

   Note: The one-megabyte transfer chunk size between web server and application server cannot be customized.

   Note: The web server-to-application server transfer is performed by using Oracle Jolt, which is securely encrypted. Because this transfer is done using the standard Oracle Jolt mechanism, no additional settings to the firewall are required (you do not need to open additional ports).

3. The file chunks from the web server are transferred by the application server to a temporary table in the database. The chunk size for this temporary storage depends on the ultimate storage location.

   For database tables, the chunk size is governed by the value of the Maximum Attachment Chunk Size field on the PeopleTools Options page. The chunk size in temporary storage can vary, but it never exceeds this maximum attachment chunk size limit. For all other storage locations, the chunk size in the temporary storage table is 16 KB.

4. Then, the application server transfers the file in chunks from the temporary database table to its ultimate storage location.

   Once the entire file is transferred, the application server deletes the temporary copy from the PeopleTools table in the database.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: If the storage location is a database table, then the chunk size is exactly as specified by the value of the Maximum Attachment Chunk Size field (except for the last chunk written, which can be smaller than the maximum).

See File Attachment Chunk Size.

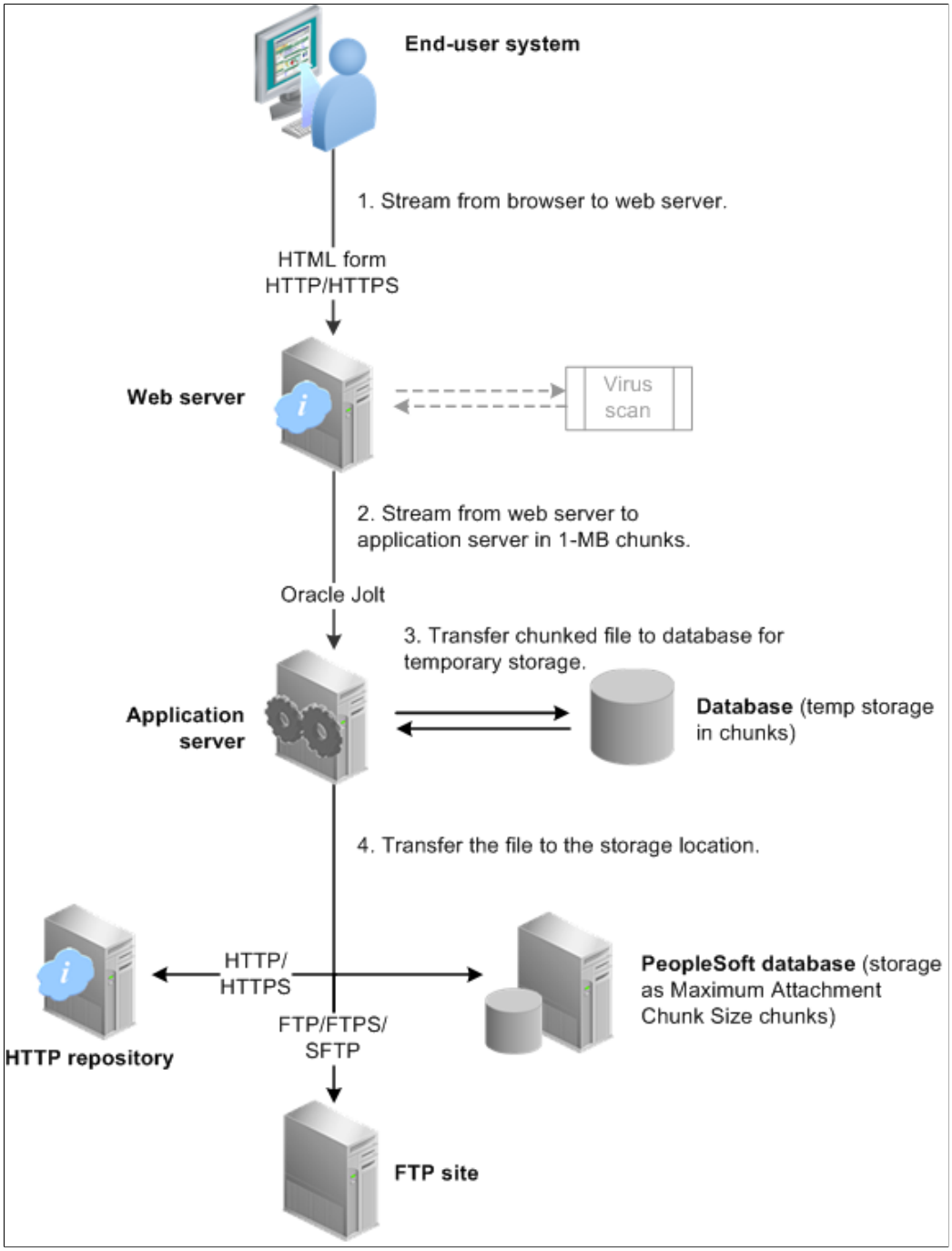
Note: If the storage location is an FTP site or an HTTP repository, the file is reassembled into a whole file at the destination.

Note: The file transfer process for MAddAttachment is, in general, similar to the process for AddAttachment. With a call to MAddAttachment, the files are streamed from the browser to the web server in bulk but from the web server through to the storage location one file at a time.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: AddAttachment file transfer process

The following diagram depicts this process of transferring a file with the AddAttachment function:



The file attachment architecture is designed for use in the frame template or the iframe template only. It is not supported in a pagelet or an HTML template. When content is rendered in a pagelet or HTML template, the user interaction is managed through the PeopleSoft portal servlet. For the file attachment architecture to work, the browser must communicate directly with the PeopleSoft content servlet, which requires the use of the frame or iframe template.

Related Links

"Understanding Portal Templates" (PeopleTools 8.59: Portal Technology)

Copyright © 1988, 2021, Oracle and/or its affiliates.

"Understanding Template Types" (PeopleTools 8.59: Portal Technology)

Understanding File Attachment Storage Locations

PeopleTools supports three types of storage locations: database tables, FTP sites, and HTTP repositories. Except for the CleanAttachments function, all PeopleCode file attachment functions support all three storage locations. The CleanAttachments function operates only on database tables as storage locations.

This section provides an overview of the following:

- Database storage considerations
- FTP site considerations
- HTTP repository considerations

Important! The following applies to FTP sites and HTTP repositories on Windows only:
When uploading to a Windows file system or downloading files from a Windows file system, remember that Windows processes file names in a *case-insensitive* manner. So, for example, if a file called FILE1.TXT already exists at a particular location on a Windows file system:
Attempting to download a file called file1.txt from that same location will have the effect of downloading the file FILE1.TXT that is there.
Attempting to upload a file called file1.txt to that same location will have the effect of overwriting the file FILE1.TXT that is there.

Database Storage Considerations

To store file attachments in the database, you must create a target table to store the attachments themselves. The record definition associated with this target table must include the FILE_ATTDET_SBR subrecord and no other fields.

This chapter contains additional information on how to create the target table.

See Application Development Process Overview.

When the storage location is a database table, the URL parameter of the invoked file attachment function can be specified in one of two ways:

- A URL string in the form of:

  record://*MYRECORD*

  In this case, *MYRECORD* is the record definition associated with the target table.

- A URL identifier in the form of:

  URL.*URL_ID*

  In this case, the URL identifier refers to the URL object named *URL_ID*.

FTP Site Considerations

When the storage location is an FTP site, the URL can be defined in one of two ways:

Copyright © 1988, 2021, Oracle and/or its affiliates.

- A URL string in the form of:

  ftp://*FTP_user*:*FTP_pwd*@*FTP_site*/*path*

  Important! FTPS and SFTP require that a URL object be used and do not support URL strings. This form of URL string is for use with the FTP protocol only.

- A URL identifier in the form of:

  URL.*URL_ID*

  In this case, the URL identifier refers to the URL object named *URL_ID*.

When specifying a URL for an FTP site, specify the FTP server's name or its IP address. Specify a path on the FTP server relative to the directory specified as the FTP server's home directory.

Note: For many FTP servers, the FTP server's home directory is specifiable on a per-FTP-user basis. Therefore, if an application will be using multiple FTP user accounts, then care must be take to ensure that files are being stored and accessed in a consistent manner with respect to their absolute location on the FTP server.

The default FTP port is 21. If you want to use a different port, you must specify it in the URL, as part of the FTP server address. For example:

```
ftp://ftpserver.example.com:6000/
```

Note: If the specified subdirectories do not exist the PeopleCode function tries to create them.

The following limitations apply to FTP URLs:

- The FTP user name to is limited to 30 characters.
- The FTP password to is limited to 16 characters.

HTTP Repository Considerations

When the storage location is an HTTP repository, the URL parameter of the invoked file attachment function must be specified as a URL identifier in the form of:

URL.*URL_ID*

In this case, the URL identifier refers to the URL object named *URL_ID*.

An HTTP repository can reside on a PeopleSoft web server, or on a different web server environment. If the HTTP repository resides on a PeopleSoft web server, then the psfiletransfer servlet has been provided to manage the file transfers to and from the storage location. If the HTTP repository resides on a non-PeopleSoft web server, then you need to ensure that the web server can handle file transfer security and requests.

Additional configuration is required to set up a PeopleSoft web server as an HTTP repository. See the “Working with HTTP/HTTPS” section in *PeopleTools: System and Server Administration*.

See "URL Maintenance" (PeopleTools 8.59: System and Server Administration)

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Understanding URL Strings Versus URL Objects

The URL parameter of the invoked file attachment function includes both the protocol to be used and the address for a storage location. These URLs can be specified as ad hoc strings at run time in certain limited cases. Alternatively, they can be defined and maintained as *URL objects,* which include an identifier, the URL itself, and additional URL properties. Oracle recommends that you always use URL objects since that approach gives you the flexibility of later changing the storage location of your files without having to modify your PeopleCode or the contents of any file reference tables used. Moreover, the FILE_EXT_LIST property of a URL object allows you to specify a file extension list, which is the most straightforward way to restrict the file types that can be uploaded to or downloaded from your PeopleSoft system. File extension lists cannot be applied to ad hoc URL strings.

See Restricting the File Types That Can Be Uploaded or Downloaded.

URL objects are created and maintained using the URL Maintenance page (PeopleTools, Administration, Utilities, Maintain URLs). The length of the full URL is limited to 254 characters. Certain protocols—specifically, FTPS, SFTP, HTTP, and HTTPS—require the use of URL objects because other information in addition to the URL itself are required. This additional information is defined as URL properties on the associated URL Properties page.

Note: For database tables and the FTP protocol only, the storage location can be specified as an ad hoc string at run time because these file transfer methods do not require additional URL properties.

PeopleTools: System and Server Administration contains detailed information on creating and maintaining URL objects.

See "URL Maintenance" (PeopleTools 8.59: System and Server Administration)

The following are examples of some valid storage location URLs:

```
record://MYAPP_ATT_CNTNT
ftp://user01:password@ftpserver.example.com/myfiles
ftps://ftp_user:usr_pwd@ftps.example.com:6000/images
sftp://usr10:pwd@ftp.example.com/attachments
http://www.example.com:8080/psfiletransfer/ps/docs
https://www.example.com:8090/psfiletransfer/empl/docs
```

Developing Applications that Use File Attachment Functions

This section provides an overview of the application development process and discusses:

- Delivered record definitions.
- Managing entries in file reference tables.
- Using the PeopleTools Test Utilities page.

Application Development Process Overview

Follow these steps to develop an application that uses file attachments:

1. Create an application-specific, default storage location.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Oracle recommends that you use a database table as the default storage location so that it is available to you during application development, and to customers as a default when the application is installed. You must include the FILE_ATTDET_SBR subrecord in the record definition associated with this target table; the record definition must have no other fields.

Create a storage location that is unique to your application. (Specifically, to avoid potential file name conflicts and unintended file overwrites, do not share storage locations among applications.) For example, create a record definition named MYAPP_ATT_CNTNT and build the associated database table. If you need to store other information, store it as part of the file reference, as described in the step 3, or create another record and use it in the component.

2. Create a URL object that corresponds to your default storage location.

   See Understanding URL Strings Versus URL Objects.

3. Create an application-specific record definition to define the table that will store file reference information and any additional information about the file attachments. You must include the FILE_ATTACH_SBR subrecord in this new record definition.

   For example, create a new record called MYAPP_ATT_REF. Add fields for any other information related to the transaction you want to store. Your application must populate the fields in this file reference table with the system file name, user file name, and any information about the file that will be needed for later use.

   Note: Create a file reference record that is specific to your application. In addition, you should consider whether to create a separate file reference record for each storage location. Doing so can prevent file name conflicts, eliminates the need to store the URL string or URL identifier with each file reference, and can ease the use of the CopyAttachments function.

   See Considerations When Using CopyAttachments.

4. Clone the FILE_ATTACH_WRK record to create an application-specific derived/work record with a unique name. Save the PeopleCode with the new record.

   For example, create a record named MYAPP_ATT_WRK by cloning FILE_ATTACH_WRK. You can use this copy of the sample PeopleCode as the basis for your own application.

   Important! The FILE_ATTACH_WRK record is delivered as a sample only. It is not intended for direct use as part of an application running in production. Instead an application-specific clone of it must be used. Oracle can change the delivered sample PeopleCode in future releases. Any application that directly uses the FILE_ATTACH_WRK record might fail. Using application-specific PeopleCode makes it easier to manage during upgrades and your PeopleCode can be reused in other components that use file attachment functionality.
   You must also implement the PeopleCode to manage the data in your file reference table (or tables).

   ---

5. Use the records you created in the previous steps to create the file attachment component and page.

   The derived/work record has fields with FieldChange PeopleCode that you can use for Add, Delete, Detach, and View buttons.

   Add PeopleCode—probably at the component record field level—to invoke the underlying functions in the application-specific derived/work record when the user clicks on one of the buttons.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Delivered Record Definitions

The following table summarizes the delivered record definitions for use in a file attachment application:

Record	*Example*	*Description*
FILE_ATTDET_SBR	MYAPP_ATT_CNTNT	Insert this subrecord in any record definition for target tables that will store attached files. Do not add other fields to this record.
FILE_ATTACH_SBR	MYAPP_ATT_REF	Insert this subrecord in any application-specific record for tables that will store references to attached files. The fields in this subrecord store the system file name and the user file name.
FILE_ATTACH_WRK	MYAPP_ATT_WRK	Clone this derived/work record to create your own application-specific derived/work record. In your application-specific derived/work record, you can modify your copy of the delivered sample code to meet your file attachment requirements and manage your file reference table (or tables).

FILE_ATTDET_SBR Subrecord

To use a database table as a storage location, you must create an application-specific record definition associated with the target table that will receive the attachments. You must include the FILE_ATTDET_SBR subrecord in your application-specific record, and it can contain no additional fields.

The FILE_ATTDET_SBR subrecord has the following fields:

Field	*Description*
ATTACHSYSFILENAME	The system file name, which must be unique to the storage location in order to avoid unintended file overwrites. Furthermore, if the file reference table to be used will contain references to file stored at more than one storage location, then the system file name must also be unique to that table. The value of the ATTACHSYSFILENAME field in the corresponding row of the file reference table must be identical to this value. ATTACHSYSFILENAME is a 128-character field.
FILE_SEQ	The file sequence number (to support file chunking).
VERSION	Version number.
FILE_SIZE	The physical size of the file chunk.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Field	Description
LASTUPDDTTM	Last update date and time.
LASTUPDOPRID	The user ID of the last user to update the attachment.
FILE_DATA	The data of the file chunk.

PeopleTools maintains the values in this table. Therefore, do not reuse the fields in this table to store incomplete or nonstandard versions of the file name or other data.

FILE_ATTACH_SBR Subrecord

You must insert the FILE_ATTACH_SBR subrecord in the application-specific record definition to be associated with the table that will store references to the attached files. The fields in this subrecord store the system file name and the user file name.

The FILE_ATTACH_SBR subrecord contains the following fields:

Field	Description
ATTACHSYSFILENAME	The system file name (the name of the file as it exists at the storage location). Among other things, this means that if the file is stored in a database table, then the value in this field must be identical to the value of the ATTACHSYSFILENAME field in the rows that correspond to the file chunks in the database table. ATTACHSYSFILENAME is a 128-character field.
ATTACHUSERFILE	The user file name (the name that the end user associates with the file). ATTACHUSERFILE is a 64-character field.

Your application must populate these fields with the system file name, user file name, and any information about the file that will be needed for later use.

See Managing Entries in File Reference Tables.

FILE_ATTACH_WRK Derived/Work Record

The FILE_ATTACH_WRK derived/work record provides sample PeopleCode programs that demonstrate how to use the file attachment PeopleCode built-in functions. Clone this derived/work record so that you can customize the programs to suit your application's needs.

The FILE_ATTACH_WRK derived/work record contains the following fields:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Field	*Description*
ATTACHADD	Contains a PeopleCode program used for uploading an attachment from an end-user machine to the specified storage location (via the AddAttachment built-in function).
ATTACHDET	Contains a PeopleCode program used for downloading an attachment from the specified storage location to be *saved* on the end-user machine (via the DetachAttachment built-in function).
ATTACHDELETE	Contains a PeopleCode program used for deleting an attachment from the specified storage location (via the DeleteAttachment built-in function).
ATTACHUTIL	Contains a user-defined PeopleCode function that can be called to determine (by file name extension) whether the attachment operation will be permitted on a file. In this function, an array of file name extensions identifies which types of files will be regarded as impermissible. **Note:** The sample PeopleCode programs included in the FILE_ATTACH_WRK derived/work record invoke this user-defined PeopleCode function.
ATTACHVIEW	Contains a PeopleCode program used for downloading an attachment from the specified storage location to be *viewed* on the end-user machine (via the ViewAttachment built-in function).

The PeopleTools Test Utilities page demonstrates a sample application that makes use of the PeopleCode programs in the FILE_ATTACH_WRK derived/work record.

See Using the PeopleTools Test Utilities Page.

Managing Entries in File Reference Tables

When you create a file attachment application, you create an application-specific record to be associated with the table that will store file reference information and any additional information about the file attachments. You must include the FILE_ATTACH_SBR subrecord in this new record. For example, you might create a new record called MYAPP_ATT_REF. Then, you would add fields for any other information related to the transaction you want to store.

Your application must populate these fields with the system file name, user file name, and any information about the file that will be needed for later use. Your application should use the fields in file reference tables as follows:

- When your application is uploading files (for example, with AddAttachment):

Copyright © 1988, 2021, Oracle and/or its affiliates.

- ATTACHSYSFILENAME – Save the system file name in the ATTACHSYSFILENAME field. This is the name of the file as it exists at the storage location and is also a key field of your file reference table.

  Note: ATTACHSYSFILENAME is a 128-character field.

- ATTACHUSERFILE – Save the user file name, which is the value returned by AddAttachment in its UserFile parameter. This is essentially the base name of file selected by the end user for uploading and would be used to allow end users to identify the file in other file attachment operations (such as viewing, downloading, or deleting).

  Note: ATTACHUSERFILE is a 64-character field.

- When your application is downloading or deleting files (for example, with ViewAttachment, DetachAttachment, or DeleteAttachment):

  - ATTACHUSERFILE – Use the ATTACHUSERFILE field to present a list of available files for end user selection. This field is also passed as a parameter to some of the built-in PeopleCode functions.

  - ATTACHSYSFILENAME – Use the ATTACHSYSFILENAME field (along with the ATTACHUSERFILE field, for some of the built-in PeopleCode functions) to construct the parameters to be passed to the built-in PeopleCode functions.

Related Links

Delivered Record Definitions

Using the PeopleTools Test Utilities Page

Access the PeopleTools Test Utilities page (PeopleTools > Utilities > Debug > PeopleTools Test Utilities).

The PeopleTools Test Utilities page contains a sample file attachment application that allows you to upload (Attach button), save (Detach button), delete (Delete button), and open (View button) a file attachment. The page allows you to specify a storage location as a URL identifier or as an ad hoc string. After clicking the Attach button, you are prompted to select a file to upload to the storage location. Once the selected file has been successfully uploaded, buttons appear that allow you to open, save, or delete that file from its storage location.

Note: This demonstration application permits the user to enter a URL of up to 120 characters only.

The actual page definition involved, PSTESTUTIL, contains buttons that execute FieldChange PeopleCode programs in the FILE_ATTACH_WRK derived/work record definition. These programs are provided as working examples of how to use the following file attachment functions: AddAttachment, DeleteAttachment, DetachAttachment, and ViewAttachment. If you are developing a file attachment application, you can clone the FILE_ATTACH_WRK derived/work record definition and modify the copied programs to fit your application’s file processing requirements.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Important! Do not modify the delivered FILE_ATTACH_WRK record definition or the PeopleCode programs it contains. In addition, do not directly call these PeopleCode programs from any PeopleCode programs you implement. Oracle might modify these sample programs in a future release of PeopleTools.

See Delivered Record Definitions.

The FILE_ATTACH_WRK derived/work record definition also demonstrates a programmatic methodology for restricting file types. The IsLegalAttachmentType function compares a given file to an internally defined array of illegal file extensions. This programmatic methodology can be contrasted with restricting file types through the use of file extension lists. File extension lists can be provide an easier, more flexible, and more manageable approach to restricting file types than a programmatic method.

Important! Do not combine these two methodologies in the same application.

See Restricting the File Types That Can Be Uploaded or Downloaded.

Application Development Considerations

This section discusses:

- File name considerations.
- Restrictions on invoking functions in certain PeopleCode events.
- Converting file names for files uploaded by PutAttachment.
- Considerations when using CopyAttachments.

File Name Considerations

If the source file name specified using one of the file attachment contains any of the following characters, the invoking function will be stopped and an error (%Attachment_Failed) is returned. The actual error message can be found in the logs.

- * (asterisk)
- : (colon)
- " (quotation mark)
- < (less than symbol)
- > (greater than symbol)
- ? (question mark)

When the file is uploaded to or downloaded from a storage location, the following characters are replaced with an underscore:

- (space)
- @ (at sign)

Copyright © 1988, 2021, Oracle and/or its affiliates.

- ; (semicolon)
- + (plus sign)
- % (percent sign)
- & (ampersand)
- ' (apostrophe)
- ! (exclamation point)
- # (pound sign)
- $ (dollar sign)

Note: In general, you should exercise caution when using an `@` or `:` character in the name of a file selected for uploading. In FTP URLs, the `:` character must to be used as a delimiter between the FTP user ID and the FTP password or just before the FTP port number (if one is specified). In addition, in FTP URLs, the `@` character must be used as a delimiter between the FTP password and the FTP server address.

Restrictions on Invoking Functions in Certain PeopleCode Events

Because AddAttachment, DetachAttachment, MAddAttachment, and ViewAttachment are interactive, they are known as “think-time” functions. This means that these functions should not be used in any of the following PeopleCode events:

- SavePreChange
- SavePostChange
- Workflow
- RowSelect
- Any PeopleCode event that initiates as a result of a Select or SelectNew method, or any of the ScrollSelect functions.

If you want to transfer files in a non-interactive mode with functions that aren't think-time functions, see GetAttachment and PutAttachment.

Related Links

"Select" (PeopleTools 8.59: PeopleCode API Reference)

"SelectNew" (PeopleTools 8.59: PeopleCode API Reference)

Think-Time Functions

Converting File Names for Files Uploaded by PutAttachment

Generally, a PeopleCode program that calls PutAttachment will also need to save (for later use) the name of each uploaded file as it ended up actually being named at the specified storage location. However, the destination file name (which may have been converted as described in “File Name Considerations”) is not passed back to the PutAttachment function. So, the only way for your PeopleCode program to ensure that

 Copyright © 1988, 2021, Oracle and/or its affiliates.

it is saving the correct name is to either avoid using special characters in the destination file name or to simulate the conversion process in something like the following example:

```
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, " ", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, ";", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "+", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "%", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "&", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "'", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "!", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "@", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "#", "_");
&ATTACHUSERFILE = Substitute(&ATTACHUSERFILE, "$", "_");
```

Note: Unlike the PutAttachment function, the AddAttachment function automatically returns the converted file name for reference and later use. For example, the file name My Resume.doc is returned through the AddAttachment function as My_Resume.doc, with the space converted to an underscore.

Related Links

File Name Considerations

Considerations When Using CopyAttachments

CopyAttachments does not modify the contents of any of the associated file reference tables. You must design your application in such a way that using CopyAttachments does not, by itself, require any subsequent changes to the contents of any of the associated file reference tables.

Application Deployment and System Configuration Considerations

This section discusses:

- File attachment functions in an environment with multiple application server domains.
- Configuring the web server to support additional MIME types.
- Restricting the file types that can be uploaded or downloaded.
- Setting up virus scanning.
- Using the HTML sanitizer.
- Considerations when attaching text files.
- File attachment chunk size.
- Using interfaces to the CopyAttachments and CleanAttachments functions.

The topics in this section are of interest primarily to customers deploying file processing applications, and secondarily to application developers.

Copyright © 1988, 2021, Oracle and/or its affiliates.

File Attachment Functions in an Environment with Multiple Application Server Domains

In an environment involving multiple application server domains, a call to one of the PeopleCode file attachment functions must *not* be passed a parameter designating a file that is located on the file system of a particular application server domain. The problem is that at the time of the call, the application server domain currently in use (as a consequence of load-balancing) might not be the application server domain that has the file in question. In this case, a file-not-found error would result. For example, this may be an issue for a call to PutAttachment, or this might cause a call to GetAttachment to result in the file being downloaded to an unexpected location (the file system of the wrong application server domain) or to fail entirely if the specified destination directory does not exist on the application server domain currently in use. Therefore, the path to the local file must be specified with this in mind by creating directories that can be comparably accessed regardless of which application server domain actually services the request at runtime.

Configuring the Web Server to Support Additional MIME Types

When a browser attempts to open a file attachment, the browser invokes a viewer based on the MIME (Multipurpose Internet Mail Extensions) type sent in the response header from the web server. For example, if the user tried to view an MP3 file, the response header sent to the browser by the web server would indicate the audio/MPEG content type:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Mon, 01 Oct 2001 21:25:51 GMT
Content-Type: audio/mpeg
Accept-Ranges: bytes
Last-Modified: Mon, 01 Oct 2001 21:00:26 GMT
ETag: "78e21918bc4ac11:cc8"
Content-Length: 60
```

Notice that the content-type is audio/mpeg. The browser uses this MIME type to determine that the viewer for audio/MPEG is the appropriate application to open this attachment. If the web server did not send this content-type header, the browser would not be able to determine the nature of the file being transmitted, and it would be unable to invoke the correct viewer application. The browser would try to display the file as text/plain, which is often the wrong behavior.

The web server maps file extensions to MIME types through entries in a web.xml configuration file. A copy of web.xml is deployed to each web server instance when it is installed. After a web server instance is created, edit its deployed copy to add any additional MIME types.

The location of the deployment copies varies depending on the web server:

Web Server	***Location of Deployment Copy***
WebLogic	*PS_HOME*/webserv/*web_server*/applications/peoplesoft/PORTAL.war/WEB-INF/web.xml

See your web server documentation for the name and location of the master copy of this configuration file.

This file contains definitions similar to the following:

```
<mime-mapping>
  <extension>
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
    doc
  </extension>
  <mime-type>
     application/msword
  </mime-type>
</mime-mapping>
<mime-mapping>
  <extension>
    xls
  </extension>
  <mime-type>
     application/vnd.ms-excel
  </mime-type>
</mime-mapping>
```

Let's say you want to add a mapping that causes .log files to be interpreted as regular text files. To determine the correct MIME type, check RFC (Request for Comments) documents 2045, 2046, 2047, 2048, and 2077, which discuss internet media types and the internet media type registry.

After checking the RFCs, you determine that the correct MIME type is text/plain. The following is an example of code you would add to the previous section of the configuration file:

```
<mime-mapping>
  <extension>
    log
  </extension>
  <mime-type>
     text/plain
  </mime-type>
</mime-mapping>
```

Once you save the file, the .log extension is associated with the content type of text/plain.

Note: You must restart your web server before these changes are recognized.

Note: When trying to view the objects, the extension must exactly match what is set up in the web.xml file. This value is case-sensitive. Therefore, if the PreserveCase parameter has been used when uploading files, it will be necessary to add a MIME type entry for each case-permutation of the file extension in question. If the object view appears garbled, chances are that either the extension is not set up in the web.xml file or there is a case mismatch.

Also see the documentation for your web server.

Restricting the File Types That Can Be Uploaded or Downloaded

You can restrict the file types that can be uploaded to or downloaded from your PeopleSoft system. The file type restrictions apply to the AddAttachment, DetachAttachment, MAddAttachment, and ViewAttachment functions. Allowable or disallowed file extensions are managed through a file extension list and through the FILE_EXT_LIST property of the URL object.

Note: File extension lists cannot be applied to ad hoc URL strings.

PeopleTools: System and Server Administration contains detailed information on creating and maintaining file extension lists.

See "File Extension List" (PeopleTools 8.59: System and Server Administration).

Copyright © 1988, 2021, Oracle and/or its affiliates.

Setting Up Virus Scanning

This section discusses:

- Enabling virus scanning.
- Configuring VirusScan.xml.
- Logging virus scans.
- Virus scan errors and return codes.

Virus scanning can be performed on all files uploaded with the AddAttachment, InsertImage, and MAddAttachment functions.

Note: If the HTML sanitizer is also configured on this web server, virus scanning is performed on the file before the HTML sanitizer is run.

Enabling Virus Scanning

To enable virus scanning, open the file VirusScan.xml and set the value of disableAll to "False". By default, disableAll is "True".

```
<Providers disableAll="False" logFile="./servers/PIA/logs/VirusScan%u.log">
```

The location of VirusScan.xml on your system for WebLogic is at this location:

```
PS_CFG_HOME\webserv\web_server\applications\peoplesoft\PORTAL.war\WEB-INF\classes\p⇒
sft\pt8\virusscan\
```

Configuring VirusScan.xml

These tags are mandatory in VirusScan.xml:

Tag	*Description*	*Example Value for Scan Engine*
<class>	Provider class of the scan engine Default provider class is: `psft.pt8.virusscan.provide⇒ r. GenericVirusScanProviderIm⇒ pl`	`psft.pt8.virusscan.provide⇒ r. GenericVirusScanProviderIm⇒ pl`
<icapversion>	ICAP version	ICAP/1.0
<service-name>	Service name for the scan engine host.	/SYMCScanResp-AV
<policycommand>	Policy command used by the Scan Engine. Only SCAN is supported.	?action=SCAN
<address>	IP address of Scan Engine host.	IP address of the machine where the scan engine is running

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Tag	Description	Example Value for Scan Engine
<port>	IP port of Scan Engine host.	Port where the scan engine is running

See *PeopleTools: MultiChannel Framework* for complete details on configuring VirusScan.xml.

See "Enabling Virus Scanning" (PeopleTools 8.59: MultiChannel Framework).

Logging Virus Scans

Detailed logging is configured in the logging.properties file for WebLogic server:

```
PS_CFG_HOME\webserv\web_server\applications\peoplesoft\logging.properties
```

Set the location of the log file in VirusScan.xml.

```
<Providers disableAll="False" logFile="./servers/PIA/logs/VirusScan%u.log">
```

The following results are logged with the date and the file name that was scanned:

- CLEAN, INFECTED, and SCANERROR

  The results for these statuses is logged in this form:

  ```
  filename = result
  ```

  For example:

  ```
  finance.xls = INFECTED
  ```

- CONNECTERROR and CONFIGERROR

  The results for these statuses is logged in this form:

  ```
  Unable to connect to the Scan engine: REASON = result
  ```

  For example:

  ```
  Unable to connect to the Scan engine: REASON = CONFIGERROR
  ```

Virus Scan Errors and Return Codes

If the file is uploaded successfully and no problems are found in the virus scan, the AddAttachment, InsertImage, and MAddAttachment function returns %Attachment_Succeeded.

If a problem is found, the PeopleCode function returns one of the following return codes:

Numeric Value	Constant Value	Description
13	%Attachment_ViolationFound	File violation detected by virus scan engine.
14	%Attachment_VirusScanError	Virus scan engine error.
15	%Attachment_VirusConfigError	Virus scan engine configuration error.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Numeric Value	Constant Value	Description
16	%Attachment_VirusConnectError	Virus scan engine connection error.

Using the HTML Sanitizer

Setting Up the HTML Sanitizer

The HTML sanitizer is used to scan and sanitize HTML files uploaded with the AddAttachment and MAddAttachment functions. To use the HTML sanitizer, you configure a allowlist file to specify the files to be scanned and sanitized by file extension along with the specific HTML tags to be allowed.

Note: If virus scanning is also configured on this web server, virus scanning is performed on the file before the HTML sanitizer is run.

To enable and configure the HTML sanitizer on your web server:

1. Locate the HTMLAllowlist.xml file for WebLogic server:

   ```
   PS_CFG_HOME\webserv\web_server\applications\peoplesoft\PORTAL.war\WEB-INF\clas⇒
   ses\psft\pt8\htmlsanitizer\
   ```

2. Edit the HTMLAllowlist.xml file to enable the HTML sanitizer:

   ```
   <Sanitizer enable="True">
   ```

3. Explicitly specify each file extension that you want to be scanned and sanitized. In the following example, only files with an extension of html will be scanned and sanitized. All other files will not pass through the HTML sanitizer:

   ```
       <AppliedFileExtentions>
           <Extention>html</Extention>
   <!-- <Extention>htm</Extention>
           <Extention>shtml</Extention>
           <Extention>xhtml</Extention>
           <Extention>hta</Extention> -->
       </AppliedFileExtentions>
   ```

   Note: File extension specifications are case sensitive; HTML and hTML are not equivalent to html.

4. Identify the tags that you want to be allowed (written) as tags themselves in the HTML output.

   - Identify the tags for container elements— for example, <html>, <head>, and <body>.
   - Identify the tags for content elements—for example, <p>, headings (<h1>, <h2>, and so on), lists (<ol>, <ul>, <li>), and other content elements.
   - Identify the tags for additional content attributes—for example, <id>, <src>, <height>, and so on.
   - If a tag is not specified in the allowlist file, then the tag itself will be stripped from the HTML output. However, the content of the tag will still be allowed in the output.
   - Conversely, to strip the content of the tag (but not the tag itself), use allowtext="False".

 Copyright © 1988, 2021, Oracle and/or its affiliates.

5. Explicitly specify each tag for the container elements as well as whether to allow untagged text content within those container tags. The following example allows the <html>, <head>, and <body> tags along with untagged text:

```
<Element name="html" allowtext="True">
</Element>

<Element name="head" allowtext="True">
</Element>

<Element name="body" allowtext="True">
</Element>
```

Note: Allowing untagged text within a container element will ensure that line breaks are not stripped from files that pass through the HTML sanitizer.

6. Explicitly specify each tag for content elements and their attributes. As an example, the following entries allow <title>, <p>, and <b> tags. However, the text within the <b> tag is disallowed and stripped from the HTML output:

```
<Element name="p" allowtext="True">
</Element>

<Element name="b" allowtext="False">
</Element>

<Element name="title" allowtext="True">
</Element>
```

In this example, the src attribute is allowed for the <img> tag:

```
<Element name="img" allowtext="False">
     <Attribute name="src">
     </Attribute>
 </Element>
```

Testing the HTML Sanitizer

You can use the PeopleTools Test Utilities page to test the settings you've made in the HTMLAllowlist.xml file. To test the HTML sanitizer:

1. Select PeopleTools >Utilities >Debug >PeopleTools Test Utilities.
2. In the URL Id or String URL field, enter the following URL ID: *URL.FILEDB*
3. Click the Attach button and browse to select an HTML file to upload for the test.
4. Click the View button to view the sanitized file in a new browser tab. Click the Detach button to download the sanitized file.

If the file is uploaded successfully and no problems are encountered by the HTML sanitizer, the AddAttachment or MAddAttachment function returns %Attachment_Succeeded. If a problem is encountered, the PeopleCode function returns one the following return codes (which are the same codes as used by the virus scan engine):

Numeric Value	***Constant Value***	***Description***
13	%Attachment_ViolationFound	File violation detected by HTML sanitizer.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Numeric Value	*Constant Value*	*Description*
14	%Attachment_VirusScanError	HTML sanitizer error.
15	%Attachment_VirusConfigError	HTML sanitizer configuration error.
16	%Attachment_VirusConnectError	HTML sanitizer connection error.

Example

The following example presents:

- An example of a configured HTMLAllowlist.xml file.
- A sample HTML input file.
- The HTML output of the HTML sanitizer.

Example HTMLAllowlist.xml

The following code provides an example of a simple allowlist file to demonstrate the inclusion and exclusion of tags as well as the effects of the allowtext setting:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--HTML white listing configuration file -->
<Sanitizer enable="True">
    <AppliedFileExtentions>
        <Extention>html</Extention>
<!--    <Extention>htm</Extention>
        <Extention>shtml</Extention>
        <Extention>xhtml</Extention>
        <Extention>hta</Extention> -->
    </AppliedFileExtentions>

    <Element name="html" allowtext="True">
    </Element>

    <Element name="head" allowtext="True">
        <Element name="title" allowtext="True">
        </Element>
    </Element>

    <Element name="body" allowtext="False">

        <Element name="p" allowtext="True">
        </Element>

        <Element name="b" allowtext="False">
        </Element>

        <Element name="style" allowtext="True">
        </Element>

<!--        <Element name="img" allowtext="False">
            <Attribute name="src">
            </Attribute>
        </Element>

        <Element name="a" allowtext="True">
           <Attribute name="href">
                <AllowURL>True</AllowURL>
                <RegularExp>(?:ht|f)tps?:.*</RegularExp>
           </Attribute>
```


Copyright © 1988, 2021, Oracle and/or its affiliates.

```
        </Element>

        <Element name="h1" allowtext="True">
        </Element>
    </Element> -->

    </Element>

</Sanitizer>
```

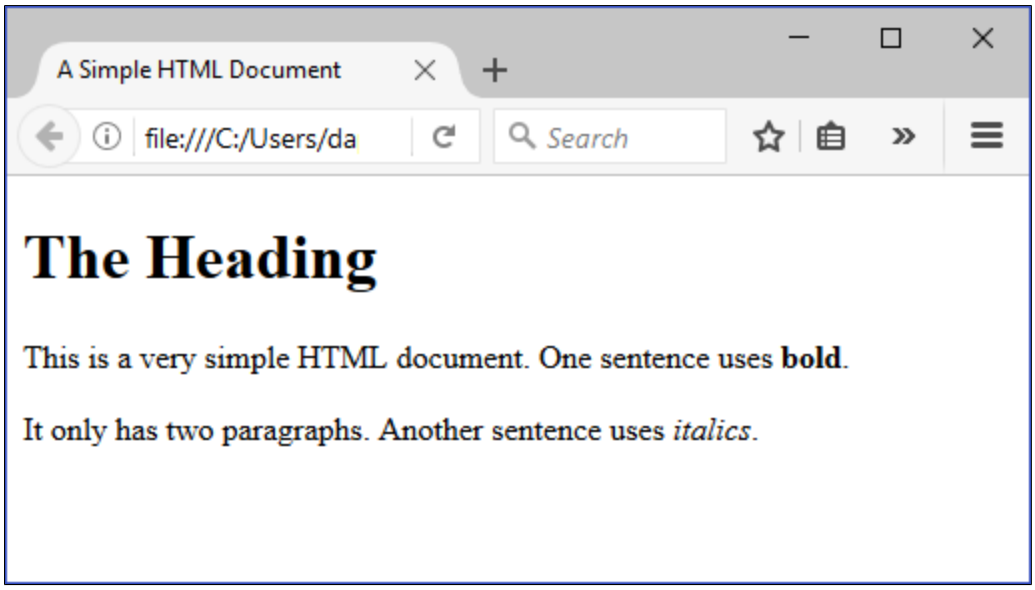
Sample HTML File

The following code presents the contents of a simple HTML file. The line numbers are included for reference only and are not part of the file:

```
[01]  <!DOCTYPE html>
[02]  <html>
[03]  <head>
[04]  <title>
[05]  A Simple HTML Document
[06]  </title>
[07]  </head>
[08]  <body>
[09]  <h1>The Heading</h1>
[10]  <p>This is a very simple HTML document. One sentence uses <b>bold</b>.</p>
[11]  <p>It only has two paragraphs. Another sentence uses <i>italics</i>.</p>
[12]  </body>
[13]  </html>
```

Image: Example file: HTML output

The browser displays this example HTML file as follows:



Example of Sanitized HTML Output

The following code presents the sanitized HTML output of the simple HTML file. Similarly, the line numbers are included for reference only and are not part of the file:

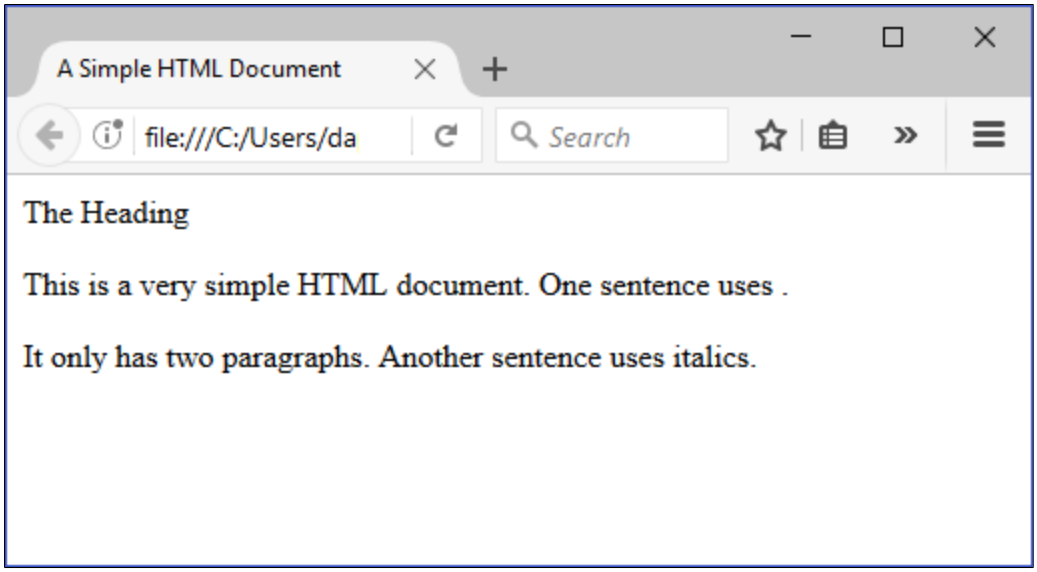
```
[01]
[02]  <html>
[03]  <head>
[04]  <title>
[05]  A Simple HTML Document
[06]  </title>
[07]  </head>
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
[08]  <body>The Heading<p>This is a very simple HTML document. One sentence uses <b⇒
></b>.</p><p>It only has two paragraphs. Another sentence uses italics.</p></body>
[09]  </html>
```

Image: Example file: Sanitized HTML output

The browser displays this sanitized HTML output file as follows:



Considerations When Attaching Text Files

The PeopleCode file attachment functions do not provide text file conversions when files are attached or viewed. In fact, when any file is uploaded, it is always copied to the specified destination byte-for-byte.

Warning! You may encounter problems when a text file is uploaded from one operating system or environment and then later viewed on another. For instance, suppose a text file on a DB2 system is encoded in EBCDIC. A user viewing that file in a Windows environment might see garbled content because the text file viewer is expecting ANSI encoding.
Similar issues can occur when two file systems have different character sets, such as Japanese JIS and Unicode, or different line endings.

It is the developer's responsibility to manage this issue in their environments. A number of text file conversion utilities are available for various platforms.

Some steps you can take to avoid conversion problems include:

- Educate your users.
- Standardize on file formats and encodings.
- Make sure that the user's environment supports the files being transferred.
- Restrict attachments to file types that are known to be compatible across user platforms.

Copyright © 1988, 2021, Oracle and/or its affiliates.

File Attachment Chunk Size

When using a database table as the storage location, the file is automatically “chunked,” or stored, in multiple rows of the database table. The size of each chunk is determined by the Maximum Attachment Chunk Size field on the PeopleTools Options page.

Because each file is chunked, you cannot pull whole files directly from the database. You must use the PeopleCode file attachment functions, which automatically put the data back together into one file for you. Because the chunk size is stored with the file, if you change the system chunk size, you can still retrieve files with different chunk sizes.

Related Links

"PeopleTools Options" (PeopleTools 8.59: System and Server Administration)

Using Interfaces to the CopyAttachments and CleanAttachments Functions

PeopleTools provides three interfaces to executing the CopyAttachments and CleanAttachments functions (select PeopleTools, Utilities, Administration, Administer File Processing):

- Copy File Attachments (Batch) page.

  Use the Copy File Attachments (Batch) page to execute the CopyAttachments function asynchronously via the COPYATTS Application Engine program. This ensures that a large CopyAttachments job does not terminate prematurely due to a timeout. See "Copy File Attachments (Batch)" (PeopleTools 8.59: System and Server Administration) for more information.

- Delete Orphan Files (Batch) page.

  Use the Delete Orphan Files (Batch) page to execute the CleanAttachments function asynchronously via the CLEANATT84 Application Engine program. This ensures that a large CleanAttachments job does not terminate prematurely due to a timeout. See "Delete Orphan Files (Batch)" (PeopleTools 8.59: System and Server Administration) for more information.

- Manage Attachment Repositories page.

  Use the Manage Attachment Repositories page to execute either CopyAttachments or CleanAttachments directly in a synchronous manner. See "Manage Attachment Repositories" (PeopleTools 8.59: System and Server Administration) for more information.

  Note: Executing either CopyAttachments or CleanAttachments directly in a synchronous manner may result in a timeout if the number of file attachments to be processed is large. Therefore, Oracle recommends that you use the batch processing interfaces instead.

Note: None of these interfaces allow you to set all of the optional parameters of either CopyAttachments or CleanAttachments. If you want to enable the use of these optional parameters, you must clone and then modify the existing PeopleCode or implement your own PeopleCode to do this.

Related Links

"CleanAttachments" (PeopleTools 8.59: PeopleCode Language Reference)

"CopyAttachments" (PeopleTools 8.59: PeopleCode Language Reference)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Debugging File Attachment Problems

This debugging section discusses the following:

- Enabling tracing on the web server or application server.
- Problems with transfers to and from FTP sites.
- Attachments with non-ASCII file names.
- Problems uploading files.
- Passing error messages to the end user.

The topics in this section are of interest primarily to customers deploying file processing applications, and secondarily to application developers.

Enabling Tracing on the Web Server or Application Server

This section discusses how to:

- Enable tracing on the web server.
- Enable PeopleCode tracing on the application server.

Enabling Tracing on the Web Server

To enable web server tracing of file attachment processes:

1. Select PeopleTools, Web Profile, Web Profile Configuration, and open the current web profile.
2. Select the Custom Properties page.
3. Add a new row, and enter these values:

Column	***Value***
Property Name	*IDDA*
Validation Type	*Number*
Property Value	*32* (File processing)

4. Set the .level property of the logging.properties file to *ALL*.
5. Restart the web server.

The log files are written to a directory that depends on the java.util.logging.FileHandler.pattern property of the logging.properties file.

More information on IDDA logging is available in the PeopleTools PeopleBooks.

See "Enabling IDDA Logging" (PeopleTools 8.59: System and Server Administration).

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Enabling PeopleCode Tracing on the Application Server

For tracing file attachment issues, set the PeopleCode trace level to 2048 (Statement Tracing, which shows each statement as it's executed). In addition, higher PeopleCode trace settings are recommended *whenever* CopyAttachments is run. You can enable PeopleCode tracing on the application server in several ways:

- For all client sessions by setting TracePC in Configuration Manager.
- For a specific client session through the Trace PeopleCode page (select PeopleTools, Utilities, Debug, Set PeopleCode Trace Options.

Because PeopleCode tracing can generate a lot of output, setting tracing for a specific client session only is recommended.

Application server log files can be found in the *PS_CFG_HOME*/appserv/*domain*/LOGS directory.

- The application server log files have names in the form APPSRV_*MMDD*.LOG (in which *MMDD* represents the month and date).
- The file transfer log file has a name in the form of FILETRANSFER*pid*.LOG.
- The PeopleCode trace file has a name of the form, *.tracesql.

See "Specifying Trace Settings" (PeopleTools 8.59: System and Server Administration).

Problems with Transfers to and from FTP Sites

A common reason that a transfer fails is that the FTP server is not accessible from the application server. This error could be due to:

- An incorrect password.
- An incorrect account name.
- An inability of the application server to resolve the FTP server's host name.
- The FTP server is down.

Try to ping the FTP server machine from the application server system, and then try to manually transfer a file to the FTP server machine from the application server.

If the FTP site is on Microsoft Windows, the host name for the system might not be associated with a fixed IP address and might not be resolvable using DNS (Domain Name System). If the application server is on a UNIX machine, the application server can resolve the host name using DNS only—or perhaps using NIS (Network Information System) or an /etc/hosts file. However, the application server will be unable to use Windows mechanisms such as WinBeui or WINS. Therefore, the application server will not be able to convert the host name indicated for the Microsoft Windows file server into an IP address and route to it.

If the file transfer fails, you must resolve the problem by either specifying the numeric IP address in the FTP URL or by putting the host name for the FTP site into DNS, NIS, or the hosts file on your application server so that the name can be resolved.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Typically, the URL used for file attachments has the following format:

```
ftp://user:pwd@system_name/dir1/subdir
```

However, if the domain name cannot be resolved with DNS, then use the numeric IP address. The following example assumes *system_name* has the IP address of 192.0.2.1:

```
ftp://user:pwd@192.0.2.1/dir1/subdir
```

Note: Use numeric IP addresses only when absolutely necessary.

Attachments with non-ASCII File Names

To successfully upload an attachment with a file name containing non-ASCII characters (such as Japanese), Oracle recommends running the application server with the locale that supports those specific non-ASCII characters—for example, ja_JP.sjis. If the storage location for the attachment is an FTP site or an HTTP repository, Oracle recommends that the storage location also be running in an environment that supports the same language or locale as the file names used. The web server (which serves as an intermediary in the transfer of the file from the browser to the application server and then on to the storage location) can be running either a multi-language environment (for example, UTF-8) or a non-ASCII character language environment.

The uploading of a file attachment involves multiple tiers—for example, the browser on the client, the web server, the application server, the database server, and ultimately the file attachment storage location. To ensure the correct transfer of data and files between these tiers, Oracle recommends configuring each server tier (web server, application server, database server, and file storage location) to use the same character set as follows:

- If your PeopleSoft system operates in a multi-language environment, use a UTF-8 character set on each server tier.
- If your PeopleSoft system operates in a single language environment, use the native language character set for that language on each server tier.

The Global Technology PeopleBook provides examples of which character set to specify on which tier for three typical configurations—a multi-language environment, a single language environment (Western), and a single language environment (non-Western).

Related Links

"Understanding Character Set Selection" (PeopleTools 8.59: Global Technology)

Problems Uploading Files

You cannot use a relative path to specify the file that is to be uploaded; you must use a full path. If users experiences problems in uploading files, ensure that they browse to the file they wish to upload rather than attempting to manually enter the full path name of the file.

This problem can manifest itself differently depending on the browser used. For example, with some browser versions, the PeopleSoft page appears to be in an infinite “Processing” state.

Related Links

PeopleTools Browser Compatibility Guide (Oracle Support Document 704492.1)

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Problems Downloading Files

The combination of the attachmentExtraWindow web profile property and some of Internet Explorer's settings can silently interfere with downloading and opening of files with the ViewAttachment function. For more information, see the documentation on the attachmentExtraWindow web profile property.

Related Links

"Configuring Custom Properties" (PeopleTools 8.59: Portal Technology)

Passing Error Messages to the End User

When working with the attachment functions, if you want the end user to be able to view error messages (such as that the file is too large, that the file was not found, that there is no disk space at the storage location, and so on), then you need to write or clone PeopleCode to interpret function return codes and pass error messages back to the user.

As an example, each of the programs in the FILE_ATTACH_WRK derived/work record includes a parameter that sets the message level, but does not translate these into user terminology. The message levels that can be set are:

- 0 – Suppress all messages including errors.
- 1 – Display all messages.
- 2 – Suppress success messages only, but display error messages.

By default, the message level is 0 for each of these programs. The programs are demonstrated on the PeopleTools Test Utilities page.

Related Links

Delivered Record Definitions
Using the PeopleTools Test Utilities Page

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 11

Accessing PeopleCode and Events

Understanding PeopleCode Programs and Events

Every PeopleCode program is associated with an aspect of a Application Designer definition and an event. Events are predefined points either in the Component Processor flow or in the program flow. As each event is encountered, it fires on each component, triggering any PeopleCode program associated with that component and that event. Each definition in Application Designer can have an *event set,* that is, a group of events appropriate to that definition. A definition can have zero or one PeopleCode programs for each event in its event set.

Some definitions have events that fall outside the Component Processor flow. These definitions include Application Engine programs, component interfaces, and application packages. In addition, security has a signon event, which is described in the documentation for the definition or topic.

Related Links

"Understanding Component Interface Class" (PeopleTools 8.59: PeopleCode API Reference)

"Understanding Application Classes" (PeopleTools 8.59: PeopleCode API Reference)

"Adding Message Definitions" (PeopleTools 8.59: Integration Broker)

"Specifying PeopleCode Actions" (PeopleTools 8.59: Application Engine)

"Security Administration Overview" (PeopleTools 8.59: Security Administration)

Understanding Automatic Backup of PeopleCode

A PeopleCode program is automatically saved to a file while you are working on it. This checkpoint occurs at the following times:

- Every 10 keystrokes.
- On a save command, just before the save is executed (in case the save does not actually execute because the code is invalid).
- When another PeopleCode program is selected to be edited (if you have two PeopleCode editor windows open at the same time and you move from one to the other).

The file is saved to your temp directory, as specified in your environment, in a file with the following name:

```
PPCMMDDYY_HHMMSS.txt
```

, where MMDDYY represents the month, date, and year of the checkpoint, respectively, and HHMMSS represents the hour, minute, and second of the checkpoint, respectively.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The top of the checkpoint file contains the following information:

```
[PeopleCode Checkpoint File]

[RECORD.recordnameFIELD.fieldnameMETHOD.eventname]
```

If your PeopleCode program saves successfully, checkpoint files associated with that program are automatically deleted.

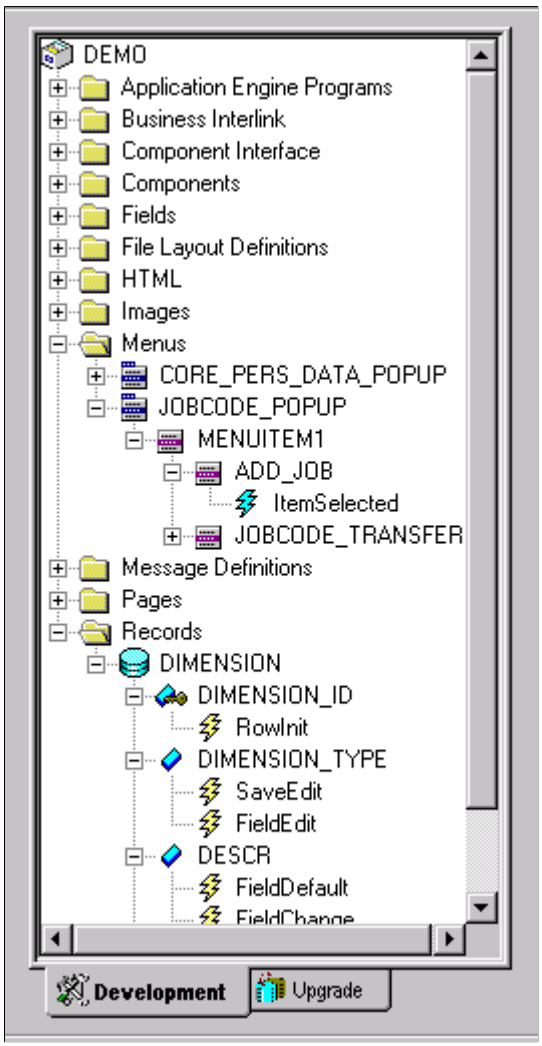
Accessing PeopleCode in Application Designer

You can access PeopleCode associated with Application Designer definitions in several ways.

For record fields and pop-up menu items, the Project view displays PeopleCode programs within the project hierarchy using a lightning bolt icon. The programs are children of the fields and pop-up menu items with which they are associated, and they are named according to their associated events, such as ItemSelected, RowInit, or SaveEdit, as shown in the following example.

Image: Example of PeopleCode programs in the Project view hierarchy

The following image is an example of PeopleCode programs in the Project view hierarchy.

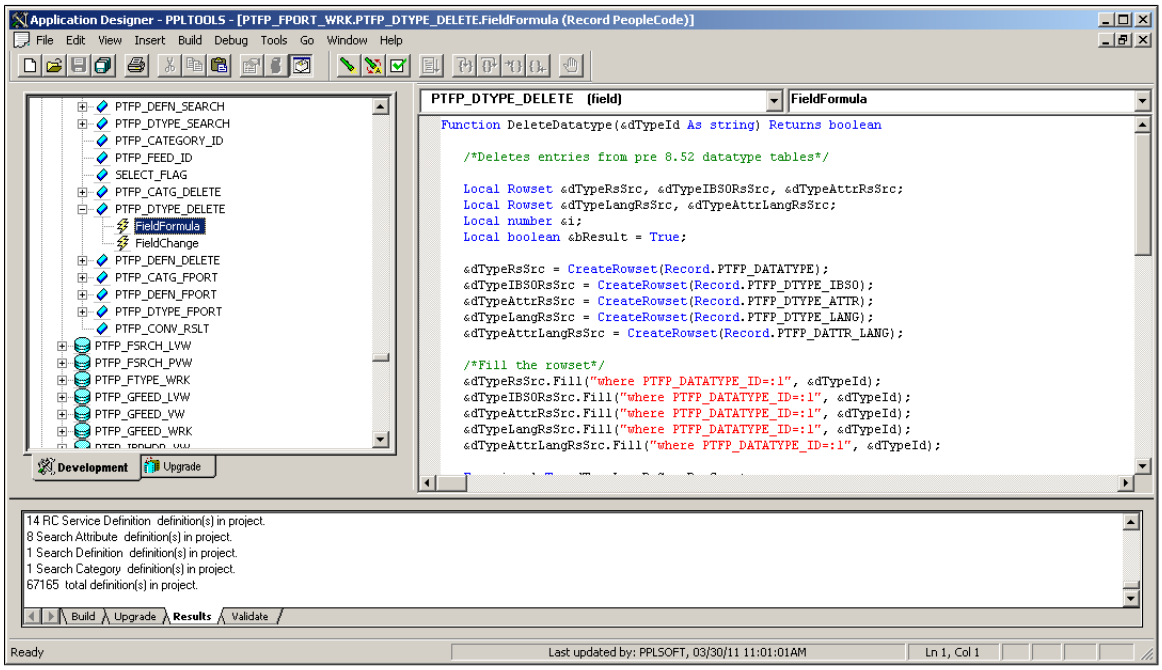


 Copyright © 1988, 2021, Oracle and/or its affiliates.

Double-click a record field or pop-up menu item program in the Project view to start the PeopleCode Editor and load that program for editing. When you load a program in the PeopleCode Editor, the status bar at the bottom of the Application Designer window displays the date, time, and the ID of the user who last updated the program as shown in the following example:

Image: Status bar displaying the last update information for a PeopleCode program

This example illustrates the fields and controls on the Status bar displaying the last update information for a PeopleCode program. You can find definitions for the fields and controls later on this page.



You can associate PeopleCode with other types of definitions, such as:

- Components
- Pages
- Component interfaces

Such PeopleCode programs do not appear in the Project view. Instead, you right-click the name of the definition and select View PeopleCode. You can also access these programs from their associated definitions.

PeopleCode can also be associated with:

- Component records (specific records included in components).
- Component record fields (specific record fields included in components).
- Application packages.

Because component record fields and component records do not appear in the Project view, you must access their associated programs through their parent definitions.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

Accessing Record Field PeopleCode
Accessing Component PeopleCode

Accessing Record Field PeopleCode

This section provides an overview of the record field event set and discusses how to access record field PeopleCode.

Understanding Record Field PeopleCode

A record is a table-level definition. Record definitions are of different types, such as SQL table, dynamic view, derived/work, and so on.

Record fields are child definitions of records. Record field PeopleCode programs are child definitions of record fields. The following events are associated with a record field:

- FieldChange
- FieldDefault
- FieldEdit
- FieldFormula
- RowInit
- RowInsert
- RowSelect
- RowDelete
- PrePopup
- SaveEdit
- SavePreChange
- Workflow
- SavePostChange
- SearchInit
- SearchSave

Related Links

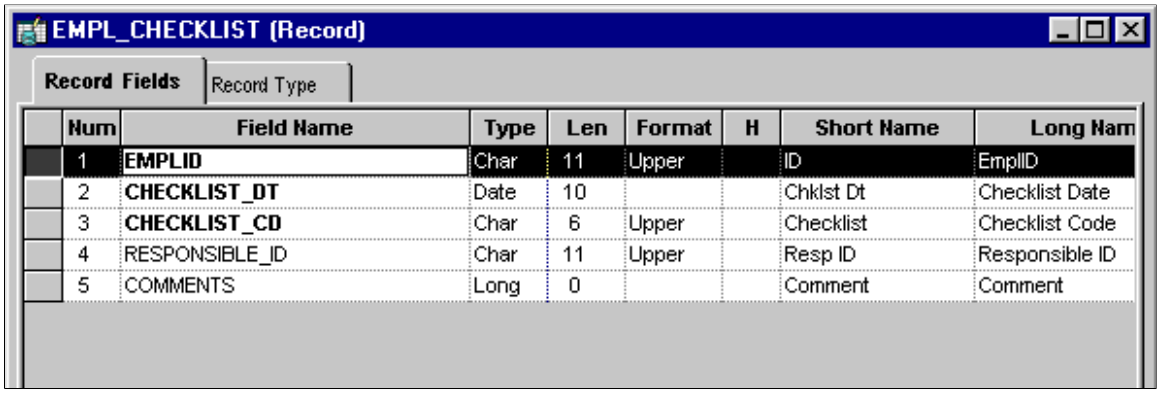
PeopleCode Program Triggers

Copyright © 1988, 2021, Oracle and/or its affiliates.

Accessing Record Field PeopleCode from a Record Definition

Image: Record definition showing three fields associated with PeopleCode

This image is an example of Record definition. This shows three fields associated with PeopleCode which appear in bold type in record views.



In the previous example, the first three fields (in boldface font) have PeopleCode associated with them. If you expand the subrecords in a record definition, any fields in the subrecord that have PeopleCode associated with them also appear in bold type.

To access record field PeopleCode from an open record definition:

1. Click the PeopleCode Display button on the toolbar.

   A grid appears with a column for each event in the record field event set. Each cell represents a field-event combination. The column names are abbreviations of the record field event names, for example, *FCh* for the FieldChange event and *RIn* for the RowInit event. A check mark appears in the appropriate cell for each field/event combination that has an associated PeopleCode program.

2. Access the PeopleCode using one of these methods:

   - Double-click the cell.
   - Right-click the cell and select View PeopleCode.
   - Select View, PeopleCode.

The PeopleCode Editor appears. If the field/event combination has an associated program, it appears in the editor.

Related Links

Understanding Record Field PeopleCode
Using the PeopleCode Editor

Accessing Record Field PeopleCode from a Page Definition

You can associate a PeopleCode program with any page control that you can associate with a record field.

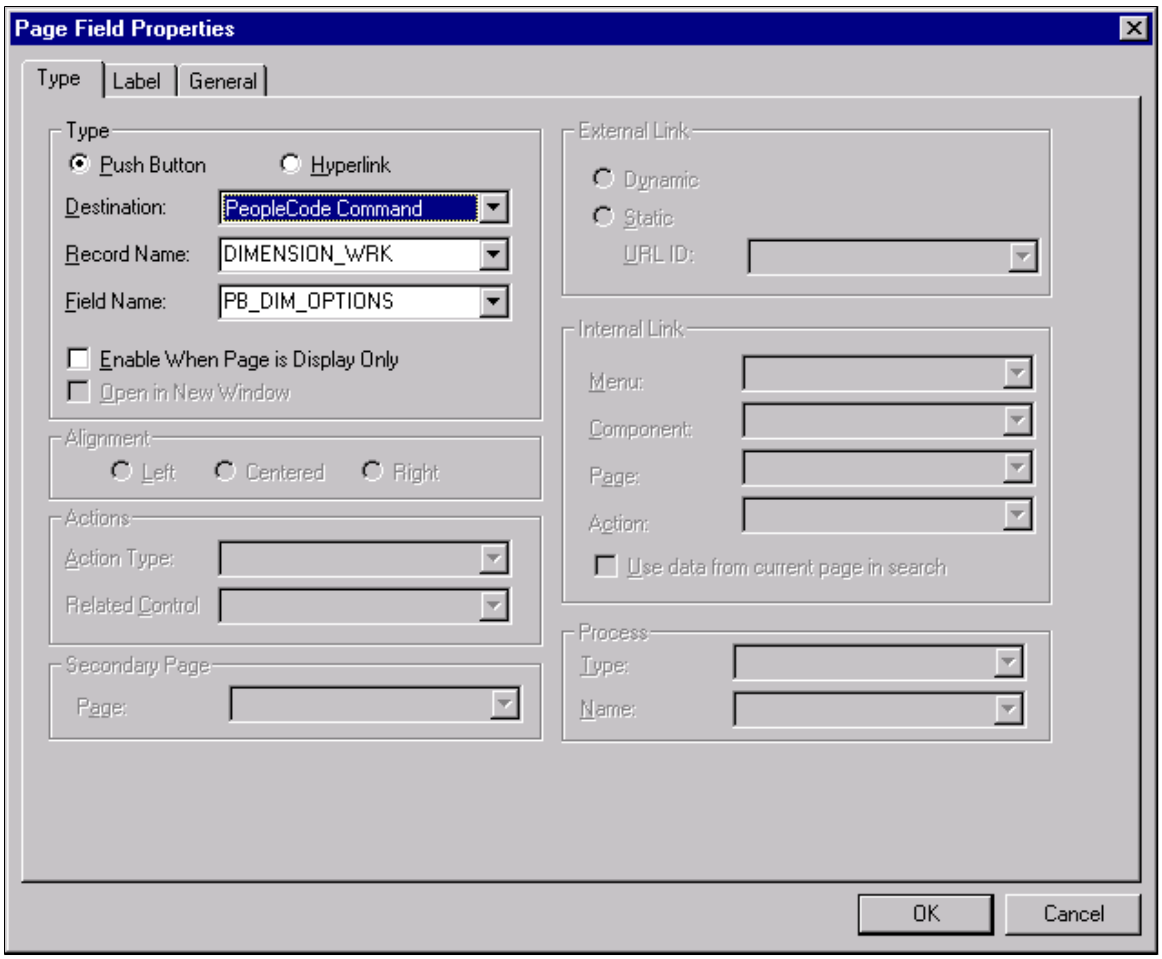
Copyright © 1988, 2021, Oracle and/or its affiliates.

To access record field PeopleCode from a page definition, right-click a page control and select View Record PeopleCode. The PeopleCode Editor appears, displaying the first event in the event set associated with the underlying record field of that control.

Button controls are a special case. You can associate a PeopleCode program with a button only if its destination is defined as PeopleCode Command. When the user clicks a button defined using this method, the FieldEdit and FieldChange events are triggered, so the PeopleCode must be associated with one of those two events. Typically, you use the FieldChange event. The following example shows button properties:

Image: Page Field Properties dialog box for buttons

The following image illustrates Page field properties for buttons.



To define a command button:

1. In the page definition, double-click the button to access its properties.
2. Select *PeopleCode Command* as the button destination.
3. Select the record and field with which your button and PeopleCode are associated.

   You should associate the button with a derived/work record field, which separates its PeopleCode from the PeopleCode associated with any of the page's other underlying record fields. You can then store generic PeopleCode with this field so that you can reuse it with buttons on other pages.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

4. Click OK to return to the page.

   Right-click the command button and select View PeopleCode to access the PeopleCode Editor.

Related Links

Using the PeopleCode Editor

Accessing Component Record Field PeopleCode

This section provides an overview of the component record field event set and discusses how to access component record field PeopleCode.

Understanding Component Record Field PeopleCode

Component record field PeopleCode is associated with a record field, but only with respect to a component and one of its events. Use this type of association to tailor your programs to a particular component. This PeopleCode is accessible only through the component structure view, not from a record definition. The following events are associated with a component record field:

- FieldChange
- FieldDefault
- FieldEdit
- PrePopup

Related Links

PeopleCode Program Triggers

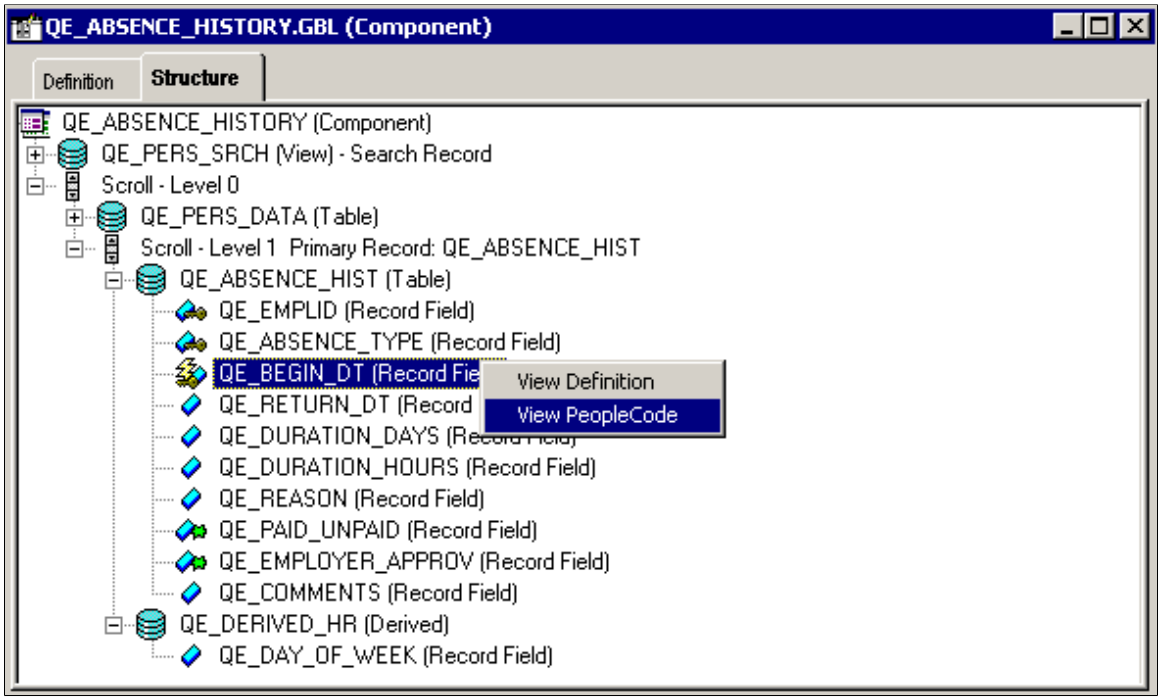
Accessing Component Record Field PeopleCode

To access PeopleCode associated with a component record field, open the component, click the Select tab, select a field, right-click the field name, and select View PeopleCode. A lightning bolt appears next to the field name if PeopleCode is associated with the field at the component level. If PeopleCode is associated with the field at the record level, then a lightning bolt does not appear, as shown in the following example:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Accessing component record field PeopleCode from the component structure

This example illustrates the fields and controls on the Accessing component record field PeopleCode from the component structure. You can find definitions for the fields and controls later on this page.



Note: The Structure tab displays only the runtime state of the PeopleCode. That is, it only displays record field PeopleCode. For example, PeopleCode programs that are orphaned as a result of a page definition change do not appear on the Structure tab. Orphaned PeopleCode programs do appear, however, in the PeopleCode Editor, which displays the design-time view of PeopleCode.

The PeopleCode Editor appears. If that field has associated PeopleCode, then the first program in the component record field event set appears in the editor.

Related Links

Using the PeopleCode Editor
Accessing Record Field PeopleCode

Accessing Component Record PeopleCode

This section provides an overview of the component record event set and discusses how to access component record PeopleCode.

Understanding Component Record PeopleCode

Component record PeopleCode is associated with a record definition, but only with respect to a component and one of its events. Use this type of association to tailor programs to a particular component. This PeopleCode is directly accessible through the component structure view, not from the record definition.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Search records and non-search records in components have different associated event sets. The following events are associated with search records within a component:

- SearchInit
- SearchSave

The following events are associated with non-search records within a component:

- RowDelete
- RowInit

  In rare circumstances, the Component Processor does not run RowInit PeopleCode for some record fields. The Component Processor runs RowInit PeopleCode when it loads the record from the database. However, in some cases, the record can be initialized entirely from the keys for the component. When this happens, RowInit PeopleCode is not run.
- RowInsert
- RowSelect
- SaveEdit
- SavePostChange
- SavePreChange

Related Links

PeopleCode Program Triggers

Accessing Component Record PeopleCode

To access PeopleCode associated with a component record, open the structure view of the component, select a record, right-click the record name, and select View PeopleCode.

The PeopleCode Editor appears. If the record has associated PeopleCode, then the first program in the component record event set appears in the editor.

Related Links

Using the PeopleCode Editor

Accessing Component PeopleCode

This section provides an overview of the component event set and discusses how to access component PeopleCode.

Understanding Component PeopleCode

Component PeopleCode is associated with a component definition and an event. The following events can be associated with a component:

Copyright © 1988, 2021, Oracle and/or its affiliates.

- PostBuild
- PreBuild
- SavePostChange
- SavePreChange
- Workflow

Related Links

PeopleCode Program Triggers

Accessing Component PeopleCode

To access PeopleCode associated with a component, open its structure view, select the component name, right-click the name, and select View PeopleCode.

The PeopleCode Editor appears. If the component has associated PeopleCode, the first program in the component event set appears in the editor.

Related Links

Using the PeopleCode Editor

Accessing Page PeopleCode

This section provides an overview of page PeopleCode and discusses how to access page PeopleCode.

Understanding Page PeopleCode

Page PeopleCode is associated with a page definition. The page event set consists of a single event, the Activate event, which fires every time the page is activated. This event is valid only for pages that are defined as standard or secondary, and it is not supported for subpages.

Related Links

PeopleCode Program Triggers

Accessing Page PeopleCode

To access PeopleCode associated with a page, right-click any part of the page definition and select View Page PeopleCode.

Note: Page PeopleCode can only be accessed in this way. You cannot access Page PeopleCode from the component definition Structure tab, from a project, or any other way.

The PeopleCode Editor appears. If the page has associated PeopleCode, it appears in the editor.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Note: The term *page PeopleCode* refers to PeopleCode programs owned by pages. Do not confuse *page PeopleCode* with PeopleCode properties related to the appearance of pages, such as the Visible Page Class property.

Related Links

"Understanding Page Class" (PeopleTools 8.59: PeopleCode API Reference)
Using the PeopleCode Editor

Accessing Menu Item PeopleCode

This section provides an overview of menu item PeopleCode and discusses how to:

- Define PeopleCode pop-up menu items.
- Access menu item PeopleCode.

Related Links

"Working With Menu Definitions" (PeopleTools 8.59: Application Designer Developer's Guide)
Using the PeopleCode Editor

Understanding Menu Item PeopleCode

PeopleTools menus are one of two types, either pop-up or standard, both of which are standalone definitions in the project hierarchy. However, you can only associate PeopleCode with menu items in pop-up menus.

The menu item event set consists of a single event, the ItemSelected Event. This event fires whenever an user selects a menu item from a pop-up menu.

Note: Do not confuse *menu item PeopleCode* with PeopleCode functions related to the appearance of menu items, such as CheckMenuItem.

Related Links

ItemSelected Event

Defining PeopleCode Pop-Up Menu Items

To define a PeopleCode pop-up menu item:

1. In the open pop-up menu definition, double-click the menu item to access its properties.

   If you are creating a new menu item, double-click the empty rectangle at the bottom of the pop-up menu.

   The Menu Item Properties dialog box appears.

2. If this is a new menu item, enter a name and a label for the item.

Copyright © 1988, 2021, Oracle and/or its affiliates.

3. Select *PeopleCode* from the Type group box.
4. Click OK to close the Menu Item Properties dialog box.

Accessing Menu Item PeopleCode

To access pop-up menu item PeopleCode:

1. Open the pop-up menu definition.
2. Right-click the menu item and select View PeopleCode.

   The PeopleCode Editor appears, displaying the associated program for that menu item, if any.

Copying PeopleCode with a Parent Definition

When you copy a Application Designer definition that contains PeopleCode, you can choose whether to copy all PeopleCode programs and the definition. Each copy of the definition receives a separate copy of the PeopleCode programs.

To copy a definition with its PeopleCode:

1. Open the definition you want to copy.
2. Select File, Save As.

   The Save As dialog appears.
3. Type a name for the new definition in the dialog box.
4. Click OK, and then click Yes to copy the PeopleCode.

   Click Yes to copy all PeopleCode associated with the definition.

Upgrading PeopleCode Programs

You can upgrade PeopleCode programs independently of the definitions with which they are associated. Refer to the upgrade instructions for your product for details.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 12

Using the PeopleCode Editor

Navigating Between PeopleCode Programs

After you access a PeopleCode program associated with a Application Designer definition, you can access programs associated with other related definitions without having to close the editor window.

This section provides an overview of the PeopleCode Editor window and discusses how to:

- Navigate between programs associated with a definition and its children.
- Navigate between programs associated with a definition's event set.

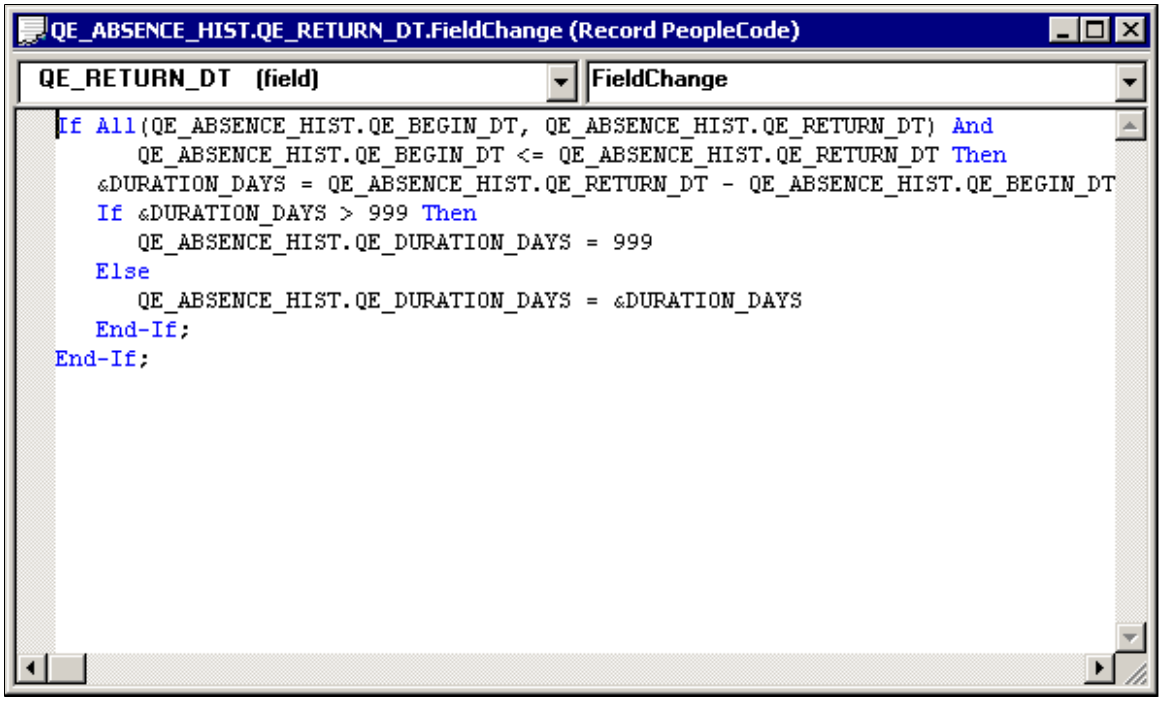
Related Links

Understanding PeopleCode Programs and Events

Understanding the PeopleCode Editor Window

Image: PeopleCode Editor window with record field PeopleCode

Application Designer supplies an independent editor window for each parent definition, such as a record, component interface, or an analytic model, for which you invoke the editor. The editor window's title bar displays the name and type of the parent definition, as shown in the following illustration:



Copyright © 1988, 2021, Oracle and/or its affiliates.

The editor window contains the main edit pane, the drop-down definition list at the upper-left, and the drop-down event list at the upper-right. The drop-down lists enable you to navigate directly to the PeopleCode associated with related child definitions, for example, fields within a record and their event sets.

Note: When you make a selection from either drop-down list box, your selected entry has a yellow background, indicating that you must click the edit pane before you can start typing.

You can open as many editor windows as you want and resize them in Application Designer. Each line of code wraps automatically based on the window's current width. A vertical scroll bar appears if the program has more lines than the editor can display in the edit pane.

Note: You cannot open two editor windows for a single parent definition, or for any two of its child definitions.

Related Links

Navigating Between Programs Associated With a Definition and Its Children
Navigating Between Programs Associated With Events

Navigating Between Programs Associated With a Definition and Its Children

You use the drop-down definition list to navigate between PeopleCode programs that are associated with a parent definition and its children. The list displays the complete hierarchy of child definitions to which you can navigate; bold items have PeopleCode associated with at least one event in the item's event set. The structure of the definition list depends on the type of parent definition. Parent definitions include:

- Records.

  Select record fields from the record drop-down list. The record name appears at the top of the list as a visual clue to clarify the location of the record fields, but you cannot associate PeopleCode with a record.

- Components.

  Select component records and component record fields from the component drop-down list.

- Pages.

  Select the page definition from the page drop-down list.

- Pop-up menus.

  Select pop-up menu items from the menu drop-down list. The menu and menu bars appear in the list as visual clues, but you cannot associate PeopleCode with these elements.

Navigating Between Programs Associated With Events

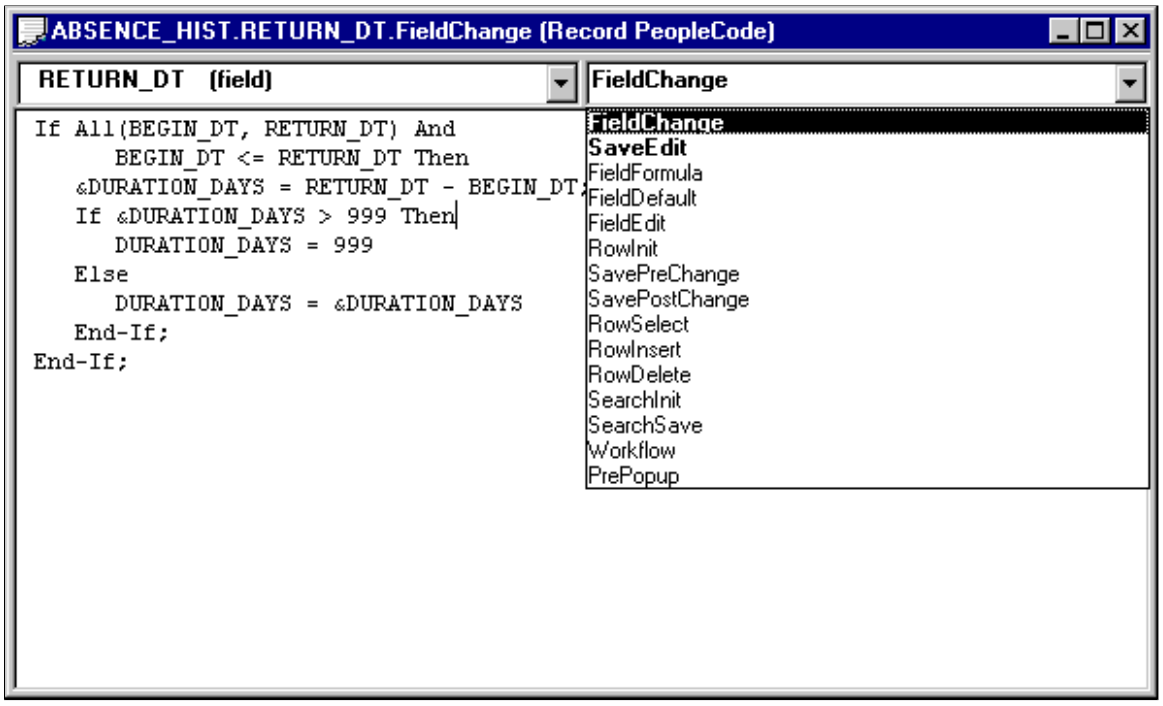
Use the PeopleCode Editor's drop-down event list to select an event from the event set of the currently selected definition. Use this event list to navigate between PeopleCode programs that are associated with

 Copyright © 1988, 2021, Oracle and/or its affiliates.

that definition. For every definition-event combination with associated PeopleCode, the event name is displayed in bold, and it appears at the top of the event list, as shown in the following illustration:

Image: Selecting an event from the PeopleCode Editor

This example illustrates the fields and controls on the Selecting an event from the PeopleCode Editor. You can find definitions for the fields and controls later on this page.



Related Links

Understanding PeopleCode Programs and Events
PeopleCode Program Triggers

Using the PeopleCode Editor

This section provides an overview of the PeopleCode Editor and color-coded language elements and discusses how to:

- Access PeopleCode Editor.
- Editing functions.
- Format code automatically.
- Use drag-and-drop editing.
- Access PeopleCode external functions.
- Access definitions and associated PeopleCode.
- Access help.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Set up help.
- Change colors in the PeopleCode Editor.
- Select a font for the PeopleCode Editor.
- Change word wrap in the PeopleCode Editor.
- Use PeopleCode Event properties.
- Use auto-complete in the PeopleCode Editor.
- Use event mapping button in the PeopleCode Editor..

Understanding the PeopleCode Editor

The PeopleCode Editor works much like any other text editor, but has capabilities specifically geared toward the PeopleTools environment. Some of its features include:

- Editing functions are integrated with the menus and toolbar of Application Designer and are also accessible from a pop-up window.
- It checks, formats, and saves all programs associated with Application Designer definitions simultaneously when any definition is saved.
- It includes a Validate Syntax command for checking and formatting a single PeopleCode program without saving.
- It supports standard Microsoft Windows drag-and-drop editing.
- It supports color-coding for the different elements of the PeopleCode language.
- It supports word wrap based on either the size of the editor window or a specific number of characters per line.
- You can open separate instances of the editor simultaneously, and you can use a drag-and-drop text operation between programs.
- You can open the definition with which the current set of PeopleCode programs is associated from within the PeopleCode Editor.
- You can open a field, record, page, file layout, or other definitions from a PeopleCode reference to the field, record, page, or file layout, and so on.
- You can access PeopleCode programs associated with a field, record, page or file layout, or other definitions from a PeopleCode reference to the field, record, page, or file layout, and so on.
- You can open a PeopleCode Editor window containing an external function definition from a function declaration or function call.
- You can press F1 with the cursor in a PeopleCode built-in function, method, meta-SQL, and so on, to open the PeopleSoft help for that item.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Accessing PeopleCode Editor

When a definition is open in Application Designer, you can access the PeopleCode Editor by:

- Clicking the View PeopleCode button in the toolbar.
- Right-clicking the definition and selecting View PeopleCode from the pop-up menu.
- Selecting the definition and pressing Ctrl + E.

To access the PeopleCode Editor for an Application Class:

- Double-click the Application Class.
- Select the Application Class and press Ctrl + E.

Using the View PeopleCode Button in the Toolbar

For definitions where PeopleCode can be inserted, the View PeopleCode button appears in the dynamic Application Designer toolbar. Click this button to launch the PeopleCode Editor and view the PeopleCode associated with the current definition.

The View PeopleCode button appears for these definition types:

- records
- pages
- components
- menus
- component interfaces
- messages
- Application Engine programs

Note: Depending on the definition type, the label of the button and the number of buttons that appear might be different. For example, with pages, you see two buttons: View Page PeopleCode and View Record PeopleCode.

Selecting View PeopleCode from the Pop-up Menu

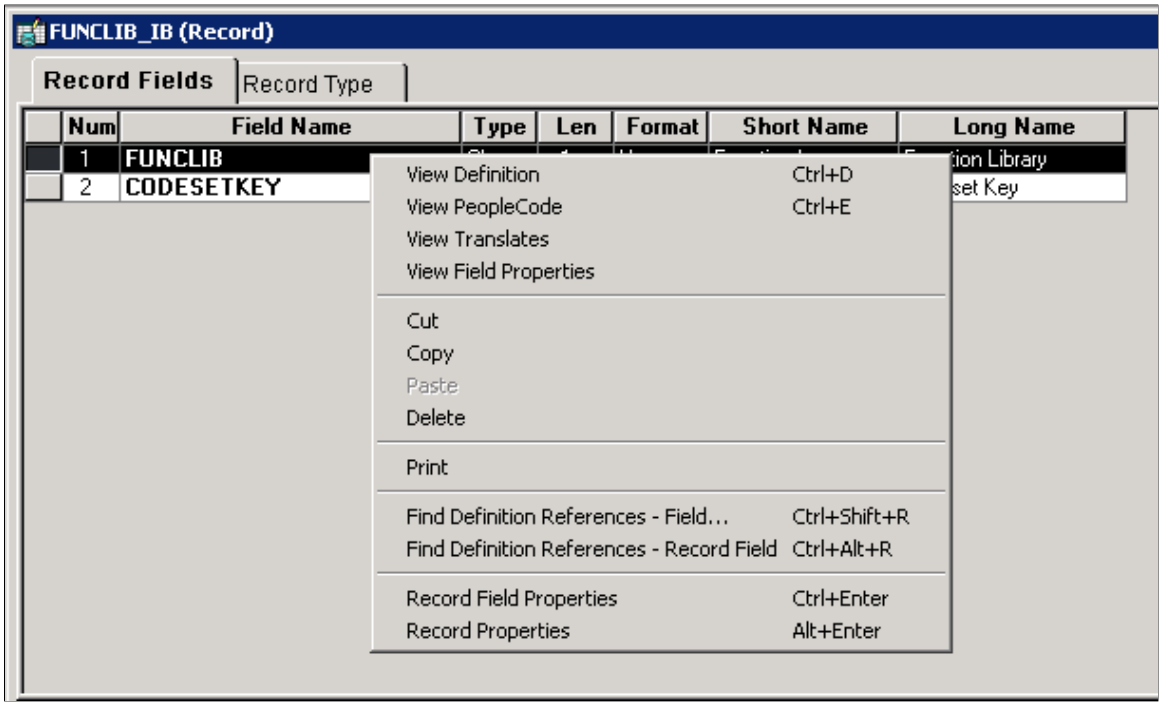
Open a definition in Application Designer, and in the Application Designer workspace, right-click a field, which displays a pop-up menu. Click View PeopleCode to launch the PeopleCode Editor.

Note: Depending on the definition type, the options to launch PeopleCode Editor might be different. For example, with pages, you see two options: View Page PeopleCode and View Record PeopleCode.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Example of accessing PeopleCode Editor from pop-up menu

This example illustrates accessing PeopleCode Editor from the current definition open in the Application Designer workspace.



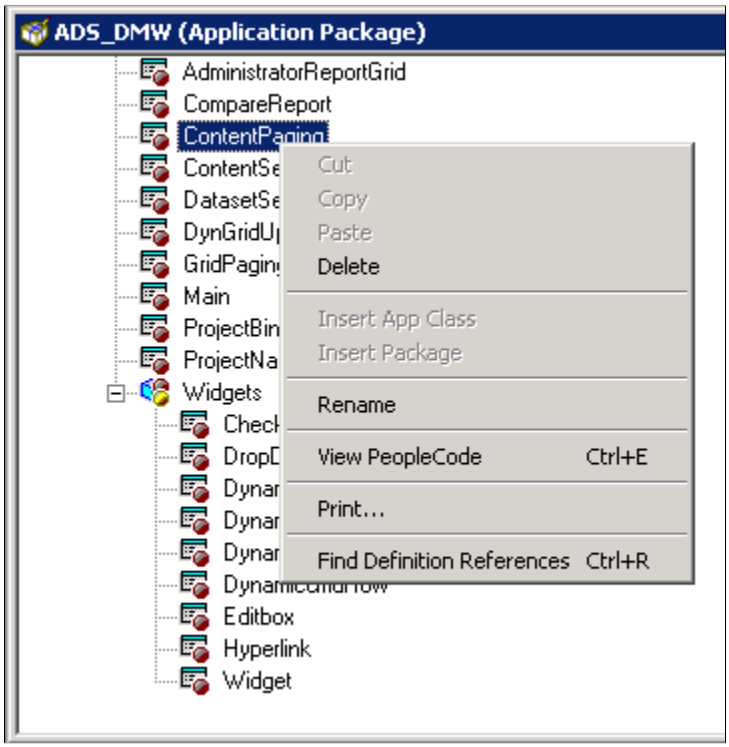
You can also access PeopleCode Editor by selecting the definition and then pressing Ctrl + E on the keyboard.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Accessing PeopleCode Editor for an Application Class

Image: Accessing PeopleCode Editor for an Application Class

This example illustrates accessing PeopleCode Editor for the Application Class from the Application Designer workspace.



You can either double click on the selected Application Class or press Ctrl + E on the keyboard after selecting the Application Class to access its PeopleCode Editor.

Editing Functions

The PeopleCode Editor supports standard editing function commands such as Save, Cancel, Cut, Copy, Paste, Find, Replace, and Undo, from the PeopleCode Editor pop-up menu. Cut, Copy, and Paste use standard Microsoft Windows keyboard shortcuts. You can also cut, copy, and paste within the same PeopleCode program or across multiple programs.

Use these buttons to perform editing functions:

	Save the current PeopleCode program. You can also use the key combination CTRL+S.
	Cut the selected text or item. You can also use the CTRL+X or SHIFT+DEL key combinations.
	Copy the selected text or item. You can also use the CTRL+C or CTRL+INS key combinations.
	Paste from the clipboard. You can also use the CTRL+V or SHIFT+INS key combinations.

Copyright © 1988, 2021, Oracle and/or its affiliates.

	Find specified text. You can also use the key combination CTRL +F.
	Find and replace specified text. You can also use the key combination CTRL+H.
	Validate the current PeopleCode program.
	Undo the last change. Use the CTRL+Z or ALT+BACKSPACE key combinations.
	Cancel the current operation. Use Esc key.

Related Links

Short Cut Keys in the PeopleCode Editor

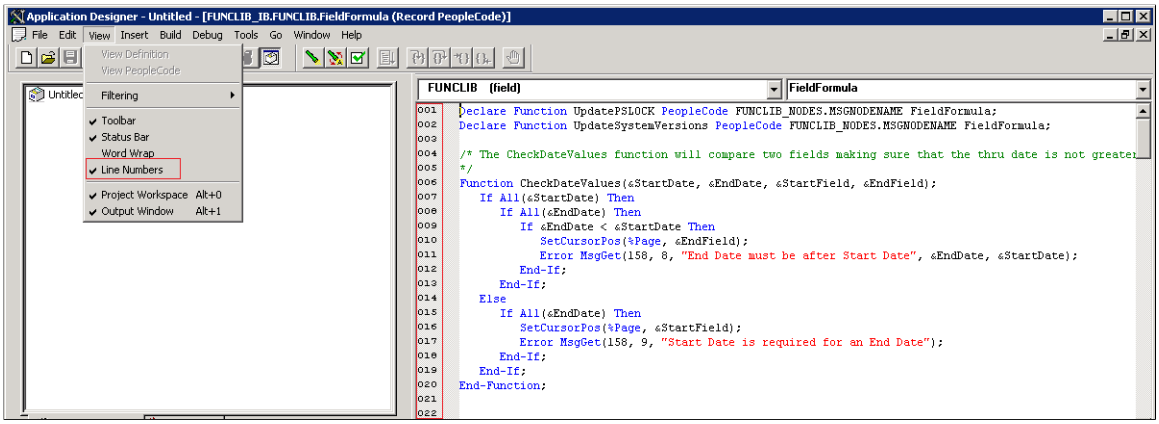
Adding Line Numbers

To add line numbers in PeopleCode Editor:

1. Go to the menu bar of Application Designer.
2. Click on View.
3. Select Line Numbers.

Image: Adding Line Numbers

The following image illustrates how line numbers are displayed in PeopleCode Editor:



To remove line numbers from PeopleCode Editor, uncheck Line Numbers.

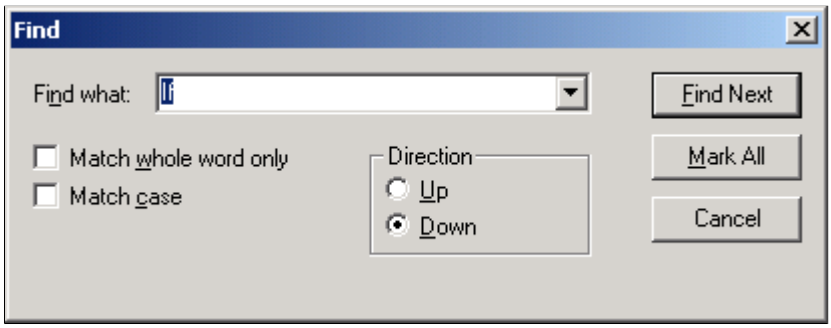
Note: Line numbers in PeopleCode Editor are not the same as statement numbers in debug mode. While line numbers are valid for all the lines in PeopleCode editor including comments, declarations and new lines, statement numbers do not include the declare functions and functions that are not debuggable.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Find and Replace Dialogs

Image: Find dialog box

When you use the Find and Replace functions, any text string that is highlighted appears when either the Find or Replace dialog boxes are called. For example, if you select the method ActiveRowCount it appears in the Find dialog box when it's called, as shown in the following example:



You can move through finding and replacing text strings one string at a time, or click Replace All to replace globally. The Undo function is available to undo the last replace or replace all.

The Mark All button places a bookmark next to all lines that have the matching text. Use Shift+CTRL+f2 to remove all bookmarks.

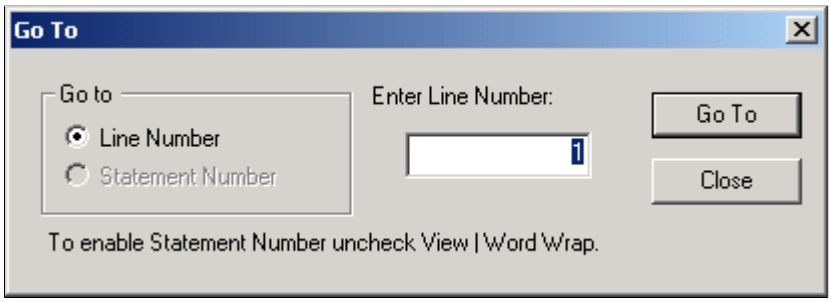
With the Replace dialog box, you can select to replace text either in a selected section or a whole file (that is, a PeopleCode program.)

Go To Dialog

Use the Go To dialog box to specify a line number in the current program, then go to that line. If you have line wrap not enabled, you can specify to go to statement numbers instead of line numbers.

Image: Go To dialog box

This example illustrates the fields and controls on the Go To dialog box.



Validate Syntax Utility

To check the syntax of the current PeopleCode program and format it if it is syntactically correct, do one of the following:

- Click the Validate Syntax button on the Application Designer toolbar.
- Within Application Designer, select Tools, Validate Syntax.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Right-click in the PeopleCode Editor window, then select Validate Syntax.

The Validate utility has several functions, such as finding undeclared variables, mismatching data types, or invalid methods or properties for a class. You can check either a single component or an entire project.

Errors or warnings produced by the Validate utility are displayed in the Validate tab at the bottom of the PeopleCode Editor window.

Any variables that you don't declare are automatically declared for you, and a warning message appears in the Validate tab for each undeclared variable. You can right-click in the Validate tab and select Clear to delete all the warnings listed there, then use the Validate utility again to ensure that your code runs without errors or warnings.

Note: This feature is convenient if you have written multiple PeopleCode programs and you want to check the syntax of one without saving. All PeopleCode programs associated with an item (record, component, and so on) are checked prior to saving.

Related Links

"Validating Projects" (PeopleTools 8.59: Application Designer Developer's Guide)

Compiling All PeopleCode Programs at Once

Show Database Name Utility

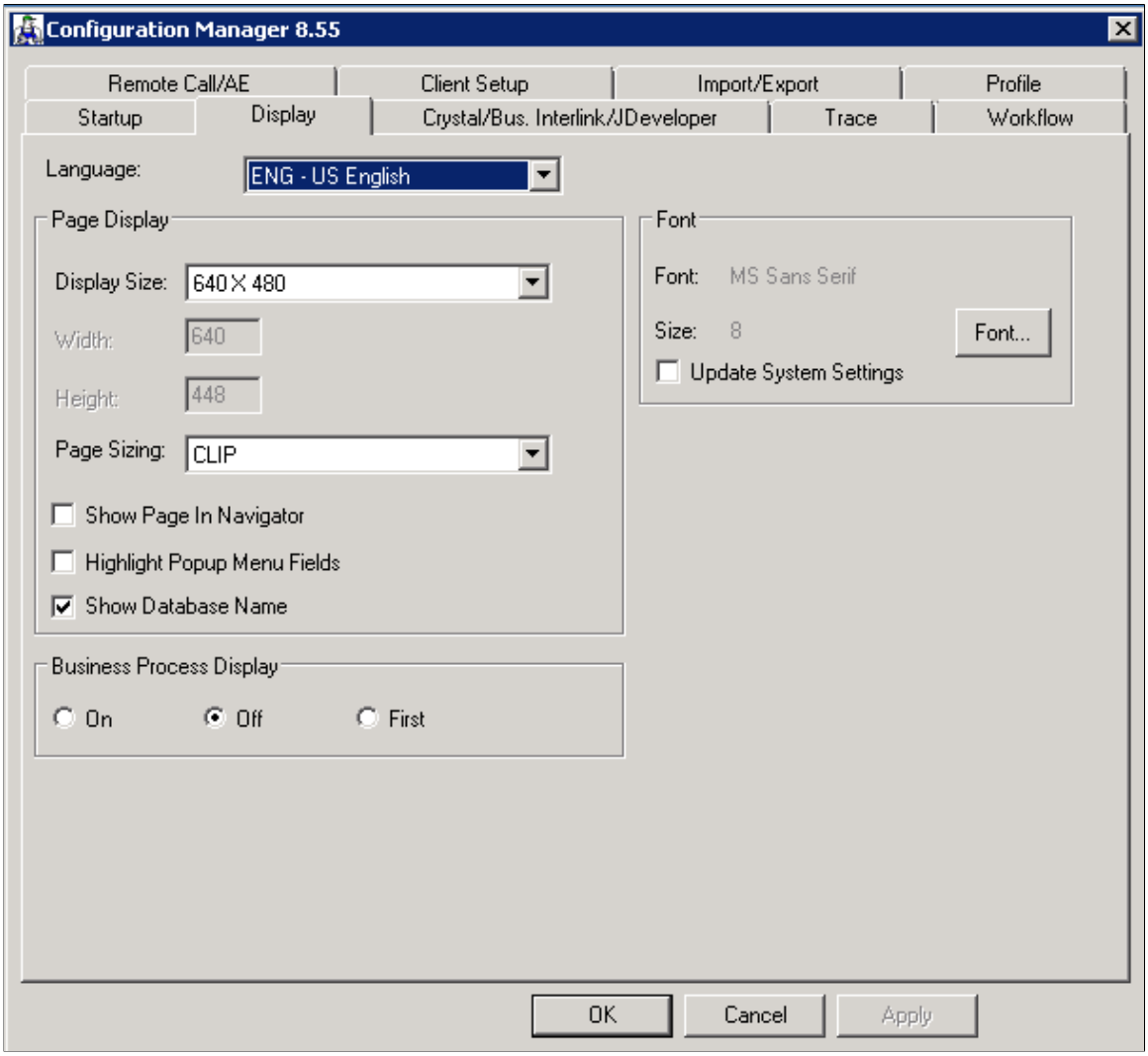
To display the name of the database in use in the current PeopleCode program:

1. Open Configuration Manager.
2. Click on the Display tab.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Display tab in Configuration Manager:

The following image illustrates the Display tab in Configuration Manager:



3. Select the Show Database Name check box.

   When you access Application Designer, the name of the database in use is displayed at the bottom right corner of the status bar.

Related Links

"Understanding PeopleSoft Configuration Manager" (PeopleTools 8.59: System and Server Administration)

"Starting PeopleSoft Configuration Manager" (PeopleTools 8.59: System and Server Administration)

Formatting Code Automatically

You do not need to format your PeopleCode statements; you need only to use the correct syntax. When you save or validate, the system formats the code according to the rules in the PeopleCode tables, no

Copyright © 1988, 2021, Oracle and/or its affiliates.

matter how you entered it originally. The PeopleCode Editor automatically converts field names to uppercase and indents statements.

PeopleCode is case-insensitive, except for quoted literals. PeopleCode does not format anything surrounded by quotation marks. String comparisons, however, are case-sensitive. When you compare the contents of a field or a variable to a string literal, make sure the literal is in the correct case.

All field names in a PeopleCode program must be fully qualified, even if the field is on the same record definition as the PeopleCode program. However, you only need to type in the name of the field. The editor validates if the field exists on the current record, and reformats the field name to *recordname.fieldname*.

Using Drag-and-Drop Editing

In addition to the standard keyboard shortcuts and toolbar buttons, you can copy or move text within a window or between two PeopleCode Editor windows by using the mouse and the CTRL key.

Note: You cannot open two editor windows for a single parent definition, or for any two of its child definitions.

To move text between instances of the PeopleCode Editor:

1. Select the text you want to move.
2. Place the mouse over the text and drag the text to the other PeopleCode Editor window.
3. When the cursor appears at the place where you want to insert the text, release the mouse button.

To copy text between instances of the PeopleCode Editor:

1. Select the text you want to move.
2. Hold down the CTRL key as you drag the text to the other PeopleCode Editor window.
3. When the cursor appears at the place where you want to insert the text, release the mouse button.

Accessing PeopleCode External Functions

An external PeopleCode function is a function written in PeopleCode (as opposed to a built-in function or external DLL function) and defined in a program outside the one from which it is called. External PeopleCode functions can be defined in any record PeopleCode program, but typically they are stored in the FieldFormula event in records beginning with *FUNCLIB_*.

The PeopleCode Editor provides immediate access to external PeopleCode function definitions. Right-click the function name in the program where the function is called, then select View Function *FunctionName.* This opens a new PeopleCode Editor window containing the external function definition.

Note: Internet scripts are contained in records similar to FUNCLIB_ records. However, their names begin with *WEBLIB_*.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Accessing PeopleCode Application Packages and Application Classes

The PeopleCode Editor provides immediate access to application packages, application classes, and application class method definitions.

Right-click the package, class, or method name and, depending on the context, select from:

- View Application Package
- View Application Class
- View Application Class Method

This opens the application package or a new PeopleCode Editor window containing the application class.

Image: Context menu with options view to application packages and classes

The following example shows the context menu for a fully-qualified application class name.

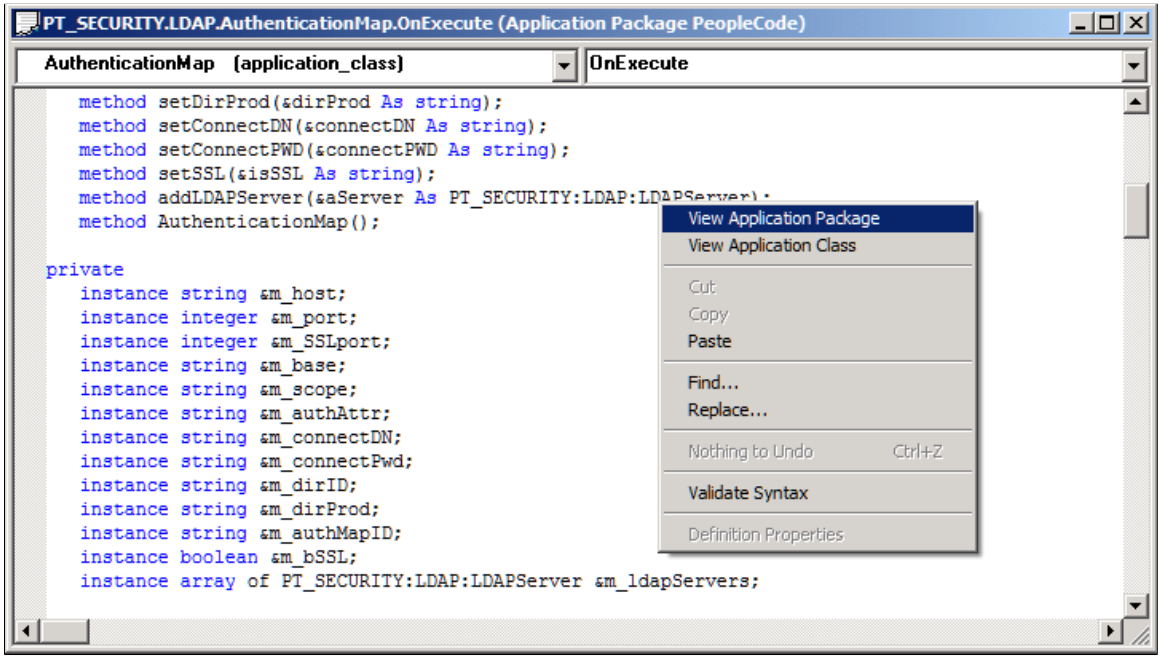
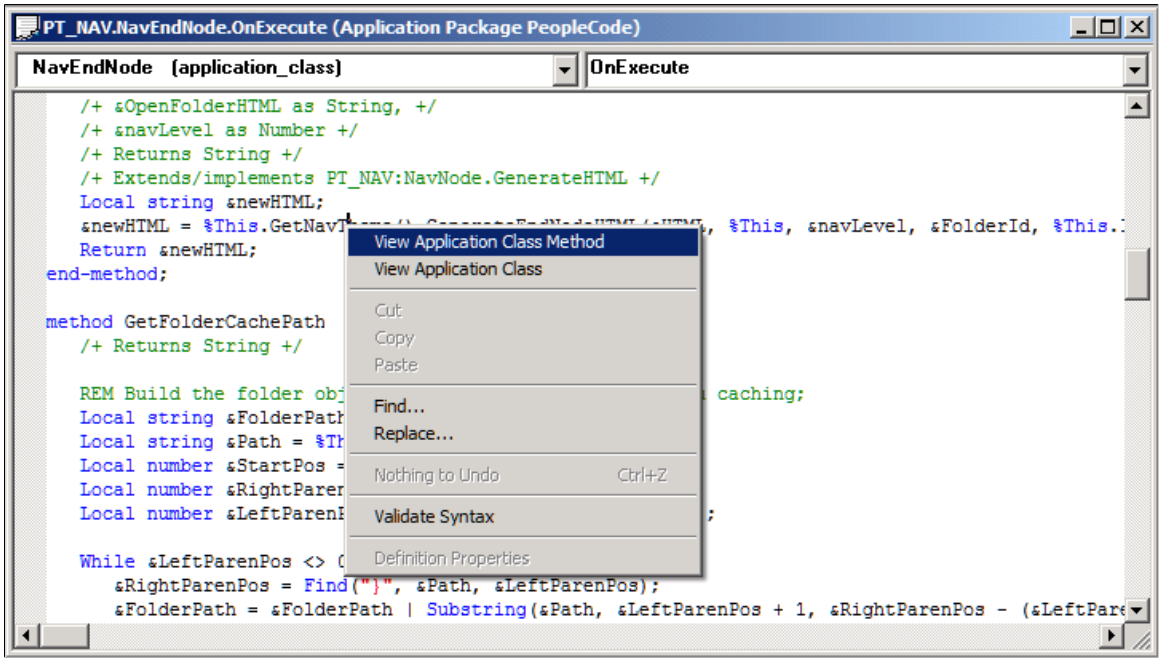


Image: Context menu with options to view application classes and methods

The following example shows the context menu for a method.



Note: The application class context menu is not available for methods that are called by indirection. In the following example the method CallMe would not be available to view using the context menu.

```
Object0.GetObject().CallMe();
```

Accessing Methods in Derived Classes

A method that is defined only in the superclass is not available if you attempt to view it using View Application Class Method with the derived class, or subclass.

For example, in the following code snippet CCI_CRM extends CCI_BASE. The method Validate is not defined in CCI_CRM; it is available to CCI_CRM by extension. The method Submit, on the other hand, is overridden in CCI_CRM.

If you right-click Validate and select View Application Class Method, the cursor will be placed at the beginning of the application class CCR_CRM, not at the method definition in CCI_BASE.

If you right-click Submit and select View Application Class Method, you will be taken to the method definition for Submit in CCI_CRM.

```
Import EOCC:CCI_CRM;
&CCI = Create EOCC:CCI_CRM();
&CCI.Validate(&Card);
&CCI.Submit(&Card);
```

This may be helpful when you need to know whether a method has been overridden.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Accessing Definitions and Associated PeopleCode

You can open fields, records, pages, application packages, and other definitions from the PeopleCode Editor. Or you can open a new PeopleCode Editor window containing the programs associated with a field, record, page, application class, or other definition.

To open a definition from the PeopleCode Editor, right-click a PeopleCode definition reference and select View Definition or View Application Package.

For example, you could open definitions by clicking the following references:

- Record.BUS_EXPENSE_PER
- BUS_EXPENSE_PER.EXPENSE_PERIOD_DT
- Page.BUSINESS_EXPENSES
- PT_BRANDING:BrandingBase

If you access a record definition from a record field reference (that is, *recordname.fieldname*) the specified record field is selected when the record definition opens.

To open a new PeopleCode editor window, right-click a reference to the definition and select View PeopleCode or, for application class PeopleCode, select View Application Class Method or View Application Class.

For example, you can access record PeopleCode from the following record and record field references:

- Record.BUS_EXPENSE_PER
- BUS_EXPENSE_PER.EXPENSE_PERIOD_DT

Note: You can only view the PeopleCode and definition when the text is in the format *recordname.fieldname.* If the text is in the format *method(i).recordname, method(i).fieldname,* or *&MyRecord.Fieldname,* the View PeopleCode and View Definition commands are not available.

You can access application class PeopleCode from the following references:

- PT_BRANDING:BrandingBase
- %This.ValidateSave(&aErrs)

Accessing Help

The PeopleCode Editor has context-sensitive online help for all PeopleCode built-in functions, methods, properties, system variables, and meta-SQL. To access online help, place the cursor in the item that you want to look up, then press F1. If there is a corresponding entry in the online reference system it appears; otherwise an error message similar to the following appears.

```
Help cannot be displayed because no help topic was found for the help context ID "m⇒
y_topic".
```

If more than one entry is applicable, a pop-up window that lists all applicable entries appears. Select the entry for the item of interest.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Setting Up Help

To set up the help, use the F1 Help URL field on the PeopleTools Options page to specify where the PeopleTools documentation is installed.

Related Links

"PeopleTools Options" (PeopleTools 8.59: System and Server Administration)

Changing Colors in the PeopleCode Editor

You can change the display (foreground) color for many language elements in the PeopleCode Editor, including quoted strings, keywords, and built-in functions. You can also change the background color.

To change the display colors:

1. Select Edit, Display Font and Colors.
2. Select the language element that you want to change.
3. Select the foreground color.

   If you click the Automatic check box, the default color is used.

   A box displaying the selected color is only available if the Automatic check box is not selected.

   If you click the box displaying the selected color, the standard color chart for your display appears. If you click Other from this dialog or click the drop-down list on the Font and Color Settings dialog box, the custom color chart for your display appears.
4. Select the background color.

If you click Reset All, the default colors for the PeopleCode language elements are reassigned.

Selecting a Font for the PeopleCode Editor

The default font for the PeopleCode Editor is 9-point Courier New.

To change the PeopleCode Editor font, select Edit, Display Fonts and Colors. Use this dialog box to change the font for the editor.

Note: When you select a font for the PeopleCode Editor, the font selection dialog box provides choices based on a character set appropriate for your international version of Microsoft Windows. If you experience trouble embedding foreign characters (such as Thai characters) in PeopleCode, you might need to change the font setting. If you are trying to display Thai characters in Microsoft Windows 95, you might also need to change your keyboard input settings for the characters to display correctly. You can change your keyboard input settings from the Input Locales tab on the Windows Regional Settings control panel, or on the Keyboard control panel.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Changing Word Wrap in the PeopleCode Editor

The PeopleCode Editor supports text word wrapping. You can turn word wrapping on and off for an open editor window. You can also specify the default value for word wrap, as well as whether the text wraps to the editor's window size or to a fixed number of characters per line.

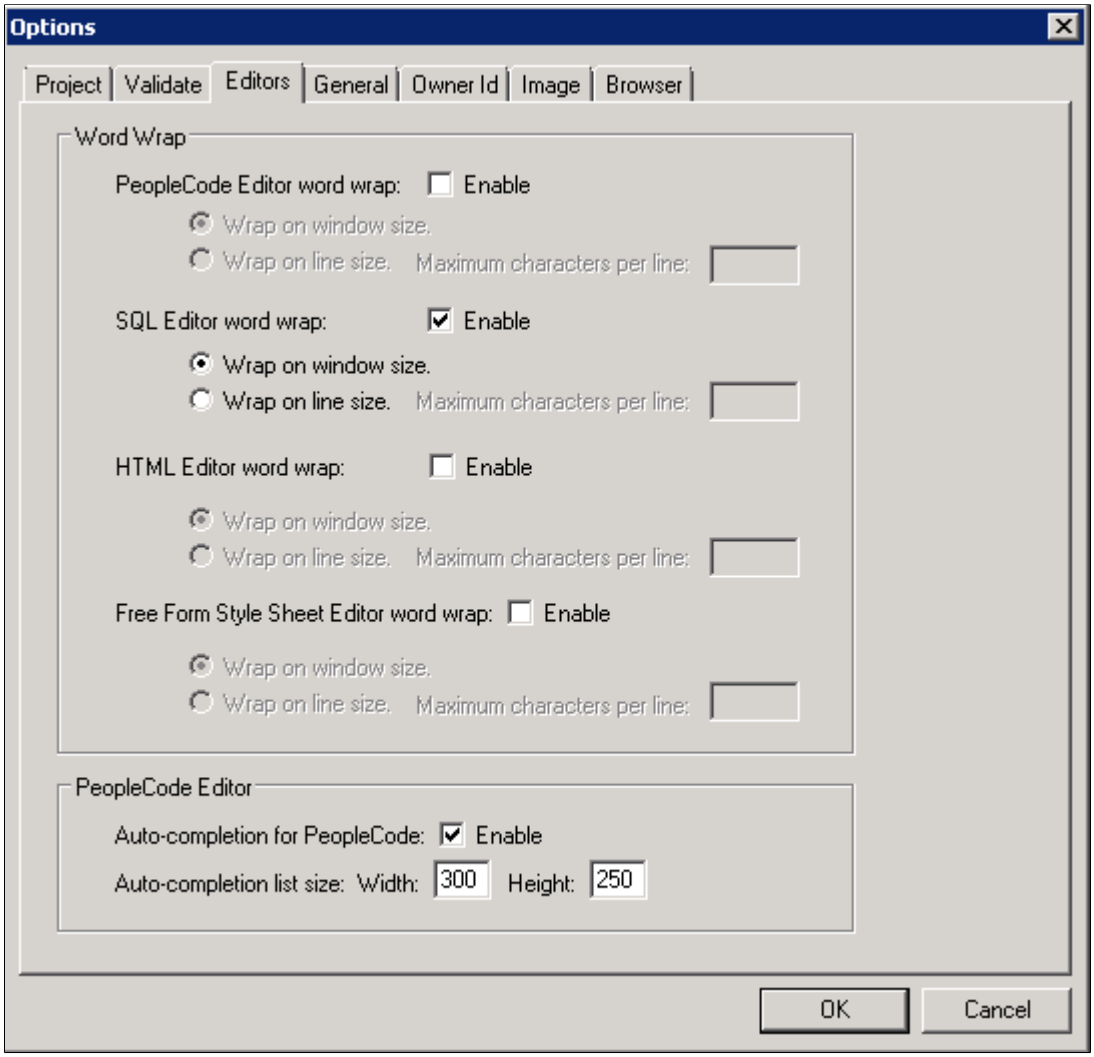
To turn word wrapping on or off for an open editor window, go to Edit, Word Wrap. After you close Application Designer, all word wrap values are reset to the default value for the editor.

Specifying Word Wrap Options

Go to the Tools, Options dialog box, Editors tab, to specify the word wrap options.

Image: Options dialog box: Editors tab

This example illustrates the fields and controls on the Options dialog box: Editors tab. You can find definitions for the fields and controls later on this page.



Enable (word wrap) Specify whether word wrap is the default mode when opening the editor. If this box is not checked, wrapping text based on window size is the default.

Copyright © 1988, 2021, Oracle and/or its affiliates.

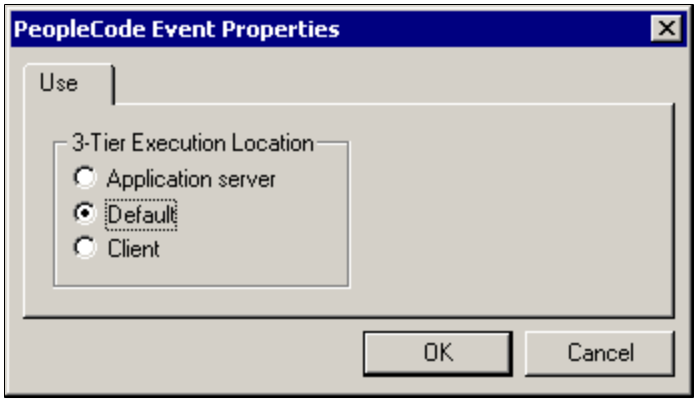
Wrap on Window Size	Specify whether the text wraps based on the size of the window.
Wrap on Line Size	Specify whether the text wraps based on the number of characters in a line. If this box is checked, you can specify the number of maximum number of characters per line.
Maximum Characters per Line	Specify the maximum number of characters allowed for a line before the text wraps. The default value is 90. Valid values are between 25 and 2000.

Using the PeopleCode Event Properties

To access the PeopleCode Event properties, open a PeopleCode editor window, then either press Alt + Enter or click the Properties button.

Image: PeopleCode Event Properties dialog box

This example illustrates the fields and controls on the PeopleCode Event Properties dialog box.



Important! This dialog box has been deprecated. It has no effect on the location of the execution of code.

Using the Auto-Complete Feature

The PeopleCode Editor provides an auto-complete feature for PeopleCode application classes, built-in classes, built-in functions, system variables and constants. The auto-complete feature displays the possible options when you insert a period (.) or dot after an object.

The auto-complete feature also supports the display of a tool tip when you hover the mouse over the selected list item. The tool tip displays the method name, parameter type, and the return type.

If you select any item listed in the auto-complete list control and then press F1, it will take you to the context-sensitive online help for that item.

The auto-complete feature is not applicable for objects that are declared as automatic variable and objects that get resolved at run-time.

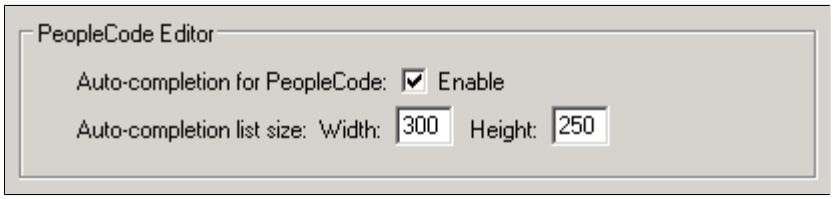
Copyright © 1988, 2021, Oracle and/or its affiliates.

Setting Up Auto-Complete Feature

To use the auto-complete feature, you must enable the Auto-completion for PeopleCode option on the Editors tab of the Options dialog box.

Image: Options page - PeopleCode Editor group box

This example illustrates the fields for the auto-complete feature.



PeopleCode Built-in Class Objects

If an object is created out of any built-in classes, such as Record, Document, Feed, the methods or properties of the appropriate class are displayed in a list from which an application developer can choose the required method or property. The list displays the set of methods and properties relevant to the object. As the user types in a string for a method, the nearest matching item gets displayed at the top of the auto-complete list control

Methods and properties are differentiated by using different icons.

For example, in the following code snippet, when you enter a period (.) after the object `&sqlObject`, methods and properties are displayed in a list box.

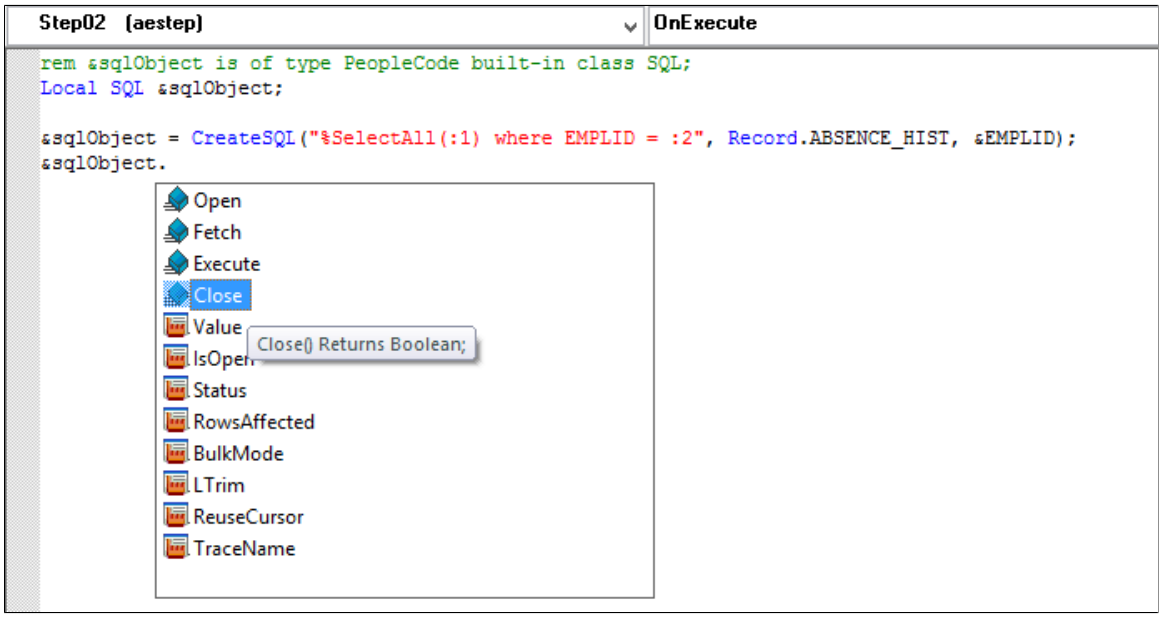
```
rem &sqlObject is of type PeopleCode built-in class SQL;
Local SQL &sqlObject;

&sqlObject = CreateSQL("%SelectAll(:1) where EMPLID = :2", RECORD.ABSENCE_HIST, &EM⇒
PLID);
&sqlObject.
```

Note: The auto-complete feature is not applicable if you create an object as an automatic variable without using the PeopleCode built-in class.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Auto-complete list box for built-in classes

This example illustrates the auto-complete list box for built-in classes.



Application Class Objects

If an object is declared and created from an application class (PeopleTools delivered or customized), the methods or properties of the appropriate class are displayed in a list box.

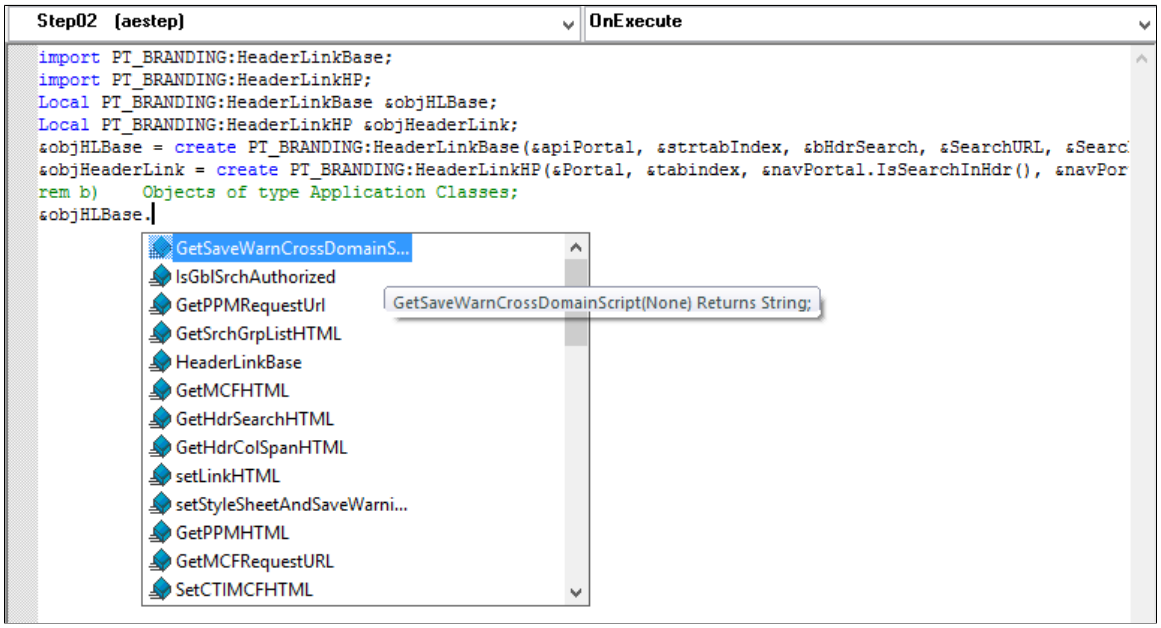
For example, in the following code snippet, when you enter a period (.) after the object `&objHLBase`, methods and properties are displayed in a list box.

```
import PT_BRANDING:HeaderLinkBase;
import PT_BRANDING:HeaderLinkHP;

Local PT_BRANDING:HeaderLinkBase &objHLBase;
Local PT_BRANDING:HeaderLinkHP &objHeaderLink;

&objHLBase = create PT_BRANDING:HeaderLinkBase(&apiPortal, &strtabIndex, &bHdrSearc⇒
h, &SearchURL, &SearchASLimit, &bGblHdrSearch);
&objHeaderLink = create PT_BRANDING:HeaderLinkHP(&Portal, &tabindex, &navPortal.IsS⇒
earchInHdr(), &navPortal.GetSearchPageURL(), &navPortal.GetSearchASLimit(), &navPor⇒
tal.IsGblSearchInHdr());

rem b)    Objects of type Application Classes;
&objHLBase.
```


Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Auto-complete list box for application class objects

This example illustrates the auto-complete list box for application class objects.



Note: The auto-complete feature is not supported for objects that are created and assigned to a variable that is not declared explicitly.

If you create an object and assign it to a variable that is not declared in the PeopleCode program, the object type is not known at design time to display the relevant methods or properties. For example, in the following code snippet, variables `&obj1` and `&obj2` are not declared in the PeopleCode program.

,

```
&obj1 = Create PT_BRANDING:BrandingBase(...);
&obj2 = CreateObject("PT_BRANDING:BrandingBase");
```

In cases where the class type changes at run time, the methods or properties of the class type that was declared are displayed. For example, if a variable is declared as of type class A and at run-time the type is changed from type A to type B based on a run-time execution of a piece of PeopleCode, the auto-complete feature displays the methods or properties of the type class A because it was declared as type A.

In the following code snippet, the PeopleCode API `CreateObject` needs to resolve it's parameter to know the Application Package name and the Application Class name. Irrespective of the value of the variable `&name` that is resolved at run-time, the auto-complete support is provided for the class type PT_BRANDING:HeaderLinkBase on the variable `&objHLBase`.

```
Local PT_BRANDING:HeaderLinkBase &objHLBase;
&name = SomeFunction();
&objHLBase = CreateObject(&name);
```

PeopleCode Built-in Functions, System Variables, and Constants Objects

In the PeopleCode editor, press CTRL+ Space bar to view a list of delivered built-in functions, system variables, and constants. The list displays icons for the different items based on the item type. To list the desired set of options, you can enter the desired characters and then press CTRL+ Space bar. For example,

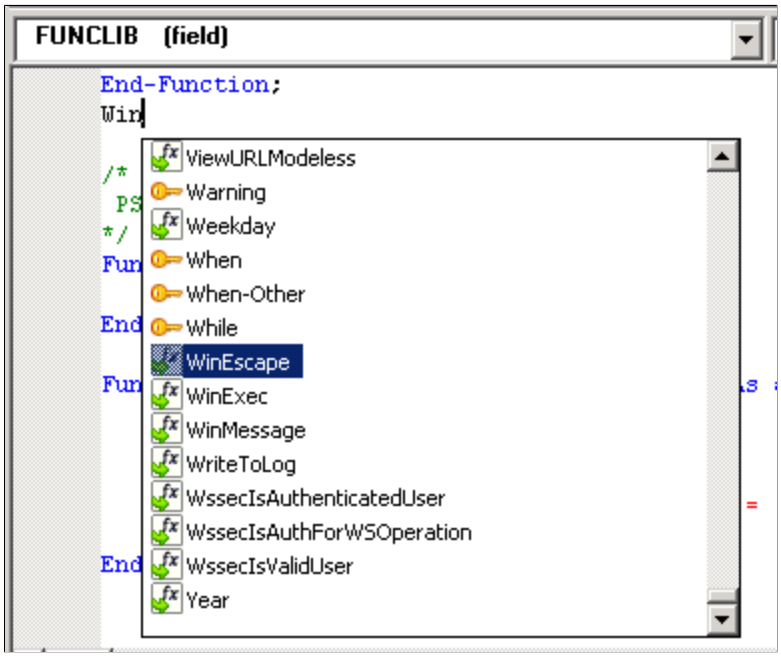
Copyright © 1988, 2021, Oracle and/or its affiliates.

to list all built-in functions, system variables, and constants beginning with “win,” type “win” and then press CTRL+Space bar.

Press the Esc key, to close the list box.

Image: Listing of built-in functions, system variables, and constants

This example illustrates the listing of built-in functions, system variables, and constants in PeopleCode Editor.



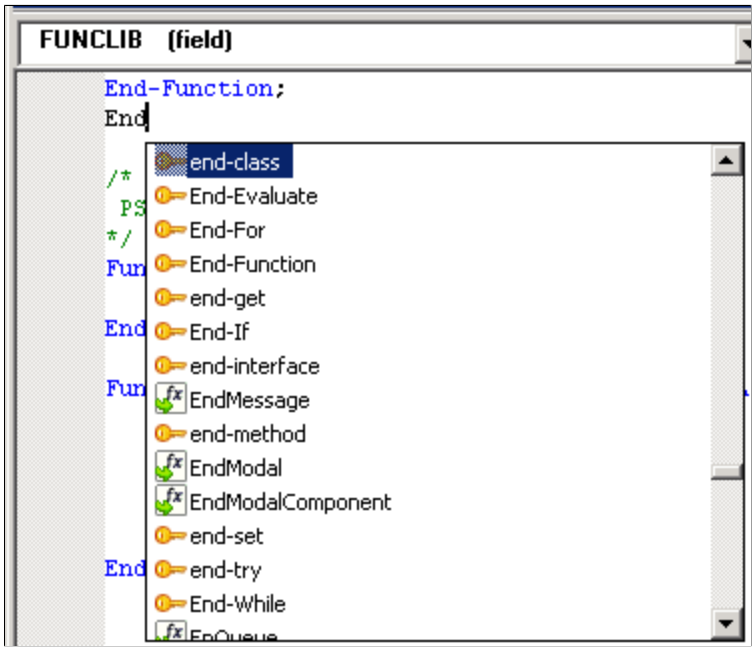
Auto-completion Support for Keywords

In a PeopleCode Editor, pressing CTRL + Spacebar brings up a list control with a list of delivered built-in functions, system variables, and constants. Pressing CTRL + Spacebar also lists keywords of PeopleCode.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Auto-completion Support for Keywords

The following image illustrates the listing of keywords and other items in PeopleCode Editor:



Keywords are shown with a key icon to differentiate them from other items.

Viewing Event Mapping References

The PeopleCode Editor provides an Event Mapping button in the dynamic Application Designer toolbar. For an application developer, the Event Mapping button is a visual indicator of custom PeopleCode programs mapped to events of a component, component page, component record, or component record field.

Using the Event Mapping Button

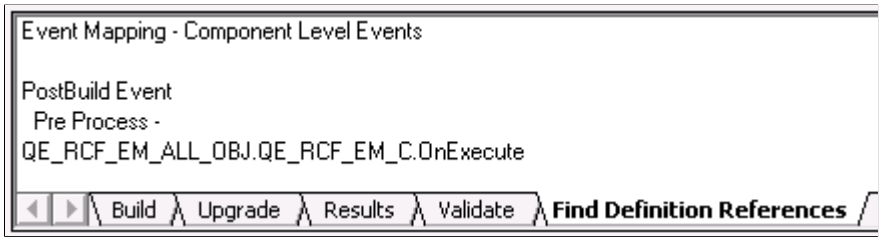
The Event Mapping button in the toolbar is enabled when the PeopleCode being viewed for a component, component page, component record or component record field event are mapped to custom PeopleCode programs. When you select the Event Mapping button, the Find Definition References tab (of the output window) lists the events that are mapped to custom PeopleCode programs along with the application class.

Select the Event Mapping button to view a list of events in the selected component, component record, or component record field.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Find Definition References tab (in output window)

This example illustrates the Find Definition References tab (in output window), which displays the list of events.



Finding Event Mapping References

To find references to other components where the custom PeopleCode programs of an application class are used, you should complete the following steps:

1. In the Find Definition References, right-click the application class.

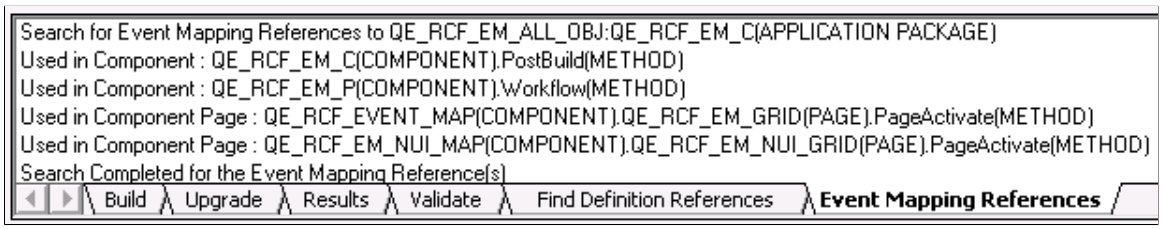
   A pop-menu is displayed.

2. From the pop-up menu, select Event Mapping References.

   The list of components is displayed in the Event Mapping References tab (of the output window).

Image: Event Mapping References (in output window)

This example illustrates the Event Mapping References (in output window), which displays the components that use the application class.



Related Links

"Mapping Application Class PeopleCode to Component Events " (PeopleTools 8.59: Portal Technology)

Generating PeopleCode Using Drag-and-Drop

You can generate references to definitions using a drag-and-drop operation. You can also generate PeopleCode templates for accessing business interlinks and component interfaces.

This section discusses how to:

- Generate references to definitions.
- Generate PeopleCode for a business interlink.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Generate PeopleCode for a component interface.
- Generate PeopleCode for a file layout.

Generating References to Definitions

When you drag definitions, such as menus, records, record fields, and pages, from a project into an open PeopleCode editor window, you generate a reference to the definition. For example, suppose your project contains a component named QEACTIVITY_GUIDE_1. If you drag the QEACTIVITY_GUIDE_1 component definition from the project into an open PeopleCode window, the word QEACTIVITY_GUIDE_1 prefixed with the keyword **COMPONENT** is written to the PeopleCode program in the place where you dragged the definition.

Generating PeopleCode for a Business Interlink

After you create a business interlink definition, you use PeopleCode to instantiate an interlink object and activate the 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 Application Designer Project view into an open PeopleCode edit pane. PeopleCode Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to suit your purpose.

The following is a snippet of the code that is generated:

```
/* ===>

/* ===>
This is a dynamically generated PeopleCode template to be
used only as a helper to the application developer.
You need to replace all references to '<*>' OR default values
with  references to PeopleCode variables and/or a Rec.Fields.*/

/* ===> Declare and instantiate: */
Local Interlink &QE_AE_NONSSL__1;
Local BIDocs &inDoc;
Local BIDocs &outDoc;
Local boolean &RSLT;
Local number  &EXECRSLT;
&QE_AE_NONSSL__1 =  GetInterlink(INTERLINK.QE_AE_NONSSL_BI);
.
.
.
.
```

Generating PeopleCode for a Component Interface

After you create a 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 the Application Designer Project view into an open PeopleCode edit pane. Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to meet your requirements.

The following is a snippet of the code that is generated:

```
/* ===>
This is a dynamically generated PeopleCode template to be
used only as a helper to the application developer.
You need to replace all references to '<*>' OR default values
with references to PeopleCode variables and/or a Rec.Fields. */
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
Local ApiObject &oSession;
Local ApiObject &oCurrencyCdCi;
Local ApiObject &oPSMessageCollection;
Local ApiObject &oPSMessage;
Local File &LogFile;
Local number &i;
Local String &strErrMsgSetNum, &strErrMsgNum, &strErrMsgText,
&strErrType;
.
.
.
```

You can also access a component interface using the component object model (COM). You can automatically generate a Visual Basic template, a Java template, or a C template, similar to the PeopleCode template, to begin.

To generate a template:

1. Open a component interface in Application Designer.
2. Right-click anywhere in the open component interface and select a template type.

   You must save the component interface before generating the template.

   When the template is successfully generated, a message appears with the full path and name of the file containing the template.
3. Open the generated file and modify the source code to meet the needs of your application.

   The following is the initial code snippet that is generated for a Visual Basic template:

```
Option Explicit
'===>
'This is a dynamically generated Visual Basic template to be
'used only as a helper to the application developer.
'You need to replace all references to '<*>' OR default
'values with  references to Visual Basic variables.

Dim oSession As PeopleSoft_PeopleSoft.Session

Private Sub ErrorHandler()
'***** Display PeopleSoft Error Messages *****
If Not oSession Is Nothing Then
   If oSession.ErrorPending Or oSession.WarningPending Then
      Dim oPSMessageCollection As PSMessageCollection
      Dim oPSMessage As PSMessage
      Set oPSMessageCollection = oSession.PSMessages
      Dim i As Integer
      For i = 1 To oPSMessageCollection.Count
         Set oPSMessage = oPSMessageCollection.Item(i)
         Debug.Print "(" & oPSMessage.MessageNumber & "," &
oPSMessage.MessageSetNumber & ") : " & oPSMessage.Text
      Next i
      '***** Done processing messages in the collection;
      '***** OK to delete *****
      oPSMessageCollection.DeleteAll
   End If
End If
End Sub
.
.
.
.
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Generating PeopleCode for a File Layout

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 the PeopleCode Application Designer Project view into an open PeopleCode edit pane. Application Designer analyzes the definition and generates initial PeopleCode as a template, which you can modify to meet your requirements.

This example shows some 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_OneZero);
   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;
      Return False;
   Else
      Return True;
   End-If;
End-Function;
.
.
.
.
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 13

Using the SQL Editor

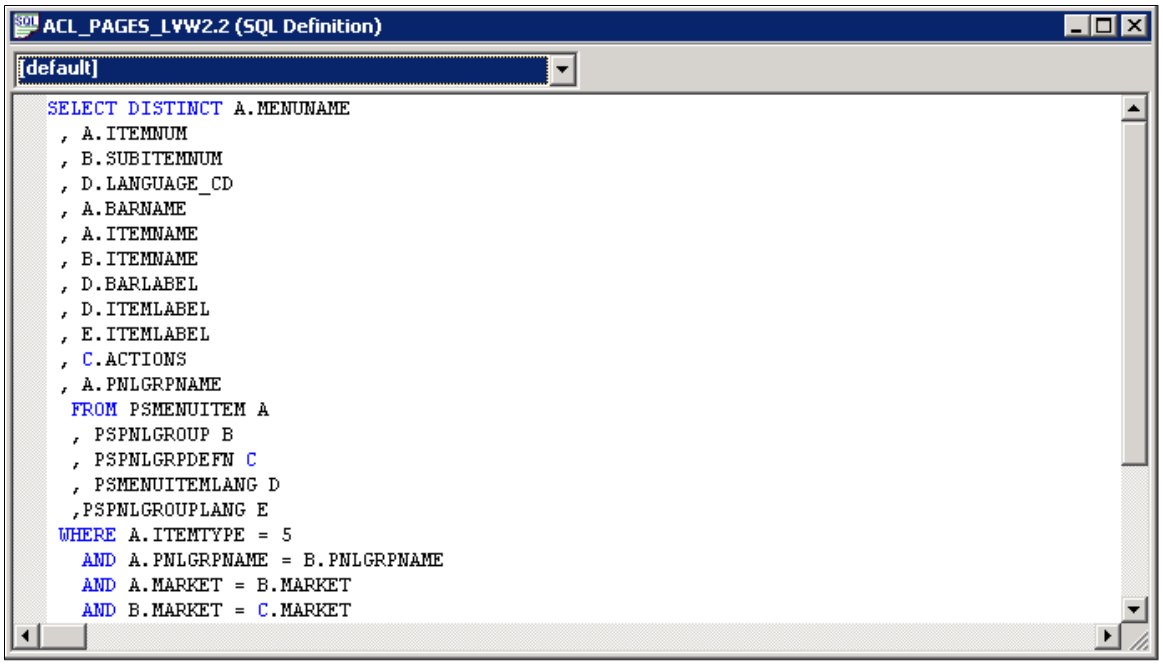
Understanding the SQL Editor Window

Use the SQL Editor to create and validate SQL for SQL definitions for PeopleCode programs, record views, and Application Engine programs.

The title bar of the editor window displays either the name of the SQL definition or the name of the component that contains the SQL. For example, if the SQL statement is part of an Application Engine program, then the names of the program, the section, the step, and the action are listed in the title bar, as shown in the following example:

Image: Example of the SQL Editor window

This example illustrates the SQL Editor window.



The editor window consists of the main edit pane. For SQL definitions and SQL views, a drop-down database list appears at the upper left.

Note: For SQL definitions only, you can optionally specify an effective date, which appears in a drop-down at the upper right.

Accessing the SQL Editor

This section discusses how to:

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Create SQL definitions.
- Access SQL definition properties.
- Create dynamic view or SQL view records.
- Access the SQL Editor from Application Engine programs.

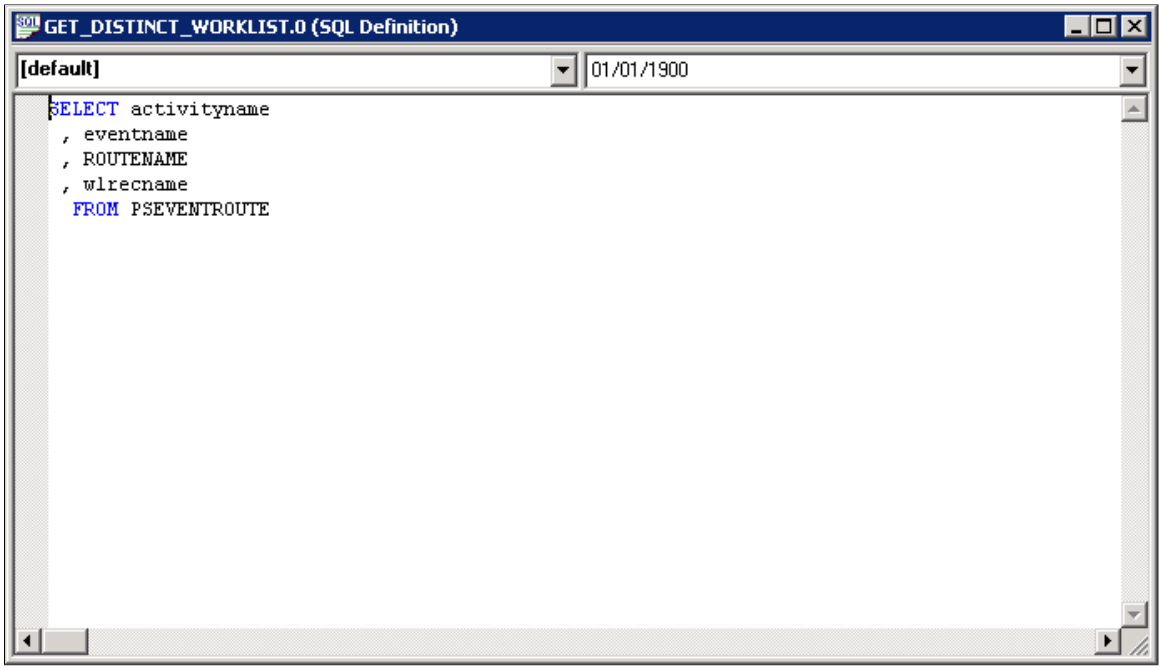
You access the SQL Editor differently for each definition type.

Creating SQL Definitions

A SQL definition contains SQL statements, which can be entire SQL programs or just fragments that you want to reuse. You can access, create, change, or delete SQL definitions using Application Designer, or you can use the SQL class in PeopleCode. You can upgrade SQL definitions, and you can add them to a project. The following example shows a SQL definition:

Image: Example of a SQL definition with an effective date

This example illustrates a SQL definition with an effective date.



To create a SQL definition:

1. From Application Designer, select File, New, SQL.
2. Specify the database type to associate with the SQL definition.

   You can associate more than one database type with a single SQL definition. In PeopleCode, you can specify the appropriate database type for the program. However, at least one of the SQL statements must be of type Default.

3. (Optional) Specify an effective date.

   To specify an effective date with your SQL definition:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

a. Access the object properties by selecting File, Object Properties.

   Alternatively, select the SQL definition, right-click it, and then select Object Properties, or press ALT + ENTER.

b. Click the Advanced tab, and then select the Show Effective Date check box.

   When you click OK, the SQL definition shows a date in the right-hand drop-down menu.

4. Enter the SQL code.

   You do not need to format your code. The SQL Editor validates and formats it when you save the SQL definition.

Related Links

"Working with Definitions" (PeopleTools 8.59: Application Designer Developer's Guide)
"Understanding SQL Class" (PeopleTools 8.59: PeopleCode API Reference)

Accessing SQL Definition Properties

Do one of the following to access the definition properties for the SQL definition:

- Press ALT+Enter.
- Select File, Definition Properties.
- Right-click in the definition and select Definition Properties.

Use general properties to specify a description for the SQL definition as well as additional comments. The description appears in Application Designer search lists.

Use the advanced properties to display an effective date with the SQL definition.

Note: The Audit SQL field on the Advanced Properties tab is not used.

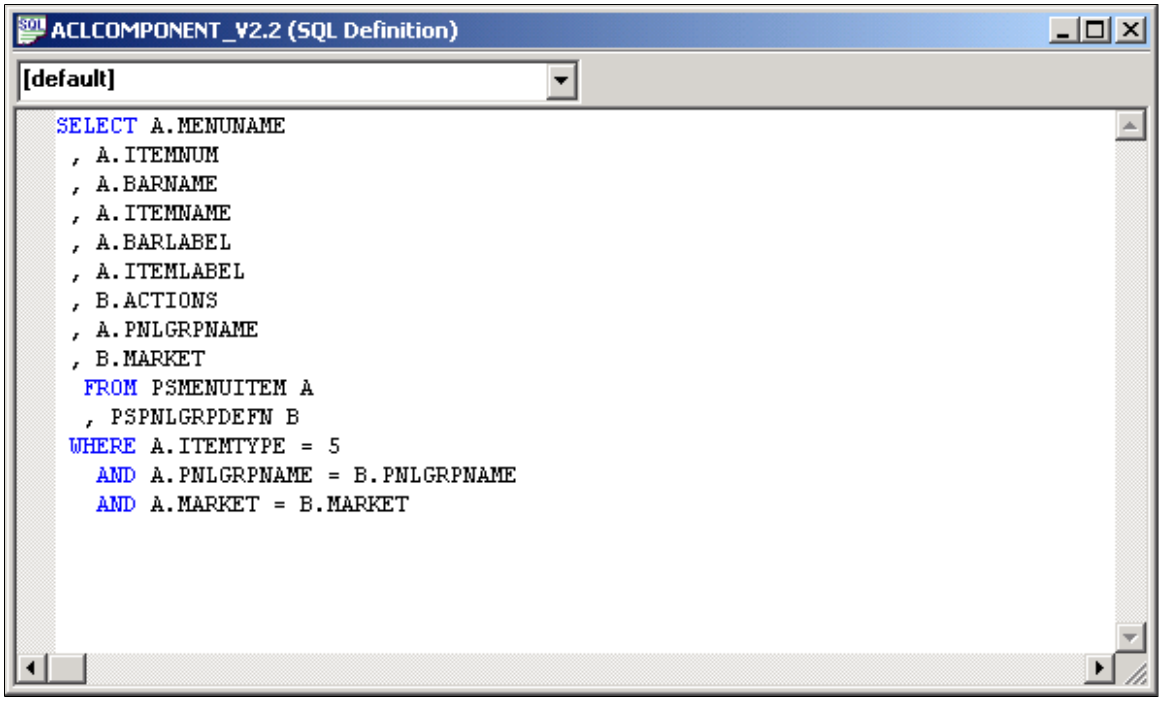
Creating Dynamic View or SQL View Records

When you create a SQL view or dynamic view record definition, you enter a SQL view *Select* statement to indicate the field values that you want to join and the tables that contain the field values. You do this in the SQL Editor, as shown in the following example:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: SQL Editor for SQL view record definition

This example illustrates the SQL Editor for SQL view record definition.



To access the SQL Editor from record definitions:

1. Open or create a dynamic view or SQL view record definition.
2. Select the Record Type tab.
3. Click the Click to Open SQL Editor button.

   You can select a database type, but not an effective date, from the SQL Editor for dynamic view and SQL view record definitions.

Note: You must be sure to save record definitions of the SQL View type before opening the SQL Editor. Once the SQL Editor is open, the Save options are disabled and inaccessible. If you do not save your changes before opening the SQL Editor, you may lose your work.

Accessing the SQL Editor from Application Engine Programs

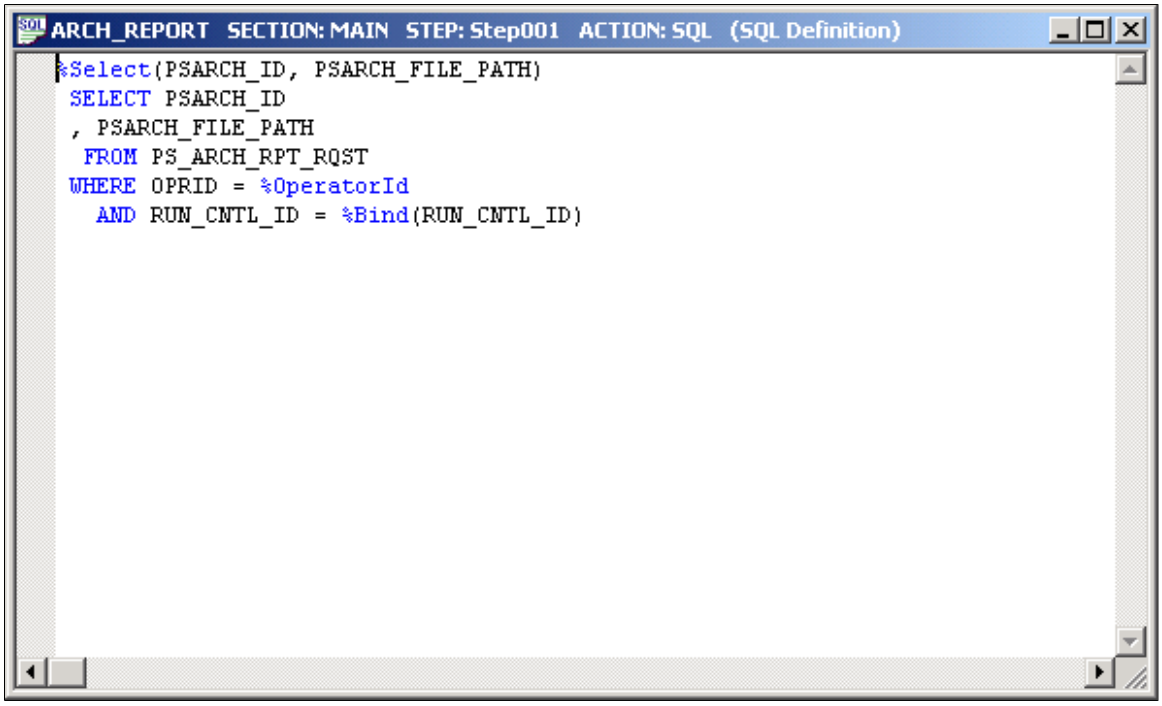
You can access the SQL Editor from the following action types:

- Do Select
- Do Until
- Do When
- Do While
- SQL

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Example of Application Engine SQL code in the SQL Editor window

The following example shows an Application Engine SQL code in the SQL Editor:



To access the SQL Editor from an Application Engine program:

1. Open the Application Engine program.
2. Select the action.
3. Either right-click and select View SQL, or select View, SQL.

   Select the database type and effective date for this SQL in the section, not in the SQL Editor.

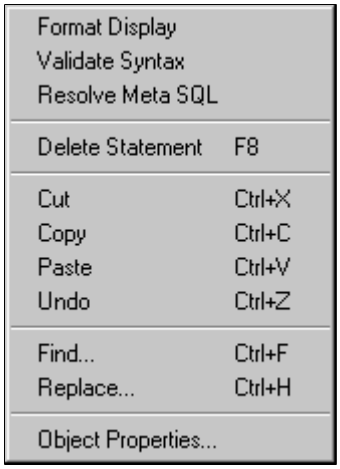
Using the SQL Editor

The SQL Editor works similarly to many other text editors. Many of the same functions are available as in the PeopleCode Editor, for example, cut, paste, find, and replace.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: SQL Editor shortcut menu

When you right-click in an open SQL Editor window, the available functions appear in a pop-up menu:



This table describes the functions that are available in the SQL Editor but not the PeopleCode Editor:

Function	***Description***
Format Display	You do not need to format your SQL statements; you only need to use the correct syntax. When you save or validate, the system formats the code according to the rules in the PeopleCode tables, no matter how you entered it originally. It automatically converts field names to uppercase and indents statements. The resulting look of SQL is consistent with other programs in the system.
Resolve Meta SQL	If the SQL contains meta-SQL, select Resolve Meta SQL to expand the meta-SQL statement in the output window. The expanded meta-SQL appears in the Meta SQL tab.
Delete Statement	You can delete standalone SQL statements. This menu item is not enabled with SQL statements that have a database type of Default with no effective date, or for statements that have a database type of Default and an effective date of 01/01/1900.

This example, using Resolve Meta-SQL, shows how the following code expands:

```
%Join(COMMON_FIELDS, PSAEAPPLDEFN ABC, PSAESECTDEFN XYZ)
```

Image: Meta-SQL expanded in the output window

This example illustrates meta-SQL expanded in the output window.

The SQL Editor and the PeopleCode Editor interfaces and functions are similar. You can add, delete, and change text; you can use the find and replace function; and you can validate the SQL. When you save a SQL definition, the code is automatically validated and formatted (indented, capitalized, and so on). You

 Copyright © 1988, 2021, Oracle and/or its affiliates.

can select the colors for displaying keywords, comments, operators, and so on; these settings are shared with the PeopleCode Editor. You can also separately specify word wrap options.

See the documentation on the PeopleCode Editor for details on relevant functions.

Related Links

Editing Functions
Find and Replace Dialogs
Validate Syntax Utility
Using Drag-and-Drop Editing
Changing Colors in the PeopleCode Editor
Selecting a Font for the PeopleCode Editor
Changing Word Wrap in the PeopleCode Editor

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 14

Creating Application Packages and Classes

Understanding Application Packages

Use the Application Packages Editor to create application packages. A package contains other packages or application classes. A subpackage is any package within a primary, or parent, package.

The title bar of the editor window displays the name of the application package definition. The main window displays the classes and other application packages that make up the application package definition.

The application package hierarchy is displayed as a tree structure. You can use the expand icon (+) and the collapse icon (-) to expand or collapse individual nodes.

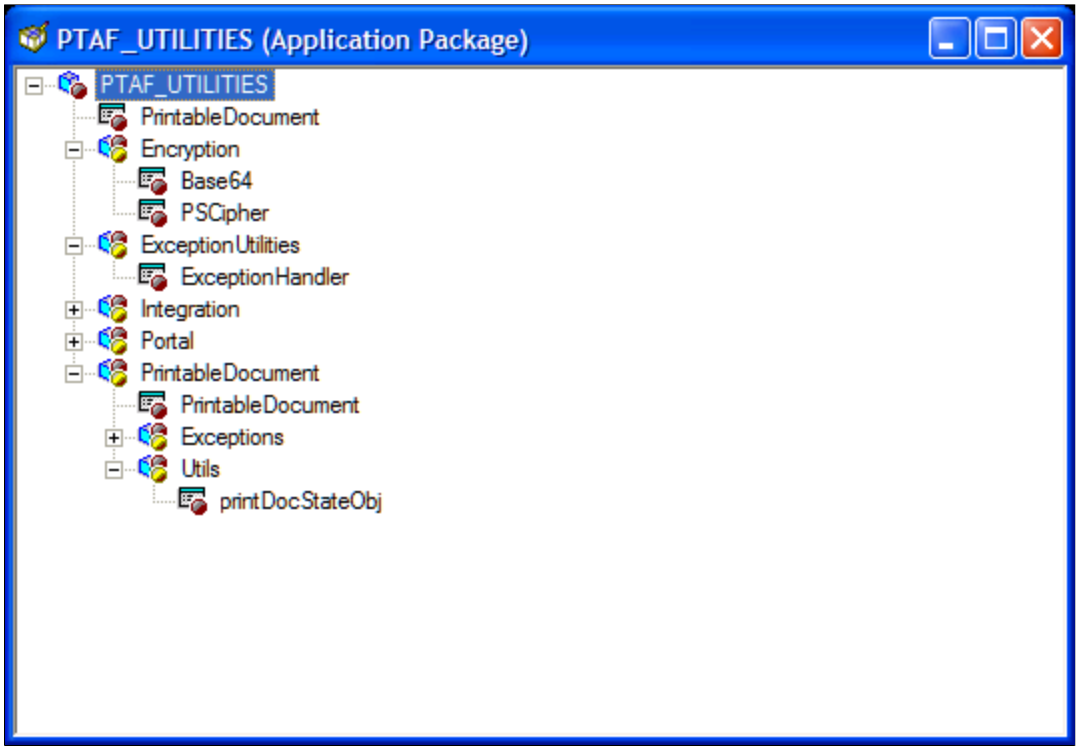
To expand all the nodes in a package select View, Expand All or click the Expand All Nodes button on the toolbar. To collapse all the nodes in a package select View, Collapse All or click the Collapse All Nodes button on the toolbar.

In the following example, PTAF_UTILITIES is the primary package, and Encryption, Exception Utilities, Integration, and so on, are subpackages. Printable Document is a class in the PTAF_UTILITIES application package, while Base64 and PSCipher are classes in the Encryption subpackage.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Application Package Editor main window

This example illustrates the fields and controls on the Application Package Editor main window. You can find definitions for the fields and controls later on this page.



Related Links

"Understanding Application Classes" (PeopleTools 8.59: PeopleCode API Reference)

Creating Application Packages

This section provides an overview of package names and discusses how to create application package definitions.

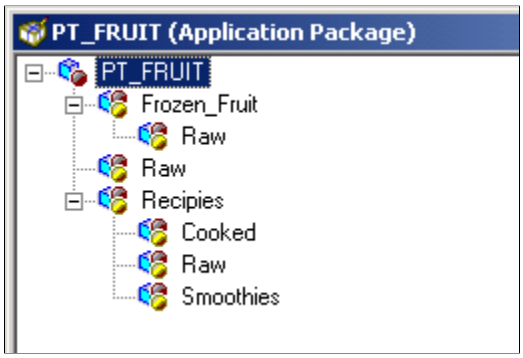
Understanding Package Names

You can create a subpackage with the same name as another package or subpackage within the same application package definition, as long as the fully qualified name is unique for each subpackage. Each subpackage is differentiated by the full path name of the class (from the package definition name and the subpackage name).

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Example of application package naming conventions

The following is an example of application package naming conventions, which shows a case where in, suppose in the application class PT_FRUIT, where PT_FRUIT is the primary class, you had the following structure of subpackages (no classes are listed in this example):



In this example, three subpackages are named Raw, but the fully qualified name for each is unique. For example, the first one is qualified by the name of the primary package. Its fully qualified name is PT_FRUIT:Raw.

The other Raw subpackages are also qualified by the subpackages that contain them. Their names are PT_FRUIT:Reciepies:Raw and PT_FRUIT:Smoothies.Raw.

Similarly, you cannot create two classes with the same name within a given package or subpackage. You can create classes with the same name within the same application package definition, just like subpackages, as long as the fully qualified name is unique. Each class is differentiated by the full path name of the class.

Note: You cannot create a structure for which more than two levels of subpackages are defined below the primary package.

Creating Application Package Definitions

This section discusses how to create a new application package or insert a new package or class into an application package.

To create a new application package, access Application Designer and select File, New, Application Package.

To insert a new package or class, open an application package definition. Select a package or subpackage and select Insert, Package or Insert, Application Class.

Application package names and application class names must begin with an alphabetic character and must consist of only alphanumeric characters and underscores (_).

Note: In certain cases you may encounter an error if you use Save As to create a new application package. The error occurs when you choose to save the PeopleCode with the application package and the PeopleCode uses a %This system variable. The %This system variable is used in application class PeopleCode to refer to the current object.

Copyright © 1988, 2021, Oracle and/or its affiliates.

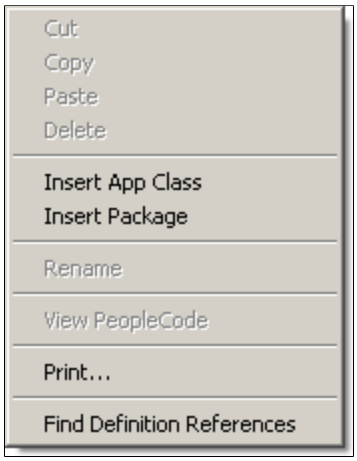
Related Links

"Using %This with Constructors" (PeopleTools 8.59: PeopleCode API Reference)

Using the Application Package Editor

Image: Application Package Editor pop-up menu

When you right-click an open Application Package Editor window, you see the available functions:



Cut, Copy, and **Paste**	Not available for this release. Instead, insert new subpackage and class nodes where needed and use the clipboard to copy and paste PeopleCode text from class to class. To copy the primary package, select File, Save As.
Delete	Click to delete either a class or a package. The PeopleCode text is not actually deleted until you save the application package. Deleted PeopleCode classes can be recovered by reinserting the class node, as long as you have not saved in the interim.
Insert App Class (insert application class)application classes inserting	Click to insert an application class. Because classes cannot have children (subclasses), they can be inserted only into an existing package.
Insert Package	Click to insert an application package. You can only insert packages into an existing package or subpackage.
Rename	Click to rename either a class or a subpackage. When you save the definition, all PeopleCode programs associated with the renamed class are also updated. To rename the primary package definition, select File, Rename.
View PeopleCode	Click to view the associated PeopleCode. PeopleCode can be defined only for application classes, and it is not directly related to package nodes.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Print	Click to print the application package definition, including all the PeopleCode in the classes.
Find Definition References	Select an application package or class and click to search for references. See Finding References to Application Packages and Classes.

Editing Application Classes

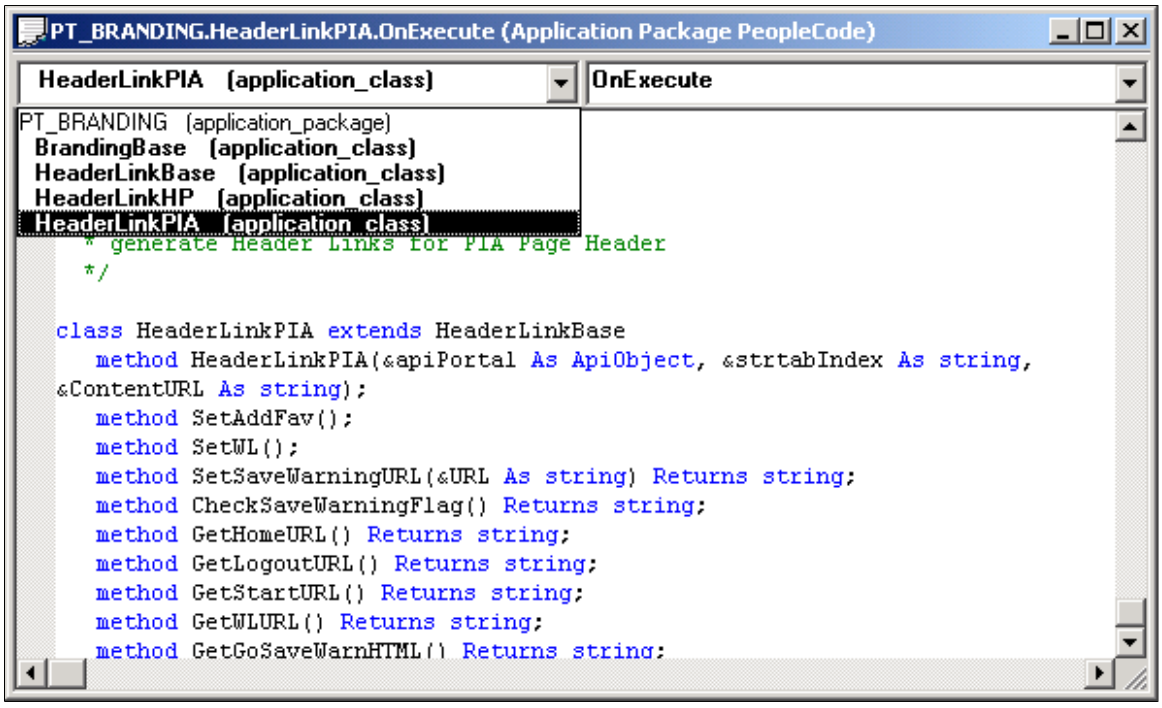
From an application package, you can access the PeopleCode programs associated with the classes of the package.

The Application Packages Editor and the PeopleCode Editor interfaces are similar. You can add, delete, and change text, you can use the find and replace functions, and you can validate syntax. When you save application packages, the code is automatically formatted (indented and so on), just as it is in the PeopleCode Editor.

The editor window contains the main edit pane, the drop-down definition list at the upper-left, and the drop-down event list at the upper-right, as shown in the following example:

Image: Example of Application Packages Editor window

This example illustrates the fields and controls on the Example of Application Packages Editor window. You can find definitions for the fields and controls later on this page.



Only one event is defined for an application class, OnExecute. This is not an event in the Component Processor flow. The application class runs when called.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The drop-down list at the upper-left enables you to navigate directly to the PeopleCode associated with every class in the package, as well as to every subpackage and its classes.

To edit an application class:

1. Open the application package.
2. Select a class.
3. Either select View, PeopleCode or right-click and select View PeopleCode.

   A PeopleCode Editor window appears.

Related Links

Editing Functions

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 15

Debugging Your Application

Understanding the PeopleCode Debugger

The PeopleCode debugger is an integrated part of Application Designer. The interface to the debugger has a visual indicator of breakpoints, an arrow indicating the current line, and the ability to step through code. You can inspect the value of a variable by holding the cursor over it and reading the pop-up bubble help. The debugger also provides variable inspection windows for global variables, local variables, function parameters, and component-scoped variables. It also enables PeopleCode objects to be expanded, so you can inspect their component parts.

Note: Do not try to use the PeopleCode debugger with the SwitchUser function. Only the first user is logged into the PeopleCode debugger. Once the switch occurs, breakpoints, logging, and so on are no longer executed.

Accessing the PeopleCode Debugger

Note: You can start a debugging session either before or after you start a PeopleSoft component.

1. Determine whether to run Application Designer in two-tier mode or three-tier mode.

   - If you are debugging Application Engine or component interface PeopleCode, run Application Designer in two-tier mode, with a direct connection to the database.
   - If you are debugging an application in PeopleSoft Pure Internet Architecture (PIA), run Application Designer in three-tier mode, through the application server. You must be logged on to PIA and to Application Designer using the same user ID.

2. Access the debugger through Application Designer by selecting Debug, PeopleCode Debugger Mode.

   The Local Variables watch pane and the Call Stack pane open. PeopleCode programs that had breakpoints set from your previous debugging session are opened also, and the breakpoints are restored.

   If you did not have breakpoints set, open the PeopleCode program you want to debug and enter debug mode. The debugger will open with the current PeopleCode program and you can set your breakpoints.

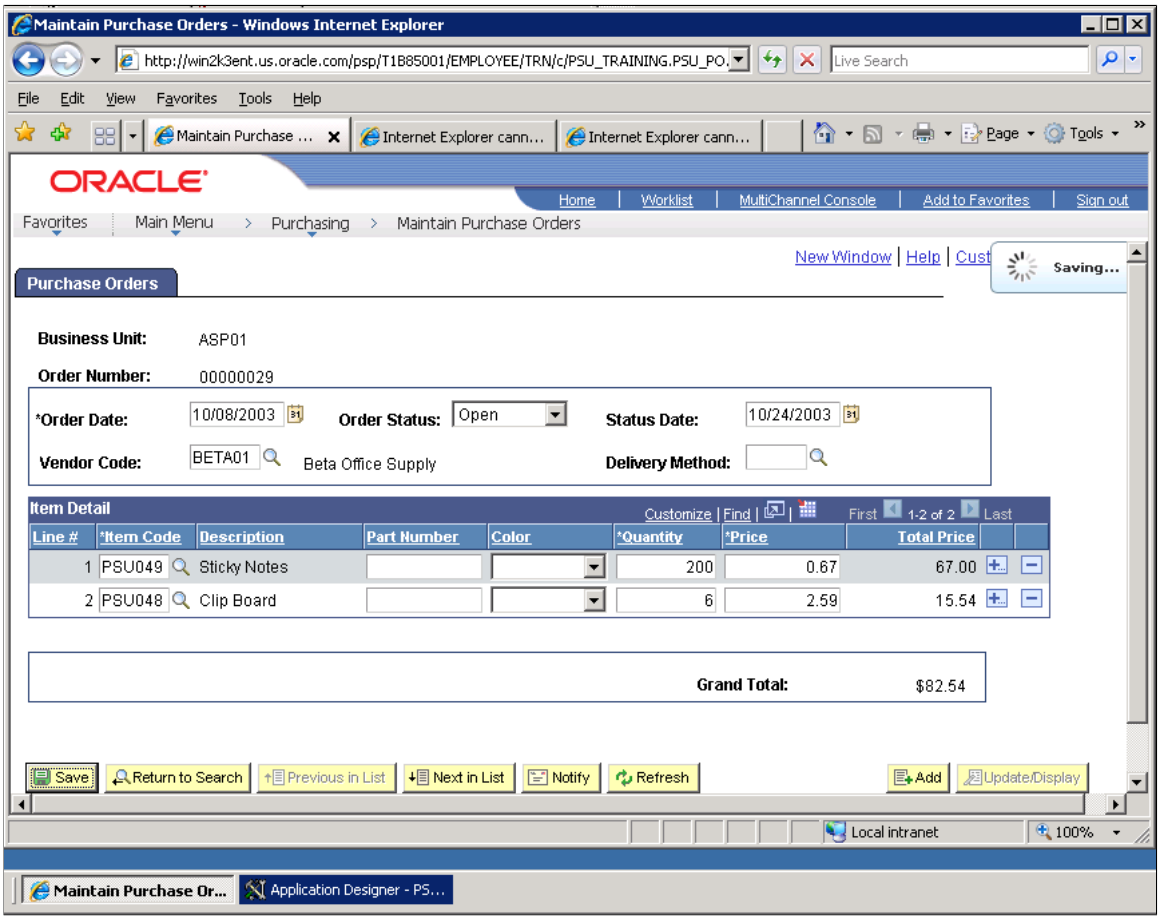
   Note: If you have already opened the debugger and then closed it, the menu may not change correctly to enable you to access the debugger a second time. If this occurs, click the Local Variables window, and then try the Debug menu again.

Copyright © 1988, 2021, Oracle and/or its affiliates.

In PIA, navigate to the point where the breakpoint occurs. Your application pauses and the Application Designer icon flashes in the task bar. Switch to Application Designer to step through your program or continue running it.

Image: Application Designer icon flashes in the toolbar when the application hits a breakpoint

This example illustrates the fields and controls on the Application Designer icon flashes in the toolbar when the application hits a breakpoint. You can find definitions for the fields and controls later on this page.



If the debugger does not engage, check that you used the same user ID to log into PIA and Application Designer, then check your application server configuration to verify that PeopleCode debugger is enabled.

Note: Your security administrator has options for allowing users to access different parts of Application Designer, including the PeopleCode debugger. If you are having problems accessing the debugger, you may need to contact your system administrator about your security access. You can access the PeopleCode debugger from outside a firewall.

Using PeopleCode Debugger Features

This section discusses:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

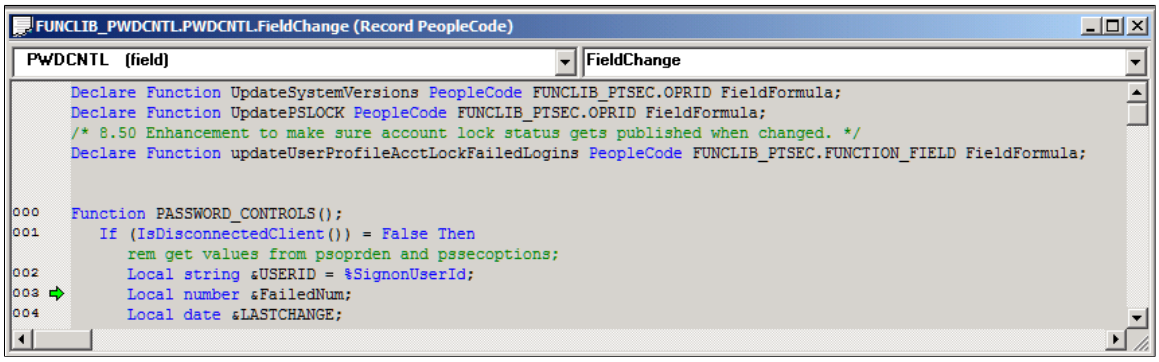
- Visible current line of execution.
- Visible breakpoints.
- Hover inspect.
- Single debugger.
- Variable panes.
- General debugging tips.

Visible Current Line of Execution

This example shows the current line indicator (green arrow displayed in left-hand gutter):

Image: PeopleCode debugger with current line of execution

This example illustrates the fields and controls on the PeopleCode debugger with current line of execution. You can find definitions for the fields and controls later on this page.



Visible Breakpoints

The PeopleCode debugger supports visual indicators that signify breakpoint locations. In the following example, the current line indicator (green arrow) is shown at the first line, and the breakpoint (red dot displayed in left-hand gutter) is on line 8:

All breakpoints are saved when Exit Debug Mode is selected.

Note: You cannot set breakpoints in function declarations, variable declarations, or comments.

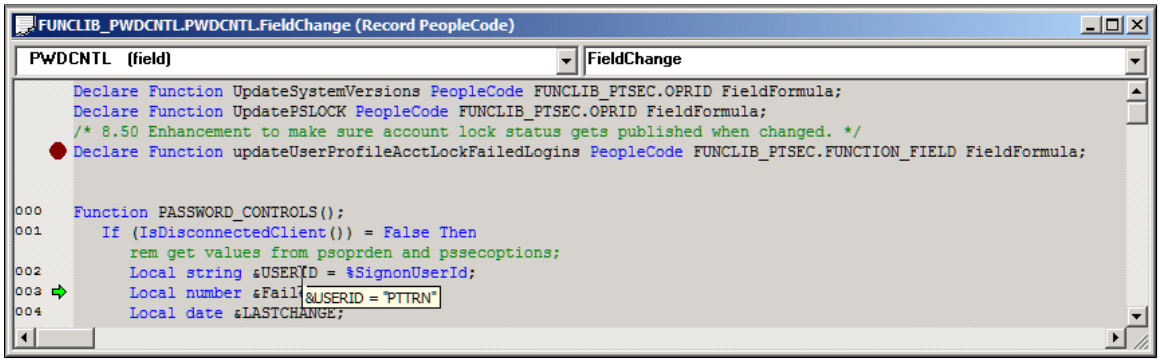
Hover Inspect

If the program is already running, you can see the actual values for the variables by holding the cursor over them. The current value appears in a pop-up window, as shown in the following example:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: PeopleCode debugger with breakpoint, current line of execution, and hover inspect

This example illustrates the fields and controls on the PeopleCode debugger with breakpoint, current line of execution, and hover inspect. You can find definitions for the fields and controls later on this page.



Hover inspect is implemented only for simple variables and fields.

Hover inspect is not implemented for object expressions (for example, rowset assignments and array assignments).

Single Debugger

Each PeopleSoft session you run on a machine can have its own debugging session. However, only one instance of the PeopleCode debugger can occur per session. If more than one instance of Application Designer is running for a session, only one may be the active debugger at a given time.

From within a running instance of Application Designer, any component in the same session is also placed into debug mode.

After the session is in debug mode, any component that is started and that belongs to that session automatically goes into debug mode.

Similarly, Application Engine PeopleCode and component interface PeopleCode can be debugged.

After you exit debug mode by selecting Debug, Exit Debug Mode or by exiting Application Designer, all components in that session go out of debug mode. If you exit a component, debugging continues with any remaining open and running components.

If more than one Application Designer session is running, the Application Designer session that is used as a debugger is the first one to be started.

In debug mode, a PeopleCode Editor window opens for every item (for example, record, component, or page) that has PeopleCode in it when that PeopleCode is executed. If a component has more than one event with a PeopleCode program, then only one window opens per item. For example, if you have a record that has PeopleCode in both the SearchSave and RowInit events, only one PeopleCode Editor window opens: first it contains the SearchSave PeopleCode program, and then the RowInit program. If you have PeopleCode in the RowInit event for two different records that are part of the same component, two PeopleCode Editor windows open, one for each RowInit PeopleCode program.

Variables Panes

The four types of variables panes are:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

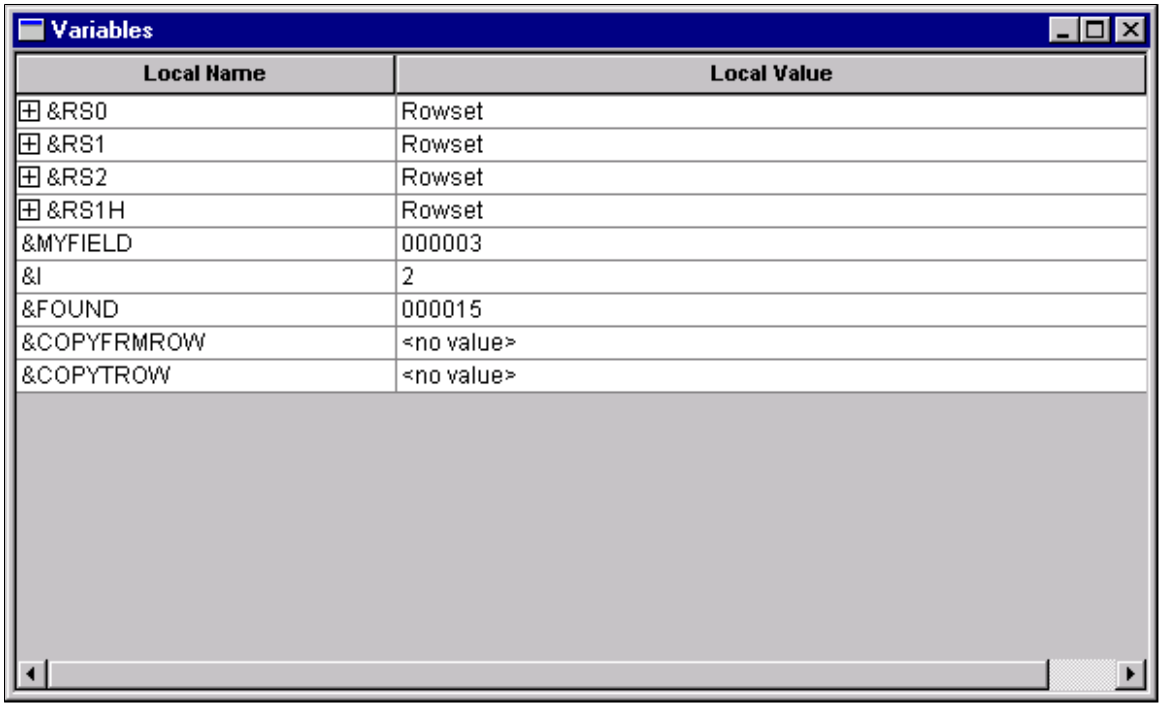
- Local
- Global
- Component
- Parameter

The Local, Global, and Component variable panes show local, global, and component variables, respectively. The Parameter variable pane shows the value of parameters passed in function declarations.

From the variables pane, you can check the value of the variables you have in the program. These values are updated as the code runs. The following example shows the variables pane:

Image: Local Variables pane

This example illustrates the fields and controls on the Local Variables pane. You can find definitions for the fields and controls later on this page.

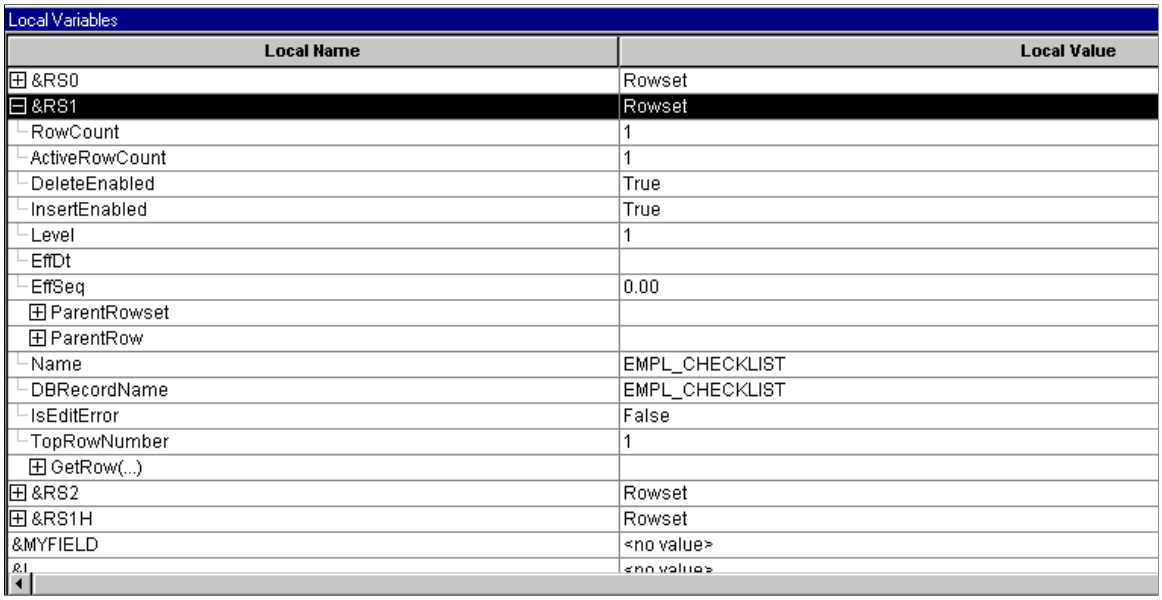


In addition, you can expand any of the objects to see its properties by clicking the plus sign next to the variable name. In the following example, a level one rowset is expanded. You can see the properties, such as ActiveRowCount and DBRecordName, that are part of the rowset.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Local Variables pane with rowset object expanded

This example illustrates the fields and controls on the Local Variables pane with rowset object expanded. You can find definitions for the fields and controls later on this page.


Local Variables

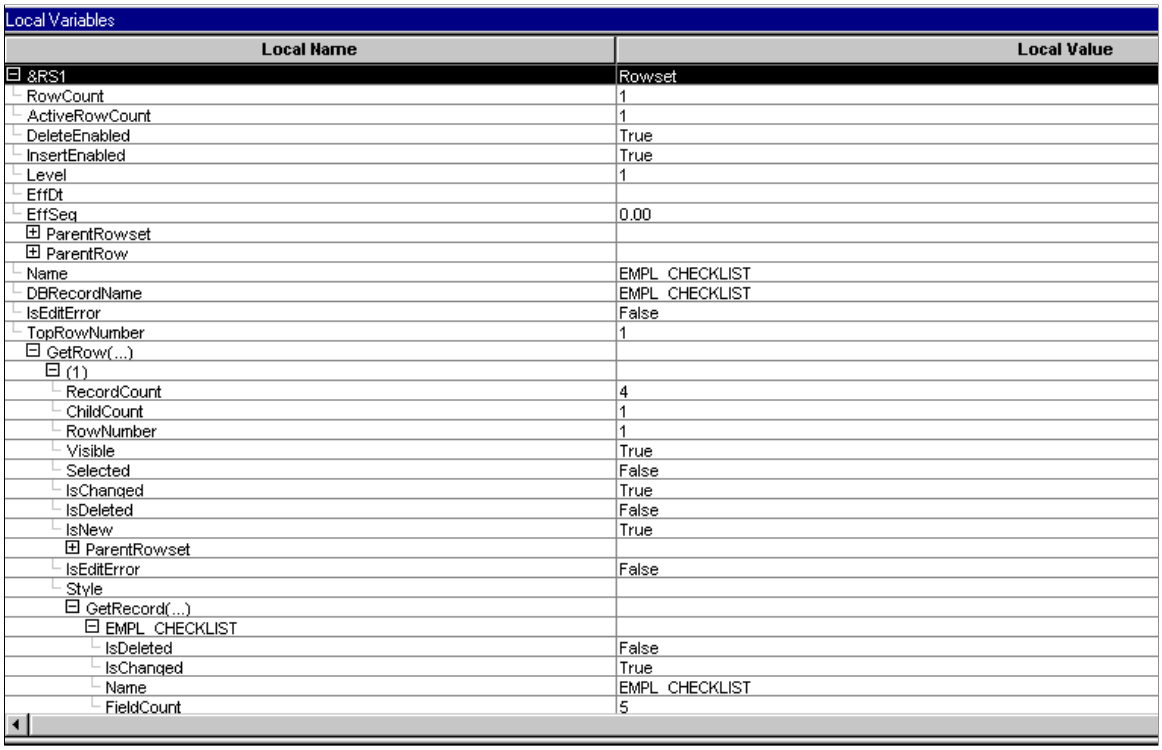
Local Name	Local Value
⊞ &RS0	Rowset
⊟ &RS1	Rowset
RowCount	1
ActiveRowCount	1
DeleteEnabled	True
InsertEnabled	True
Level	1
EffDt	
EffSeq	0.00
⊞ ParentRowset	
⊞ ParentRow	
Name	EMPL_CHECKLIST
DBRecordName	EMPL_CHECKLIST
IsEditError	False
TopRowNumber	1
⊞ GetRow(...)	
⊞ &RS2	Rowset
⊞ &RS1H	Rowset
&MYFIELD	<no value>
&I	<no value>

In addition, some objects contain other objects: a rowset contains rows, rows contain records or child rowsets, and records contain fields. You can expand these secondary objects to see their properties. In the following example, the first row of a rowset is expanded, as is the EMPL_CHECKLIST record:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Variable pane with rowset, row, and record expanded (shown with condensed font)

This example illustrates the fields and controls on the Variable pane with rowset, row, and record expanded (shown with condensed font). You can find definitions for the fields and controls later on this page.



Field Values

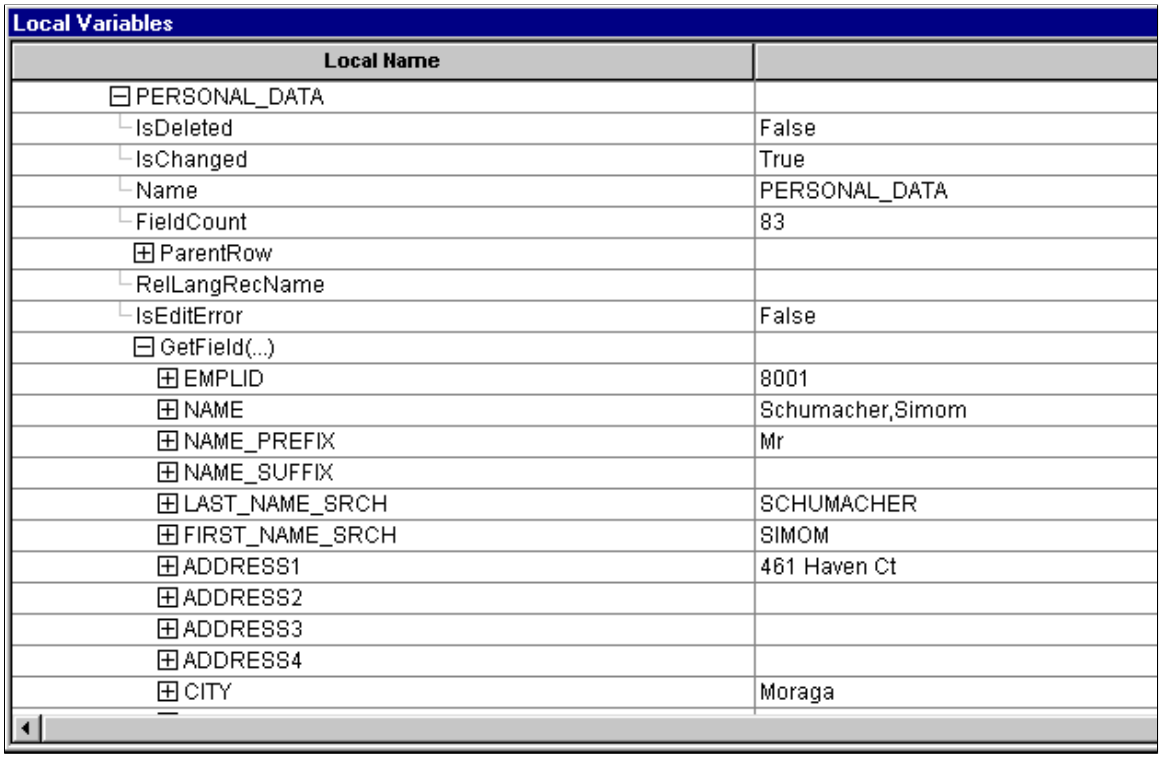
When you view a field object in the debugger, the value of the field is listed in the Value column. Therefore, you do not have to navigate to the Value property to see the value of a field.

The following example shows the PERSONAL_DATA record and the values of the fields:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: PERSONAL_DATA record field values

This example illustrates the fields and controls on the PERSONAL_DATA record field values. You can find definitions for the fields and controls later on this page.



In addition, the only fields that appear in the debugger are the fields that are actually in the Component Buffer. For example, suppose you have a derived work record, but you do not access all the fields in the work record. Only the fields that you access and that are in the Component Buffer actually appear in the debugger.

Related Links

Record Fields and the Component Buffer

Call Stack Pane

The Call Stack pane appears by default when the PeopleCode debugger is started. To reopen it, select Debug, View Call Stack Window.

The Call Stack pane displays a stack of PeopleCode functions and methods that are currently active but not completed. You can use the Call Stack pane to observe the flow of an application as it executes a series of nested functions. When a function is called, it is pushed onto the top of the stack. When the function returns, it is popped off the stack.

The Call Stack pane displays the currently executing function at the top of the stack and older function calls below that, in reverse calling order. You can navigate to the source code of a function from the call stack window. The variables panes update to reflect values for the selected function.

The Call Stack pane is updated and usable when execution is stopped at a breakpoint.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Call Stack Indicators

The Call Stack displays a current line indicator and a selected function indicator in the gutter.

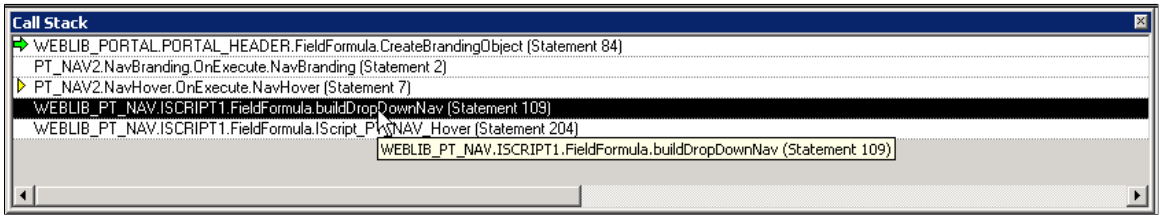
The current line indicator is a green arrow that shows where in the call stack the execution stopped. The current line indicator always appears in the call stack pane.

The selected function indicator is a yellow triangle that marks the current function being displayed by the Edit and Variables windows. The selected function indicator does not appear when the current function is at the execution point.

You can hover over a function name to see the full program path in a pop-up window, as shown in the following example.

Image: Call Stack pane example showing execution pointer, selected function indicator, and hover pop-up

This example illustrates the fields and controls on the Call Stack pane example showing execution pointer, selected function indicator, and hover pop-up. You can find definitions for the fields and controls later on this page.



The Call Stack pane is updated, when necessary, with each change of the debug state.

Go To Source Code

Right-click on a function to access a context menu with these options:

Options	***Description***
Copy	Copies the text of the selection in the call stack to the clipboard.
Select All	Selects all rows in the call stack. You can also use standard shift-click and CTRL-click actions to select multiple rows.
Go To Source Code	Displays the selected function in the Edit window. In addition, the active variables windows will be updated in sync with the Call Stack and Edit windows. Double-click a function name to go to the source code.

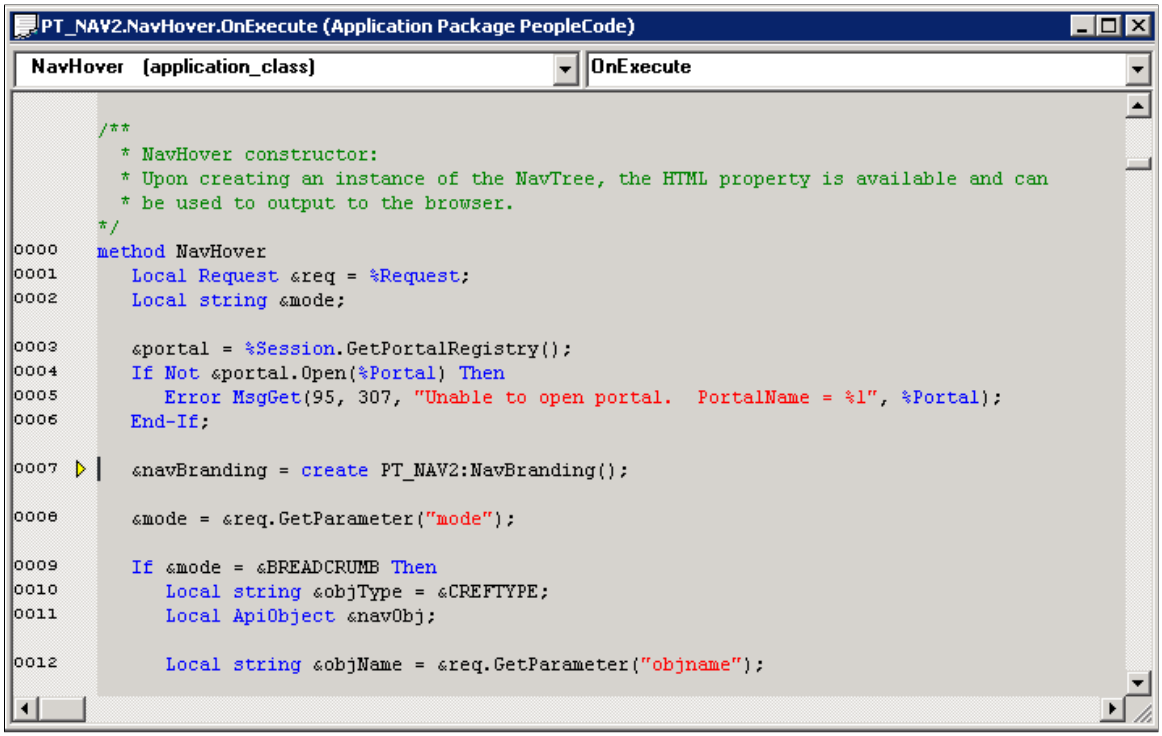
When the displayed source code is at the execution point, the execution pointer icon (green arrow) appears in the source window.

When the displayed source code is not at the execution point, the selected function icon (yellow triangle) appears in the source window.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Source code pane and call stack pane showing the yellow triangle execution pointer

This example illustrates the fields and controls on the Source code pane and call stack pane showing the yellow triangle execution pointer. You can find definitions for the fields and controls later on this page.



Viewing source code for functions that are not at the top of the stack does not change the point of execution. Nor does it disable the ability to continue execution. For example, in the previous example, selecting Go would cause the program execution to continue at "FUNCLIB_PORTAL.PORTAL_GEN_FUNC.FieldFormula PortalOpen" and not at the function in the source code window.

During debugging it is easier to go back to the previous code event from the call stack window as compared to trying to keep track of where the control was transferred and then finding the right opened PeopleCode window in Application Designer. This can be useful when trying to understand the component design and PeopleCode flow.

Setting Values for Variables and Properties

Setting the value of a variable or property in the debugger gives you the flexibility to try out changes and see the results in real time or to recover from a logic error and continue.

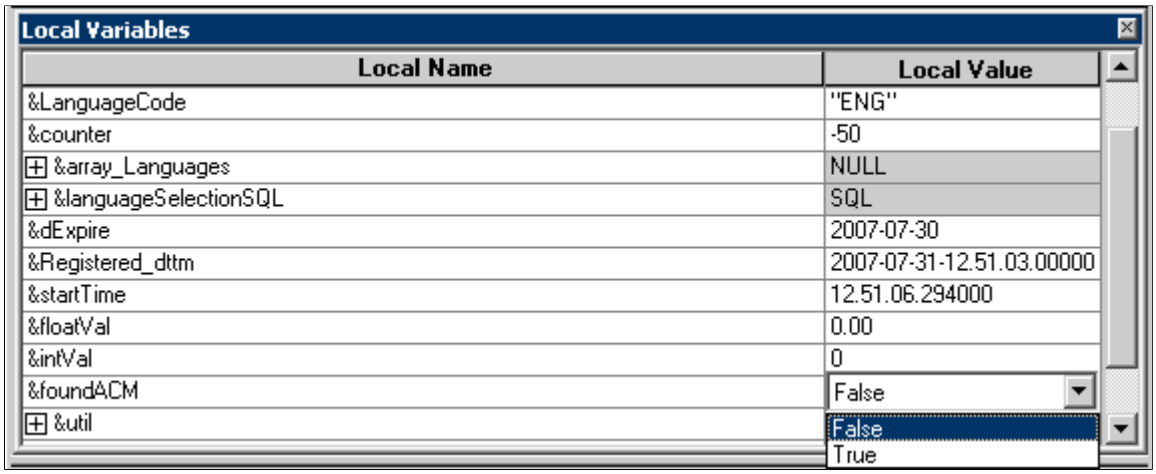
You can change the value of variables or properties in the Local Variables, Component Variables, Function Parameters, and Component Buffers panes. Only variables or properties with conventional data types (Any, Boolean, Date, DateTime, Float, Integer, Number, String, Time) can by changed. You cannot assign a new object to an object variable. Values that are not editable appear on a gray background.

While the debugger is running and halted at a breakpoint, select a field in the value column, such as the Local Value column in the example, and revise the value.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Local Variables pane showing a drop-down list to set the value for a Boolean variable

This example illustrates the fields and controls on the Local Variables pane showing a drop-down list to set the value for a Boolean variable. You can find definitions for the fields and controls later on this page.



The debugger performs data type checking to prevent entry of incorrect data type values. For example, character strings are not allowed for integer data types, and so on. However, data integrity is not verified, so be aware that changing variable values at runtime can corrupt program execution as well as program data. For example, setting an integer value higher than what is permitted in the function could cause a crash when execution continues. It is the developer's responsibility to enter an appropriate value.

Modifying a variable in a debugger pane changes the value in memory only. The change does not trigger any PeopleCode events and does not cause any PeopleCode flags to be set.

General Debugging Tips

The following are general tips for debugging your application:

- If you are having problems determining if the correct data is being loaded into the component buffers, use the View Component Buffers view window to see all the values currently in the component buffer.

  This is equivalent to putting a GetLevel0 function at the start of a program.

  Use the &LEVEL0 variable to navigate through all the levels of the rowset object, see the row, records, fields, and so on. This shows you everything that has been loaded into the component buffers for that component.

- While at a breakpoint, if you lose track of the window, or the location within the window, that is displaying the green execution location arrow, you can use the Execution Location Properties menu item's ViewCode button to find your current execution location again.

- Objects remain expanded in the variable windows as you move through PeopleCode.

  This enables quick inspection of the state of an object as you step through the PeopleCode. However, there is a performance cost for using this feature. If you are finished examining an object, you may want to collapse it to improve the response speed.

- If a database transaction has been started (either for you by PeopleTools, or by you in PeopleCode) other users of that database are blocked from accessing that database until the transaction is complete.

Copyright © 1988, 2021, Oracle and/or its affiliates.

If you are stepping through PeopleCode while this transaction is open, you could potentially block other users for an extended period of time. You may want to use a private database for debugging to avoid blocking other users.

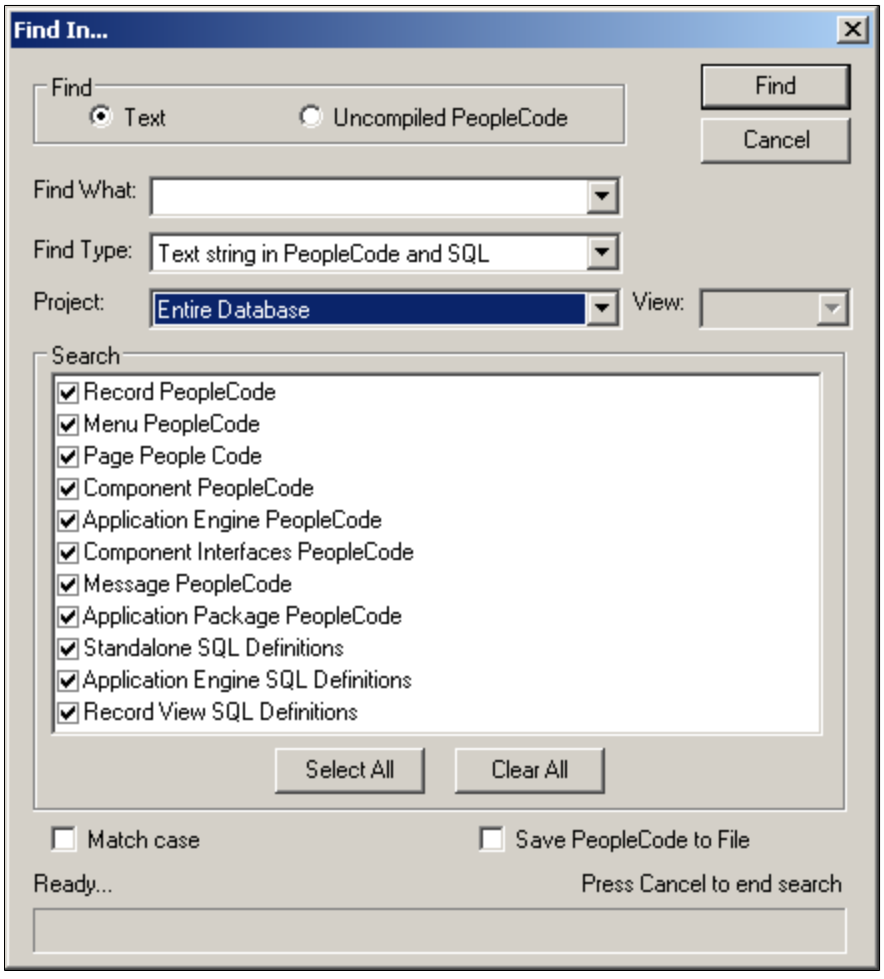
- Using the debugger is resource intensive and will impact overall system performance. Oracle recommends that you do not run debugger on your production system unless the issue you are trying to debug cannot be replicated in any other environment. If that is the case, debug when there is the least activity on the system. As an alternative you can try PeopleCode tracing.
- To create a file that contains all the PeopleCode for a project (or database), use the Find In feature and search for `;`.

  Be sure to select *Save PeopleCode to File.*

The following example shows the Find In dialog box:

Image: Find In dialog box

This example illustrates the fields and controls on the Find In dialog box. You can find definitions for the fields and controls later on this page.



See Using the Find In Feature.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

DoModal Considerations

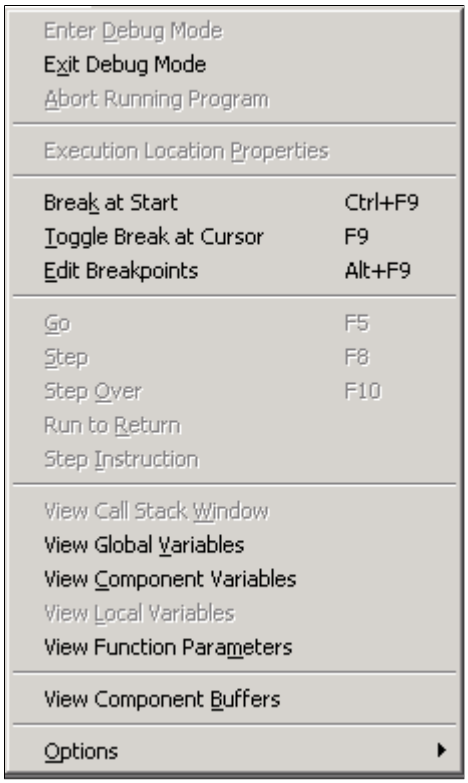
If you set the PeopleCode debugger to break at start and you are using the DoModal PeopleCode function, the DoModal window may appear behind the PeopleCode debugger window. The debugger may appear to have stopped, but it has not. Be sure to check that other windows have not opened while you are debugging the code.

Using PeopleCode Debugger Options

While the debugger is running, you can use the Debug menu to select other options:

Image: PeopleCode debugger options

This example illustrates the fields and controls on the PeopleCode debugger options. You can find definitions for the fields and controls later on this page.



Exit Debug Mode Exits debug mode. When you exit debug mode, all breakpoints are automatically saved. If you close Application Designer, you automatically exit debug mode.

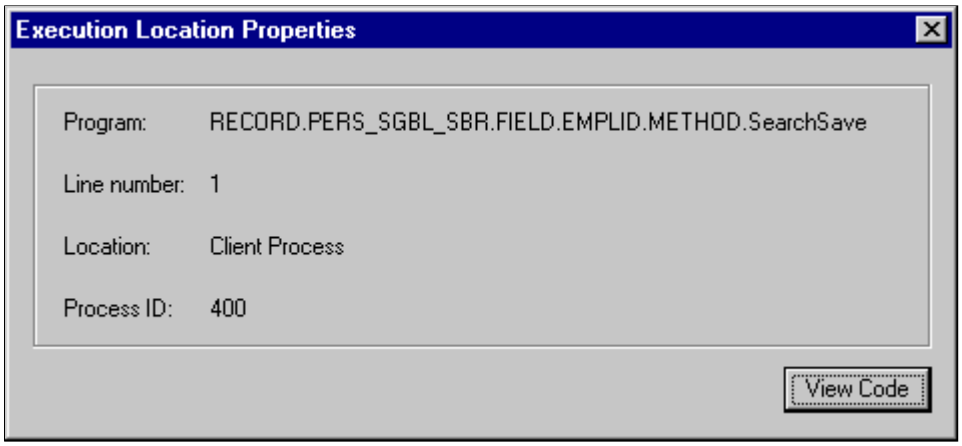
Abort Running Program Stops the PeopleCode program that is currently running.

Execution Location Properties Displays the location of the running code in a dialog box. This display includes the record name, field name, event name, and line number of the code. It also indicates if the code is executing on the client or server. You can view the exact code by clicking View Code.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Execution Location Properties dialog box

This example illustrates the fields and controls on the Execution Location Properties dialog box. You can find definitions for the fields and controls later on this page.

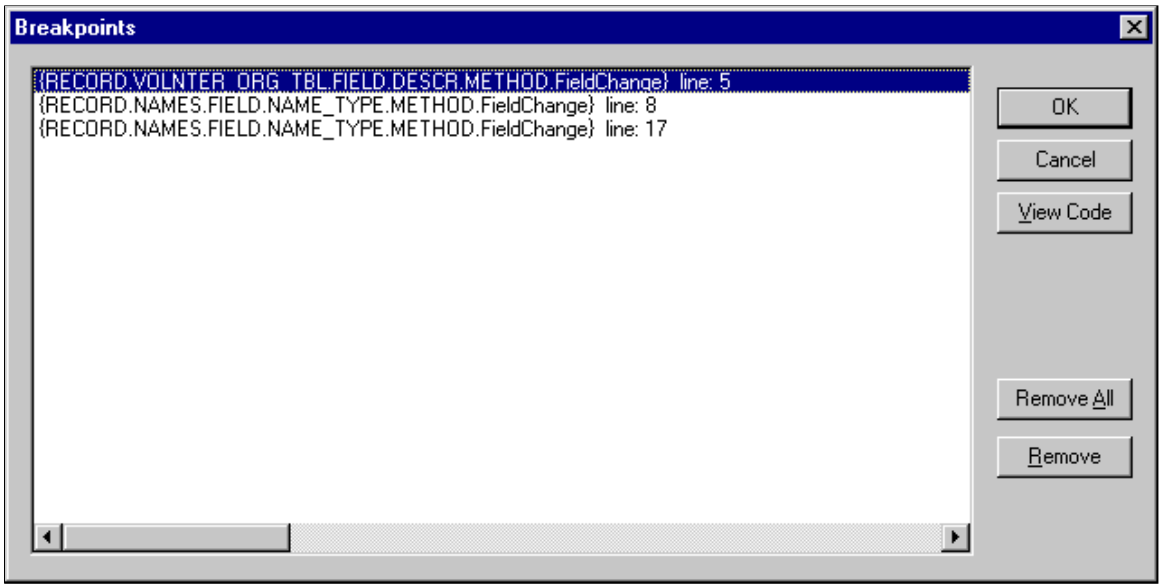


Break at Start	Pauses execution of the component on the first line of every PeopleCode program that executes in the component. If you start a component with Break at Start selected and then you start a second component, the PeopleCode associated with the second component is stopped at the first line of the first PeopleCode program as well, as part of the same debugging session.
Toggle Break at Cursor	Removes the breakpoint if the line the cursor is currently on has a breakpoint. Adds a breakpoint if the line the cursor is currently on does not have a breakpoint.
Edit Breakpoints	Opens a dialog box that displays the lines that have breakpoints. From this dialog box, you can display the code that contains the breakpoint by clicking View Code. You can also remove one or all breakpoints.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Breakpoints dialog box

This example illustrates the fields and controls on the Breakpoints dialog box. You can find definitions for the fields and controls later on this page.



Go	Continues processing until the next breakpoint. If Break At Start is enabled, processing pauses at the next PeopleCode program.
Step	Executes the current line of the PeopleCode program, stepping into functions.
Step Over	Steps through each line of the PeopleCode program, one line at a time, but steps over the functions; the functions are executed, but not stepped into.
Run to Return	Processes past the return of the current function, and then pauses.
Step Instruction	Processes low-level, pseudo-machine code instructions internal to PeopleCode. This option is used in conjunction with Log Options.
View Call Stack Window	Opens a separate window for viewing the call stack. The Call Stack window displays a stack of PeopleCode functions and methods that are currently active but not completed. You can use the Call Stack window to observe the flow of an application as it executes a series of nested functions.
View Global Variables	Opens a separate window for watching global variables.
View Component Variables	Opens a separate window for watching component variables.
View Local Variables	Opens a separate window for watching local variables.
View Function Parameters	Opens a separate window for watching user-specified parameters in function calls.

Copyright © 1988, 2021, Oracle and/or its affiliates.

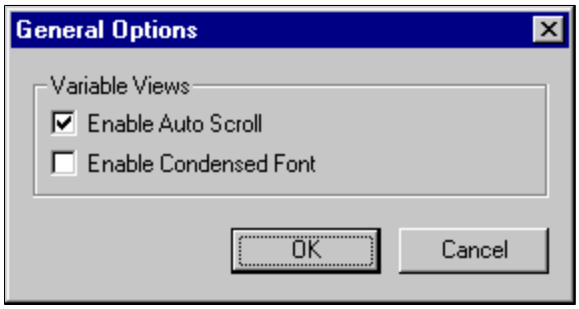
View Component Buffers	Opens a separate window for viewing the current component buffers. This is equivalent to getting a level zero rowset for the component.

Note: The previous five windows update continuously as the program executes.

Options	Enables you to select between opening a dialog box for general options or for specifying log options.

Image: General Options dialog box

This example illustrates the fields and controls on the General Options dialog box. You can find definitions for the fields and controls later on this page.



The General Options dialog box enables you to specify conditions of the view windows. The default is for both of these options to be selected.

Enable Auto Scroll	If you select this check box and you click a plus symbol next to a variable name in a view window, the variable you clicked scrolls to the top of the window.
Enable Condensed Font	Select to display all view windows with a smaller font.

Additional Features

Break at Termination	After you are in debug mode, generally, any PeopleCode program in the session that terminates abnormally first breaks in the debugger. In addition, the error message appears in the PeopleCode log in the bottom window of Application Designer.

Related Links

Setting PeopleCode Debugger Log Options

Setting Up the Debugging Environment

You can use the PeopleCode debugger for two-tier and three-tier debugging. The database and application can reside on remote servers; they do not need to reside on the local machine.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Two-tier debugging works out of the box. Setting up three-tier debugging requires you to make a few modifications in PSADMIN (PSAPPSRV.CFG) to enable debugging.

You can connect to a Microsoft Windows NT server domain that is not on your local machine. You do not have to configure a local domain to do this. You also do not have to have PeopleTools software installed locally for three-tier debugging.

Note: Application Designer will not connect to the PeopleCode debugger through a firewall using Network Address Translation (NAT).

Related Links

"Setting Up the PeopleCode Debugger" (PeopleTools 8.59: System and Server Administration)

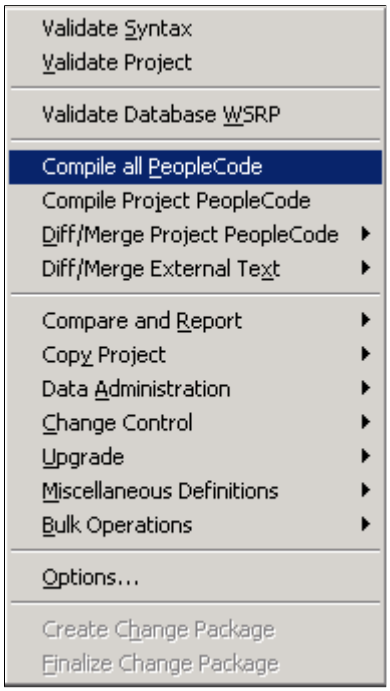
Compiling All PeopleCode Programs at Once

In addition to checking individual programs, you can compile all PeopleCode programs either in a database or in a project to check for errors. This option opens and compiles every PeopleCode program. This process can be run on an as-needed basis to check for corruption in your programs. Run this option after an upgrade to verify that all the programs were upgraded correctly.

You run this option from the Tools menu:

Image: Tools menu - Compile All PeopleCode option

This example illustrates the fields and controls on the Tools menu - Compile All PeopleCode option. You can find definitions for the fields and controls later on this page.



Copyright © 1988, 2021, Oracle and/or its affiliates.

Compile All PeopleCode

To compile all PeopleCode programs:

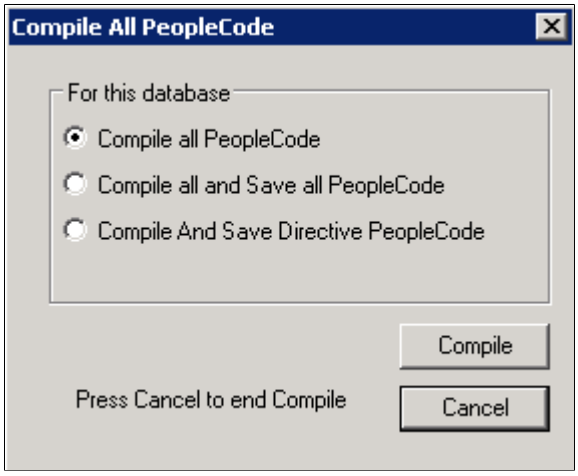
1. Open Application Designer while accessing the database that contains the PeopleCode that you want to check.
2. Select Tools, Compile All PeopleCode
3. Select which PeopleCode to compile.
4. Click Compile in the Compile All PeopleCode dialog box.

   Errors appear in the PeopleCode log display window. Warnings and auto-declared variables do not appear in the log display window, however the number of warnings and the number of auto-declared variables appear in the log display window.

Note: If you specified a log file in the debugger log options, then all errors are written to the log file as well.

Image: Compile All PeopleCode dialog box

This example illustrates the fields and controls on the Compile All PeopleCode dialog box. You can find definitions for the fields and controls later on this page.



Compile all PeopleCode Select this option to validate every PeopleCode program in the database.

Compile all and Save all PeopleCode Select this option to open, compile and save every PeopleCode program.

Compile and Save Directive PeopleCode Select this option to open, compile and save directive PeopleCode.

You can also compile PeopleCode from the command line, including the ability to compile directive PeopleCode.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Directive PeopleCode

Directive PeopleCode provides the ability write PeopleTools release specific PeopleCode. Using directive PeopleCode you can conditionally ignore PeopleCode objects in PeopleCode events based on the PeopleTools release. Directive PeopleCode uses #If #ToolsRel and #End-If. For example:

```
#If #ToolsRel >= "8.54" #Then
   &PhotoSizeName = "CARD";
#Else
&PhotoSizeName = "ORIG";
#End-If
```

The PeopleTools release is stored in the PSPCMPROG table in the PTTOOLSREL field. When directive PeopleCode is compiled, the system will check the PeopleTools release for the database and compile the correct code for that PeopleTools release.

Note: Compiling directive PeopleCode is included as a step in all PeopleTools upgrades.

For more information on directive PeopleCode functions and constructs, see "Directive PeopleCode Functions and Constructs" (PeopleTools 8.59: PeopleCode Language Reference).

Related Links

"Command Line Parameters" (PeopleTools 8.59: Lifecycle Management Guide)

"Compiling and Saving Directive PeopleCode" (PeopleTools 8.59: Lifecycle Management Guide)

Compile Project PeopleCode

To compile all PeopleCode programs:

1. Open Application Designer while accessing the database that contains the PeopleCode that you want to check.
2. Open the project containing the PeopleCode to compile.
3. Select Tools, Compile Project PeopleCode.

   Errors appear in the PeopleCode log display window.

   Note: If you specified a log file in the debugger log options, then all errors are written to the log file as well.

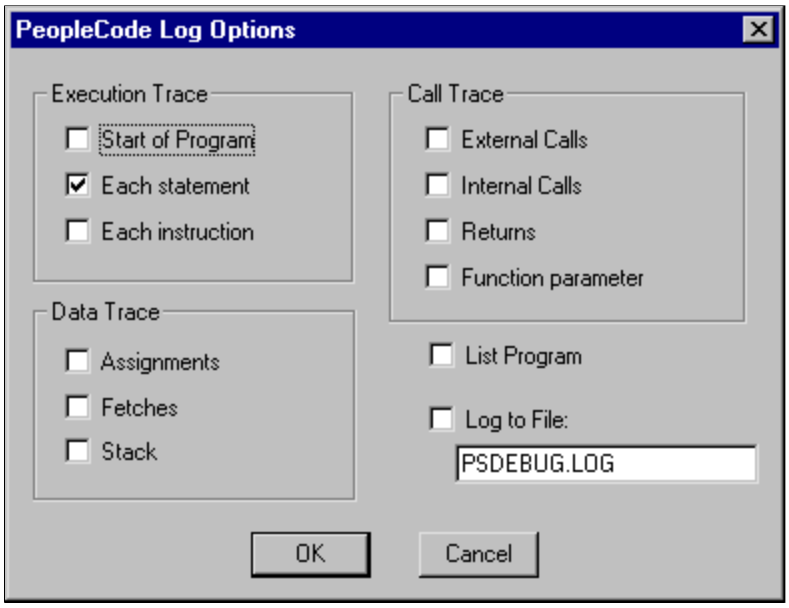
Setting PeopleCode Debugger Log Options

Use the PeopleCode debugger to view PeopleCode that is executed while you step through your application. Select Debug, Log Options to access the PeopleCode Log Options dialog box.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: PeopleCode Log Options dialog box

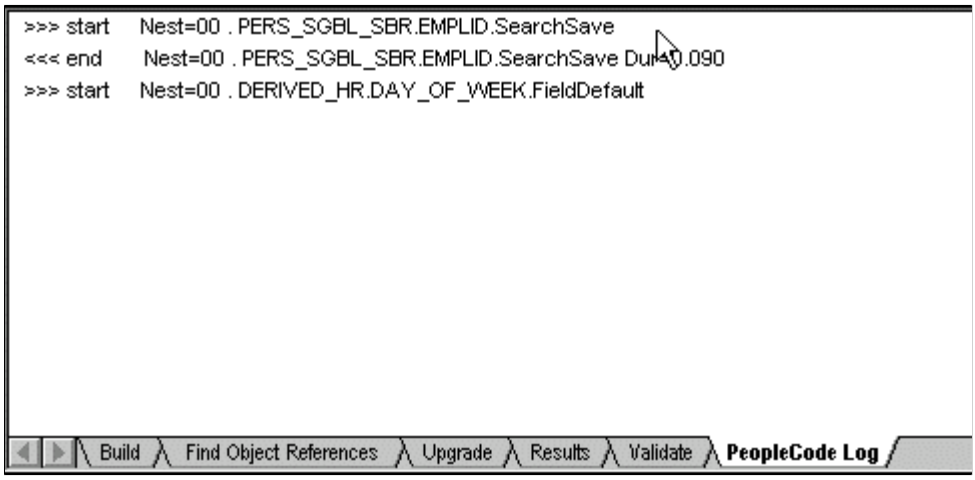
This example illustrates the fields and controls on the PeopleCode Log Options dialog box. You can find definitions for the fields and controls later on this page.



All log information appears in the PeopleCode log window, at the bottom of Application Designer.

Image: PeopleCode log window

This example illustrates the fields and controls on the PeopleCode log window. You can find definitions for the fields and controls later on this page.



You can record what you see in a log file. Also, you can tailor the log results to record a variety of online information.

If you exit debug mode but do not close Application Designer, all the log options that you specified are still there when you start debug mode again.

When you close the Application Designer, all log options are clear. The next time you enter debugging mode, you must reselect debug log options.

See Interpreting the PeopleCode Debugger Log File.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

All the options available in the Log Options dialog box are also available in PeopleSoft Configuration Manager, on the Trace tab, in the PeopleCode Trace section.

Execution Trace Options

Execution trace is set to trace each PeopleCode statement. You can also trace the start of each program or each program instruction.

Data Trace Options

This table describes the data trace options:

Option	*Description*
Assignments	Records each assignment made to a field.
Fetches	Records the field values retrieved from a PeopleCode fetch.
Stack	Indicates the contents of the internal machine stack. Typically, only PeopleSoft staff developing PeopleCode language enhancements use this option.

Call Trace Options

The call trace options, described in the following table, enable you to record the values of external calls, internal calls, returns, and function parameters.

Option	*Description*
External calls	Traces each call to external (PeopleCode) functions.
Internal calls	Records each call to internal subroutines.
Returns	Logs the occurrence of program returns.
Function parameters	Logs the value of individual PeopleCode function parameters.

Log To File

When you select this option, you must specify the name of a file, or you receive an error and logging to file is disabled.

If you do not specify a directory location, the file is placed in the same directory from which you are running PeopleTools.

If you specify the name of an existing file, a warning message appears, asking you whether to overwrite the file. You must go back into the Log Options dialog box and specify a different file name; otherwise, the log file is overwritten.

If you do not exit Application Designer before running a different application, each trace is appended to the specified log file.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"Specifying Trace Settings" (PeopleTools 8.59: System and Server Administration)

Interpreting the PeopleCode Debugger Log File

You can produce a trace log using any of the following methods:

- Using the Log File option in the PeopleCode debugger.
- With the PeopleSoft Configuration Manager Trace tab.
- Using the SetTracePC and SetTraceSQL built-in functions.
- With PeopleTools utilities (included for backward compatibility purposes only and should not be used).

All trace files except those produced using the Log File option contain timing information, such as when each line started processing and how long it took to execute.

The Log File option writes to a file that you specify. The log file produced by the other options is specified by the PeopleTools Trace File option in PeopleSoft Configuration Manager. All of these options write to the same file.

Trace files are also produced by Application Engine. These logs may contain more information.

This section discusses:

- Log file contents.
- Other items in the log file.

Related Links

"Using Debug Utilities" (PeopleTools 8.59: System and Server Administration)

Log File Contents

The log file contains information useful for debugging PeopleCode.

You can view the log using any editor that displays ASCII text, such as Notepad. The log file has the following components.

Line Count	Specifies a line number in the file.
Internal Information	Contains reference numbers used for internal tracing. You can ignore this information.
Instruction Location	Address of an instruction processed in the program. You can follow programs and functions using this number.
Operation Code	Indicates the operation performed by the program.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

Operation Operands Contains information specific to each operation. The following table lists the possible operations and the operands that appear for the list and trace options.

Other Items in the Log File

The following table describes other items that can appear in a debugging trace:

Trace Item	*Description*
Store Field:record name.field name Value=xx	Issued when the assignments trace option is selected. It contains the record and field names and the value that is stored.
Fetch Field:recordname.fieldname Value=xx	Issued when the Fetch Field option is selected. It contains the record and field name and the value that is retrieved.
Fetch Field:recordname.fieldname Contains Null Value	Issued when the Fetch Field option is selected and the selected record.field contains a null value.
Fetch Field:recordname.fieldname Does Not Exist	Issued when the Fetch Fields option is selected and when the field is not found.
Branch Taken	Displayed after a branch test when the branch is taken.
Field Not Found, Statement Skipped	Displayed whenever a referenced field was not found error causes the PeopleCode processor to skip to the next statement.
vvvvvv PeopleCode Program Listing	Issued when the List Program option is selected. It marks the beginning of a PeopleCode program listing.
^^^^^ PeopleCode Program Listing End	Issued when the List Program option is selected. It marks the end of a PeopleCode program listing.
Error Return -> NNN	Issued when a fatal error condition terminates the PeopleCode program.

Using Application Logging

Application logging enables you to do error logging using an independent application log fence mechanism. It also enables you to write to the PeopleTools log using the WriteToLog built-in function.

Note: This is an application log fence, and it is distinct from the PeopleTools LogFence setting.

In PeopleTools, a log fence is a type of barrier. Application error messages are only written to the PeopleTools log if the log fence setting that the messages are written to the log with (using WriteToLog) is less than or equal to the current application log fence setting (AppLogFence) in the application server configuration file (PSAPPSRV.CFG).

Copyright © 1988, 2021, Oracle and/or its affiliates.

For example if the AppLogFence setting is 2, only messages written using the WriteToLog function with a log fence value less than or equal to 2 will be written. This allows you to have application logging code written in your application that will only be in effect if the log fence setting permits.

The application log fence setting is available through the system variable %ApplicationLogFence.

Apart from the obvious use of allowing the application to write to the Tools log file, this mechanism is also an aid in debugging. For example, you could interleave PeopleCode, SQL, and application level tracing in the same log file to easily correlate application and PeopleTools actions.

Related Links

"WriteToLog" (PeopleTools 8.59: PeopleCode Language Reference)

"%ApplicationLogFence" (PeopleTools 8.59: PeopleCode Language Reference)

Setting the Application Log Fence in the Configuration File

The application log fence default is %ApplicationLogFence_Level1 (3). If you want to use this setting, you need to place it in the application server configuration file (PSAPPSRV.CFG.) The setting is dynamic change enabled; that is, if its value is changed in the file, then the new value will be used. As the following example illustrates, the AppLogFence setting must be in the PSTOOLS section. If you add this setting, your configuration file can look like this:

```
[PSTOOLS]
;======================================================================
; General settings for PSTOOLS
;======================================================================
AppLogFence=1
```

Related Links

"PSAPPSRV Options" (PeopleTools 8.59: System and Server Administration)

Using the Log Fence with PeopleSoft Analytic Calculation Engine

If you set the application log fence to 3 or above, all the detailed messages created on the analytic server to be sent back to the application server are also logged in the analytic server log file.

In addition, if you set the application log fence to 4 or above, all tracing information is logged to the analytic server log file.

Using the Find In Feature

Use the Find In feature of Application Designer to search for strings, either in PeopleCode programs or in SQL definitions. This feature searches:

- All PeopleCode programs and all SQL statements.
- Only PeopleCode programs.
- Only SQL statements.
- Only HTML definitions.

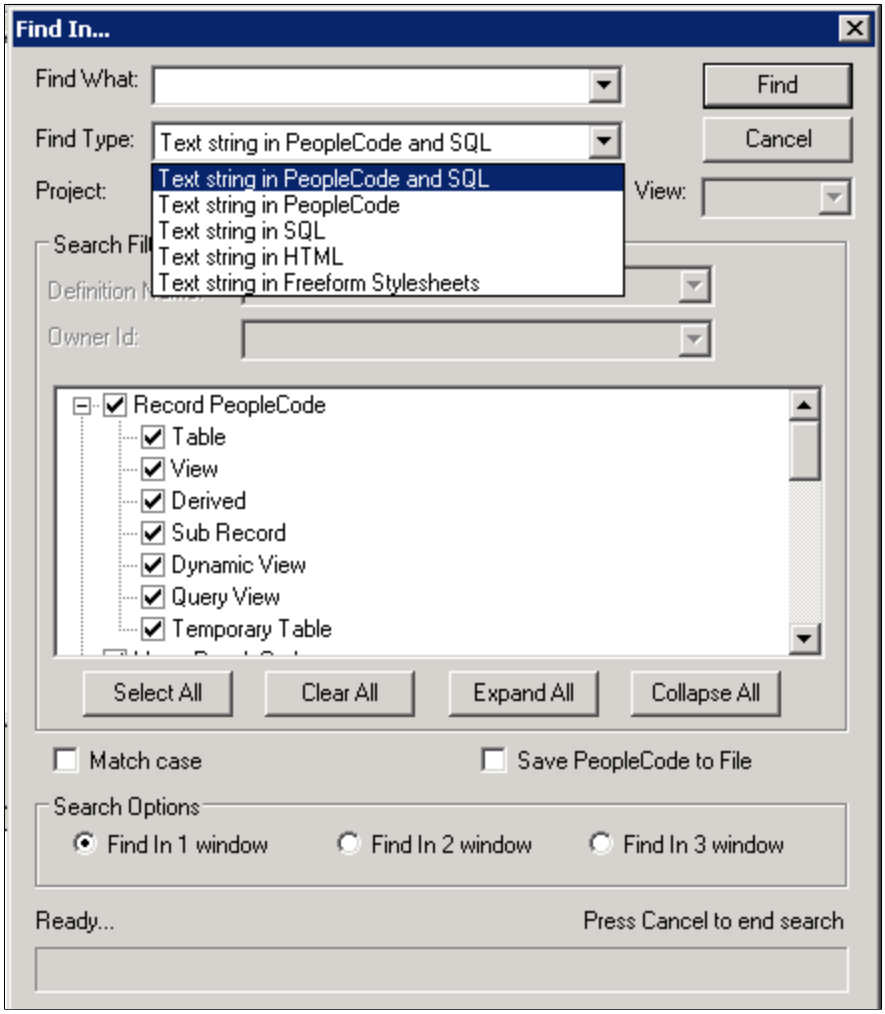
 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Only freeform style sheets.

The following examples illustrate the fields and controls in the Find In dialog box.

Image: Example of the Find In dialog box showing options for Find Type

This example illustrates the options for the Find Type list box:

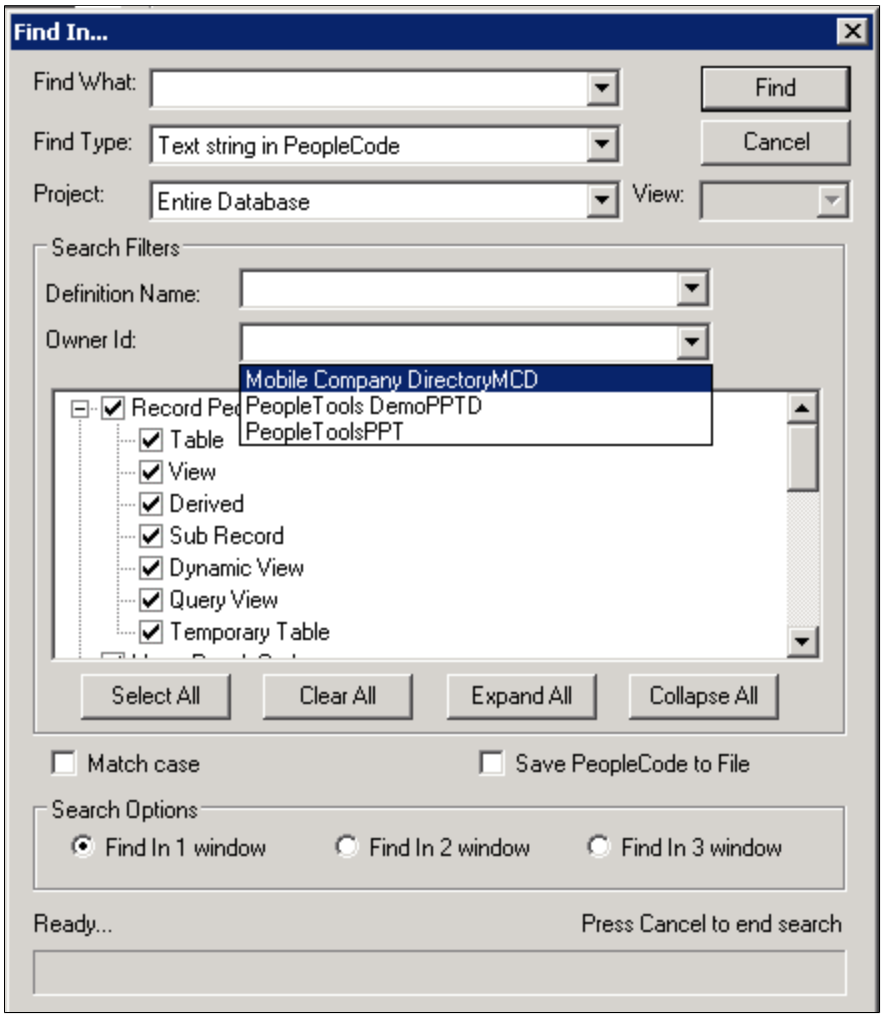


Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Example of the Find In dialog box showing options for Definition Name and Owner ID

The Definition Name is a combination box which allows text edits and includes a drop down list box that lists the 10 most recently used (MRU) items. When any text is provided in this box, objects with their names beginning with the provided text are listed in the results. When no text is provided, all the results are displayed without filtering by name. This filter is currently available for PeopleCode search only.

The Owner ID list box has all the owner ID values in the system. When any value is selected from the list box, only objects having selected owner IDs are listed in the results. When no value is selected, the filter is not applied to the search. This example illustrates the options for the Definition Name and Owner ID filters:



The Definition Name and Owner ID filters are independent of each other and can be used along with other filters.

You can select to search in a specific project or the entire database. If a project is currently open, then the initial search scope is that project. If you are searching PeopleCode programs and SQL statements, you can specify if you want record PeopleCode, page PeopleCode, menu PeopleCode, and so on.

When you specify record PeopleCode or page PeopleCode, you can filter the search by selecting the options available under record PeopleCode or page PeopleCode.

The following options are available in record PeopleCode to filter your search:

 Copyright © 1988, 2021, Oracle and/or its affiliates.

- Table
- View
- Derived
- Sub Record
- Dynamic View
- Query View
- Temporary Table

The following options are available in page PeopleCode to filter your search:

- Standard Page
- Sub Page
- Secondary Page
- Popup Page
- Layout Page

All output from the search is placed in an output window. You can save these results to a file, copy them, clear them, or print them.

From the output window, you can immediately open any of the PeopleCode programs, SQL statement, HTML definitions, or free-form style sheets listed. You also can insert selected definitions into a project from the output window. Then, if you need to search those definitions again, you can search by project.

Note: To create a file that contains all the PeopleCode for a project (or database) you can use the Find In feature and search for `;`. Be sure to select *Save PeopleCode to File.*

To find a text string:

1. In Application Designer, select Edit, Find in. Alternatively, if a project is open, right-click the project name in the project workspace; select Find In Project from the pop-up menu.

   The Find In dialog box appears. If a project is currently open, then the initial search scope is that project. Otherwise, the search scope is the entire database.

2. Type the string that you want to find in the Find What edit box.

   If you want only those items that match the case of what you entered, select the Match Case check box at the bottom of the dialog box.

3. Specify with the Find Type edit box whether you are searching in PeopleCode and SQL, just PeopleCode, just SQL, HTML definitions, or freeform style sheets.

4. Select the search scope.

   You can search the entire database or any existing project.

5. (Optional) Select the view to search.

Copyright © 1988, 2021, Oracle and/or its affiliates.

If you decide to not search the entire database, you can specify if you want to search the Development view or the Upgrade view. The default is the Development view.

6. Select the items to search.

   You can search all items that contain either PeopleCode or SQL, or a subset of items.

   Note: When you select a Find Type of Text String in HTML the Search check box list is empty. The search is conducted against all HTML definitions.
   Similarly, when you select a Find Type of Text String in Freeform Stylesheets the Search check box list is empty. The search is conducted against all freeform style sheets.

7. (Optional) Select the option to save the search results to a file.

   You can save the results of a PeopleCode search to a text file, which you can view or print using a text editor or word processor. The text file contains the entire PeopleCode program that contained the string.

   To save your results to a file, select the Save PeopleCode to File check box at the bottom of the dialog box. The results are saved to the file, and appear in the Application Designer Find In output window.

   This option is not available when searching SQL, HTML, or freeform style sheets.

8. Select which Find In output window to display the search results: 1, 2, or 3.

   Note: This option allows users to retain and display Find In results. The searches can be conducted in any of the three windows and displayed in the corresponding tab in the output window.

9. Click the Find button to start the search.

   As the Find In feature searches the database, it displays a counter at the bottom of the Find In dialog box indicating the number of PeopleCode programs searched.

   You can click the Cancel button to stop the process.

10. Check the Find in tab on the output window for results.

    The results of the search appear in the Find In tab of the output window. Each line shows where the string was found. You can open any of the programs listed by double-clicking a line in the output window.

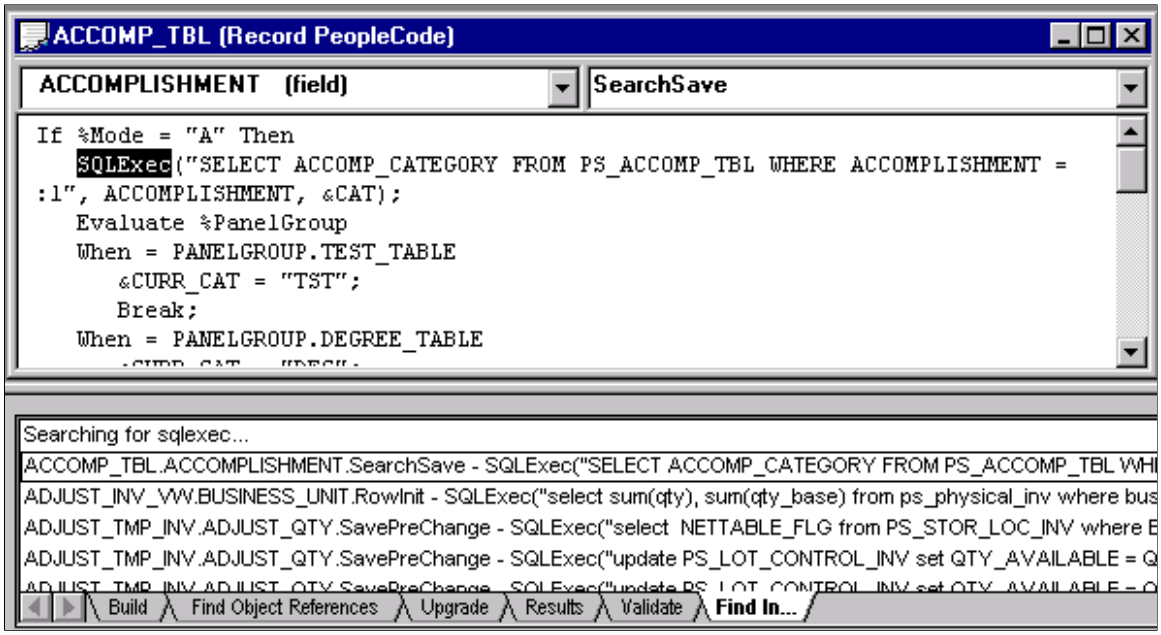
Note: To improve searching for text strings in PeopleCode programs, you can compile and save all PeopleCode in the database first. To do so, select the Compile all and Save all PeopleCode option on the Compile All PeopleCode dialog box. See Compile All PeopleCode for more information.

The following example shows the Find In tab of an output window:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Opening a PeopleCode program from the Find In tab

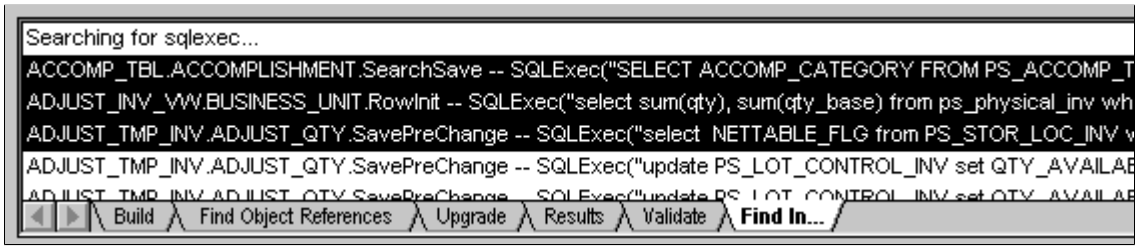
This example illustrates the fields and controls on the Opening a PeopleCode program from the Find In tab. You can find definitions for the fields and controls later on this page.



To save records, you select them in the output window, as shown in the following example:

Image: Find In output window with definitions selected

This example illustrates the fields and controls on the Find In output window with definitions selected. You can find definitions for the fields and controls later on this page.



To save definitions in a project:

1. Use the Find In feature to search for a string.
2. Press the Shift key while selecting the references to save in the output window.
3. Right-click the highlighted definitions and select Insert Into Project.

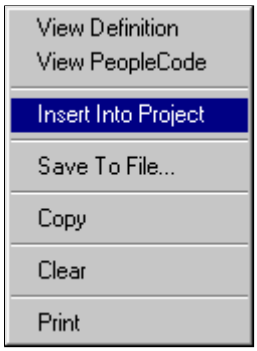
   All the selected definitions are inserted into the current open project.
4. Save your project.

The following example shows the Insert Into Project option:

Copyright © 1988, 2021, Oracle and/or its affiliates.

Image: Insert Into Project option of the Find In pop-up menu

This example illustrates the fields and controls on the Insert Into Project option of the Find In pop-up menu. You can find definitions for the fields and controls later on this page.



Finding References to Application Packages and Classes

You can use the Find Definition References menu item to quickly find references to application packages or classes.

Prerequisites

To be able to search for references to application packages or classes, you must compile all PeopleCode for the database first:

Note: This compilation process is a one-time activity per database, which takes many hours to complete and depending on the size of your database.

1. Sign into Application Designer in two-tier mode.
2. In Application Designer, select Tools > Compile all PeopleCode.
3. In the Compile All PeopleCode dialog box, select the Compile all and Save all PeopleCode option.
4. Click the Compile button.

After this initial PeopleCode compilation, new application package and class definitions and references are automatically added to the database.

Important! When importing a project that contains application package and application class definitions, you must select the Compile PeopleCode after Import option to add these definitions to the database.

Related Links

Compiling All PeopleCode Programs at Once

Copyright © 1988, 2021, Oracle and/or its affiliates.

Finding Definition References

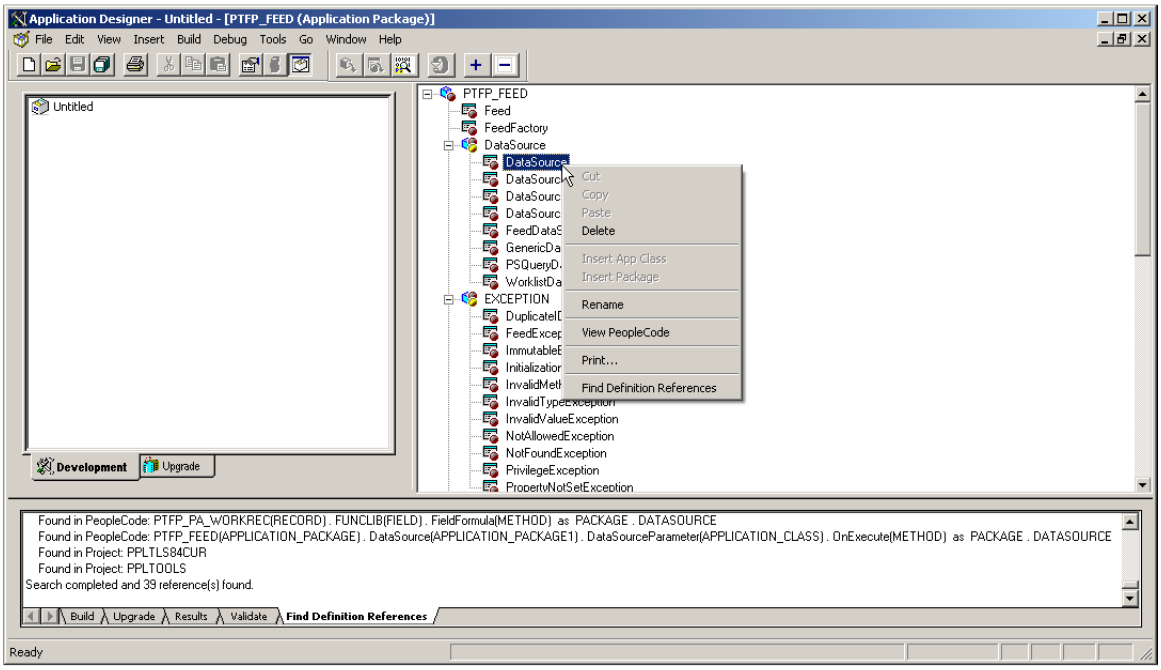
After you have compiled and saved all PeopleCode, you can search for references to application packages or application classes or application class methods:

1. In Application Designer, open the application package in the Application Package Editor.
2. In the application package structure, browse to the application package or application class or application class method for which you want to find definition references.
3. Select Edit, Find Definition References, or right-click the definition and select Find Definition References from the pop-up menu.

Note: When you want to find definition references for record definition or field definition, the Find Definition References dialog box displays a set of options for filtering the definition types. For more information on search filters for record and field definitions, see "Finding Definition References" (PeopleTools 8.59: Application Designer Developer's Guide).

Image: Finding definition references for an application class

This example illustrates the fields and controls on the Finding definition references for an application class. You can find definitions for the fields and controls later on this page.



After you select this item, a search of the database takes place, and the results appear on the Find Definition References tab of the output window.

4. Select any definition that is displayed in the Find Definition References tab by double-clicking it.

   The selected definition opens in the editor.

Note: Find definition references will not find an application package or class name when it appears in a PeopleCode comment or when the name appears in a string literal (that is, within quotes). Use the Find In feature instead.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

Using the Find In Feature

Using Cross-Reference Reports

If a field value changes and you do not know how it changed, you can find all references to a field using:

- The Find References option in Application Designer.
- Cross-reference reports.

See "Understanding Projects" (PeopleTools 8.59: Application Designer Developer's Guide)"Using Projects" (PeopleTools 8.59: Application Designer Developer's Guide).

PeopleTools is delivered with these PeopleCode cross-reference reports:

- XRFFLPC.

  Reports on all fields in the system referenced by other PeopleCode programs. The report sorts by record names and field names. XRFFLPC shows the records, fields, and PeopleCode program types that reference each field.

- XRFPCFL.

  Reports on the fields that each program references. It sorts the report by record definition, field name, and PeopleCode type. It shows the records and fields referenced for each program. This report and XRFFLPC complement each other by using converse approaches to reporting the cross references.

- XRFPNPC.

  Reports on pages with PeopleCode. This report shows pages containing fields with PeopleCode attached to them.

You can run these reports using PeopleSoft Query and either view the reports online or print them. You can also download them to a Microsoft Excel spreadsheet. The following example shows an XRFPNPC report:

Image: Example of XRFPNPC PeopleSoft Query results

This example illustrates the fields and controls on the Example of XRFPNPC PeopleSoft Query results. You can find definitions for the fields and controls later on this page.

	Panel Name	Record	Field Name	Prog Type
1	ACL_APPDES_MISC	DERIVED_APPDESM	ACCESS_CD	FieldChange
2	ACL_APPDES_MISC	DERIVED_APPDESM	ACCESS_CD	RowInit
3	ACL_APPDES_MISC	DERIVED_APPDESM	FULL_ACCESS_BTN	FieldChange
4	ACL_APPDES_MISC	DERIVED_APPDESM	ITEM_CHANGED	SavePreChange
5	ACL_APPDES_MISC	DERIVED_APPDESM	NO_ACCESS_BTN	FieldChange
6	ACL_APPDES_MISC	DERIVED_APPDESM	READ_ACCESS_BTN	FieldChange
7	ACL_APPDES_OBJECTS	DERIVED_APPDESO	ACCESS_CD	FieldChange
8	ACL_APPDES_OBJECTS	DERIVED_APPDESO	ACCESS_CD	RowInit
9	ACL_APPDES_OBJECTS	DERIVED_APPDESO	FULL_ACCESS_BTN	FieldChange

Copyright © 1988, 2021, Oracle and/or its affiliates.

Related Links

"PeopleSoft Query Overview" (PeopleTools 8.59: Query)

"Understanding Cross-Reference Reports" (PeopleTools 8.59: Application Designer Developer's Guide)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Chapter 16

Improving Your PeopleCode

Reducing Trips to the Server

This section discusses how to:

- Count server trips.
- Use deferred mode.
- Hide and disable fields.
- Use the Refresh button.
- Update totals and balances.
- Use warning messages.
- Use the fastest algorithm.

Server trips are bad for performance. Each server trip consumes resources on the application server, slows down the user data entry, and can affect type ahead. Whenever you see an hourglass as you move between fields on a page, it is because the browser is waiting for a server trip to complete.

The larger the component's buffer (based on the number of record definitions accessed, the number of fields in each record, and the number of rows in each grid or scroll area for each record), the longer each round trip to the server, because of the increased server processing.

Deferred mode reduces the user's time to complete the transaction and conserves application server resources.

The following user interactions cause a trip to the server. Only the first three items in the list are deferred in deferred processing mode.

- Entering data in fields with FieldEdit or FieldChange PeopleCode.
- Entering data in fields that have prompt table edits.
- Entering data in fields that have related displays.
- Inserting a row in a grid or scroll area.
- Deleting a row from a grid or scroll area.
- Using grid or scroll area controls to move forward or back.
- Accessing another page in the component.
- Selecting an internal tab.

Copyright © 1988, 2021, Oracle and/or its affiliates.

- Expanding or collapsing a collapsible section.
- Clicking a button or link.

Each trip goes through the same process of checking security, unpacking the buffers that store the data being processed, processing the service request, generating the HTML for the page to be redisplayed, packing updated buffers, and storing the buffers on the web server. To maximize online performance, minimize server trips.

Counting Server Trips

Count the trips to the server to quickly identify transactions that have performance issues. PeopleTools can automatically count these trips by reason (such as, adding a row in a grid or FieldChange PeopleCode) and write the output to a log file.

To turn this feature on, run a debug version of PeopleTools and add the following to the [trace] section of the appserv.cfg file:

```
showcounters = 1
```

The output is written to the appsrv.log file.

Using Deferred Mode

Keep components in deferred mode and enable fields for interactive mode only if there is a strong business case.

For every field on the component to run in deferred mode, Deferred mode must be selected at the component level, Allow Deferred Processing must be selected for each page in the component, and Allow Deferred Processing must be selected for each field.

PeopleSoft recommends that you continue to code field edits in FieldEdit PeopleCode and field change logic in FieldChange PeopleCode, but set this logic to run in deferred mode. You do not need to move field edits to SaveEdit.

Hiding and Disabling Fields

Avoid using FieldChange PeopleCode to hide, unhide, enable, or disable elements on the same page, unless the element is triggered by a separate button.

Hiding or unhiding objects and enabling or disabling objects should, as a general rule, be coded in either page Activate PeopleCode or, for objects that are on another page in the component, in FieldChange PeopleCode.

Perform cross-validation edits to prevent invalid data combinations from being written to the database for fields that previously would have been hidden or unavailable. If unhiding fields that were previously hidden or unavailable results in making the page confusing, consider designing a longer page so that users can easily associate related fields.

You can hide or unhide objects or set them to display-only in page Activate PeopleCode before the page initially appears based on setup data, configuration options, or personalization settings. You can set fields to display-only using PeopleCode by setting the DisplayOnly property for the field to True.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

You can hide or unhide fields on another page, or set the fields to display-only, based on the value that a user enters in a field on the current page, as long as that component or field is set up to run in deferred processing mode. In some cases, it may make sense to split transactions across pages to achieve progressive disclosure.

Using the Refresh Button

The Refresh button gives users control of their environment. Clicking the Refresh button forces a trip to the server. PeopleTools then redisplays the page in the browser. The refresh action allows the user to:

- See related display field values for the data entered so far.
- See any default values based on data entered previously on the page.
- Validate the data that has been entered on the page so far.

When the page is redisplayed, the cursor is positioned in the same field it was when the user pressed the Refresh button.

Note: The Refresh button does not refresh the page from the database. It simply causes a server trip so that any deferred PeopleCode changes are processed. If no deferred changes exist or the deferred changes do not cause any errors or other changes on the page, it may appear to the user as if nothing has happened. Fields on derived work records are not updated if the user clicks the Refresh button.

Updating Totals and Balances

In some pages, totals or balances appear based on data entered into a grid or scroll area. This process should work in deferred mode also, showing the totals or balances as of the last trip to the application server.

Continue to keep any accumulation and balancing logic in FieldChange PeopleCode, but run the field in deferred mode. Users can click the Refresh button at any time to see the latest totals based on the data entered. Totals and balances in deferred mode are always updated and displayed after any trip to the application server.

Using Warning Messages

In deferred mode, FieldEdit PeopleCode errors and warnings do not appear when a user moves out of the field, but rather on the next trip to the server. This next trip might not occur until the user enters all the data and clicks the Save button.

For FieldEdit error messages running in deferred mode, PeopleTools changes the field to red and positions the cursor to the field in error when it displays the message. This behavior allows the user to associate the error message with a specific field.

For warning messages, however, PeopleTools does not change the field to red or position the cursor. For a user to clearly understand to which field a warning message applies, ensure that warning messages clearly describe the fields affected by the warning.

For example, the warning message “Date out of range” would be confusing if there are seven date fields on the page, since a user could not easily determine which date field needed to be reviewed. Instead, you could include bind variables in the message to show which dates are out of range.

Copyright © 1988, 2021, Oracle and/or its affiliates.

Using the Fastest Algorithm

You should determine which algorithms perform the best and have the smallest elapsed time. Tracing does not provide subsecond level of timing information. Plus, tracing imposes a higher overhead to the runtime environment, which skews the elapsed time reading.

However, you can use the %PerfTime system variable for determining elapsed time. %PerfTime retrieves the local system clock time by making a system call, and the return time is down to the millisecond.

The following example of %PerfTime determines how long a program takes to execute:

```
   &Start = %PerfTime;
   &results = "";
   For &I = 1 To &Count;
      &GnnwgNumber = GetNextNumberWithGapsCommit(QEORDER_DTL.QE_QTY, 999999, 1, "wh⇒
ere QE_ORDER_NBR='GNNWG'");
      &results = &results | " : " | &GnnwgNumber;
   End-For;

   &End = %PerfTime;
   &out = "Count = " | &Count | ", total GNNWG time (s) = " | NumberToString("%6.3"⇒
, Value(&End - &Start));
```

Using Better Coding Techniques for Improved Performance

This section discusses how to:

- Run a SQL trace.
- Optimize SQL.
- Use the GetNextNumberWithGaps function.
- Consolidate PeopleCode programs.
- Move PeopleCode to a component or page definition.
- Send messages in the SavePostChange event.
- Use metadata and the RowsetCache class.
- Setting MaxCacheMemory

Running a SQL Trace

Run a SQLTrace and review the transaction for SQL statements that have a long processing time.

The duration column (Dur=) in a SQL trace displays this information. If the duration is greater than 100 milliseconds, you may be able to make this SQL statement run faster. Work with your database administrator to tune the SQL.

Optimizing SQL

A simple join optimizes SQL more effectively than issuing two related SQL statements separately.

 Copyright © 1988, 2021, Oracle and/or its affiliates.

However, if your transaction requires a complex SQL statement (for instance, one that uses correlated subqueries), consider breaking it up into multiple SQL statements. You may get more predictable performance this way.

Using the GetNextNumberWithGaps Function

Many applications use a sequence number as a unique key. The last number used is stored in a common table, and a SQL statement is issued to retrieve the last number used and update the table. This action locks the common table until the whole transaction is saved and the unit of work committed.

Instead, consider using the GetNextNumberWithGaps PeopleCode function whenever gaps in the sequence numbering are acceptable. The function retrieves the last number used, increments it by one, and updates the common table. This action is done in a separate unit of work to minimize the time a database lock is held on the common table.

GetNextNumberWithGaps issues a commit only when issued from the SavePreChange or Workflow event.

Consolidating PeopleCode Programs

Consolidate RowInit PeopleCode into one field within the record to reduce the number of PeopleCode events that need to be triggered. Fewer PeopleCode programs results in fewer PeopleCode objects to manage. Do the same for RowInsert, SaveEdit, SavePreChange, SavePostChange, and Workflow PeopleCode programs.

Moving PeopleCode to a Component or Page Definition

Analyze transactions and move PeopleCode that is specific to a component from the record definition to the component or page definition. This action eliminates the need to execute conditional statements, such as `If %Component =`.

This action helps only if you are able to move all the PeopleCode in a program from the record to a component or page, and multiple components access that record.

Sending Messages in the SavePostChange Event

Messages sent online should always be coded in the SavePostChange event. To minimize the time that PeopleTools maintains locks on single-threaded messaging tables, behind-the-scenes logic in the SavePostChange event defers sending the message until just before the commit for the transaction.

Using Metadata and the RowsetCache Class

If your application uses data that is common, used by a number of users, and yet is fairly static, you may see a performance improvement by using the RowsetCache class.

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.

Note: Non-base language users may see different performance due to language table considerations.

Copyright © 1988, 2021, Oracle and/or its affiliates.

See "Understanding a Rowset Cache" (PeopleTools 8.59: PeopleCode API Reference).

Setting MaxCacheMemory

PeopleTools stores application data in a memory cache to increase system performance. However, too large a cache can leave insufficient available memory on your system, which leads to reduced performance.

Use this setting to specify the maximum size of the memory cache. PeopleTools prunes the cache to keep it within the specified size, and places the pruned data in a disk cache instead. Because using a disk cache can also reduce performance, the default setting might not be optimal for your application. You can adjust this setting to achieve the best trade-off between speed and available memory.

See "Cache Settings" (PeopleTools 8.59: System and Server Administration).

Writing More Efficient Code

Follow these steps to write more efficient PeopleCode:

1. Declare all variables.

   One of the conveniences of PeopleCode is that you do not have to declare your variables before you use them. The variable is assigned a type of ANY, taking on the type of the value it is assigned. However, if you use this feature, you lose type-checking at compile time, which can lead to problems at runtime.

   When you validate or save PeopleCode, watch for auto-declared messages and consider adding declarations to your program.

2. Declare variable types specifically.

   Most of the time, you know a variable's type, so you should declare the variable of that type when you begin.

   For example, if you know that a particular variable is going to be an Integer value, declare it to be Integer in the first place. You can get much better runtime performance. It is particularly effective for loop control variables but, since an integer has limited range (up to 9 or 10 digits), you must use it judiciously.

3. Watch references.

   In PeopleCode function calls, parameters are passed by reference; a reference to the value is passed instead of the value itself. If you are passing a reference to a complex data structure, such as a rowset object or an array, passing by reference saves significant processing.

   Watch out for unexpected results, though. In the following code, the function Test changes the value of *&Str* after the function call.

```
Function Test(&Par as String)

&Par = "Surprise";
end-function;

Local String &Str = "Hello";
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
Test(&Str);
/* now &Str has the value "surprise" */
```

4. Put Break statements in your Evaluate statements.

   In an Evaluate statement, the When clauses continue to be evaluated until an End-evaluate or a Break statement is encountered.

   If you have an Evaluate statement with a number of When clauses, and you only expect one of them to match, put a Break statement following the likely clause. Otherwise, all the subsequent When clauses are evaluated. Your program is still correct, but it is inefficient at runtime, particularly if you have a large number of When clauses, and the Evaluate statement is in a loop.

5. Govern your state.

   One of the key features in PeopleSoft Pure Internet Architecture is that the application server is stateless. When required, the state of your session is bundled up and exchanged between the application server and the web server.

   For example, on a user interaction, the whole state, including your PeopleCode state, has to be serialized to the web server. Then, once the interaction has completed, that state is deserialized in the application server so that your application can continue.

   To improve efficiency:

   - Watch the size of PeopleCode objects that you create (strings, arrays, and so on) to make sure they are only as big as you need them to be.
   - For user interactions, you might be able to change the logic of your program to minimize the state.

     For example if you are building up a large string (a couple of megabytes) and then performing a user interaction, you might be able to change your program logic to build the string after the interaction.
   - For secondary pages that are infrequently accessed but retrieve lots of data, consider setting *No Auto Select* in the Application Designer for the grids and scroll areas on the secondary page, to prevent loading the data the secondary page when the page buffers are initially built.

     Then add the necessary Select method to the Activate event for the secondary page to load the data into the grid or scroll area.

6. Isolate common expressions.

   The PeopleCode compiler is not an optimizing compiler, unlike some current compilers for languages such as C++. For example, the PeopleCode compiler does not do common subexpression analysis. So, sometimes, if you have a complicated bit of PeopleCode that is used often, you can isolate the common expression yourself. This isolation can make your code look cleaner and make your code faster, especially if it is in a loop.

   In this example, notice how the common subexpression is broken out:

```
/*---- For this customer, setup time on B is influenced by
     *---- the machine flavors of A. */
    &r_machine = &rs(&idB.GetRecord(Record.MACHINE_INFO);
    If (&typeA = "F") And (&typeB == "U") Then
        &r_machine.SETUP_TIME.Value = 50;
    Else
```

Copyright © 1988, 2021, Oracle and/or its affiliates.

```
        &r_machine.SETUP_TIME.Value = 10;
    End-If;
```

The compiler has to evaluate each occurrence of the expression, even though it would only execute it once.

Here is another example. Notice that once &RS and &StartDate are created, they can be used repeatedly in the loop, saving significant processing time.

```
   &RS = GetRowset();
   &StartDate = GetField(PSU_CRS_SESSN.START_DATE).Value;
   For &I = 1 To &RS.ActiveRowCount
      &RecStuEnroll = &RS.GetRow(&I).PSU_STU_ENROLL;
      &Course = &RecStuEnroll.COURSE;
      &Status = &RecStuEnroll.ENROLL_STATUS;
      &PreReqStart = &RS.GetRow(&I).PSU_CRS_SESSN.START_DATE.Value;
      If &Course.Value = "1002" And
            (&Status.Value = "ENR" Or
               &Status.Value = "CMP") Then
         If &PreReqStart < &StartDate Then
            &Completed = True;
            Break;
         End-If;
      End-If;
   End-For;
```

7. Avoid implicit conversions.

   The most common implicit conversion is from a character string to a number and vice versa. You might not be able to do anything about this, but—by being aware of it—you might be able to spot opportunities to improve performance.

   In the following example, two character strings are converted into numeric values before the difference is taken. If this code were in a loop and one of the values did not change, performance would improve significantly by doing the conversion once, as the second statement illustrates.

```
   &Diff = &R1.QE_EMPLID.Value - &R2.QE_EMPID.Value;
&Original = &R1.QE_EMPLID.Value;
. . .
&Diff = &Original - &R2.QE_EMPID.Value;
```

8. Choose the right SQL style.

   In certain cases, use SQLExec, as it only returns a single row. In other cases, you could benefit greatly by using a SQL object instead, especially if you can plan to execute a statement more than once with different bind parameters. The performance gain comes from compiling the statement once and executing it many times.

   For instance, code that uses SQLExec might look like this:

```
While (some condition)
    . . .set up &Rec
    SQLExec("%Insert(:1)", &rec);
/* this does a separate tools parse of the sql and db compile
of the statement and execute each time */
End-while;
```

   The following code rewrites the previous example to use the new SQL object:

```
Local SQL &SQL = CreateSQL("%Insert(:1)");
While (some condition)
. . .Setup &Rec
&Sql.Execute(&Rec); /* saves the tools parse and db compile
on the SQL statement and the db setup for the statement */
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
end-while;
```

SQL objects also have the ReuseCursor property, which can be used for further performance gains.

See "ReuseCursor" (PeopleTools 8.59: PeopleCode API Reference).

9. Tighten up loops.

   Examine loops to see if code can be placed outside the loop.

   For example, if you are working with file objects and your file layout does not change, there is no reason to set the file layout every time you go through the loop reading lines from the file. Set the file layout once, outside the loop.

10. Set objects to NULL when they will no longer be accessed.

    Once you are finished with an object reference, especially one with a global or component scope, assign it to NULL to get rid of the object. This setting allows the runtime environment to clean up unused objects, reducing the size of your PeopleCode state.

11. Improve your application classes

    Simple properties (without get/set) are much more efficient than method calls. Be clear in your design about what needs to be simple properties, properties with get/set, and methods. Never make something a method that really should be a property.

    Analyze your use of properties implemented with get/set. While PeopleCode properties are in a sense first class properties with more flexibility in that you can run PeopleCode to actually get and set their values, make sure you actually need get and set methods. If all you have is a normal property which is more of an instance variable then avoid get/set methods. In the following example (without the strikethrough!) by having get/set for the property SomeString you have made it much more inefficient to get/set that property since every property reference has to run some PeopleCode. Often, this inefficiency can creep in when properties are designed to be flexible at the beginning and never subsequently analyzed for whether getters/setters were really needed after all.

```
class Test
...
property String SomeString get set;

end-class;get SomeString
return &SomeString;
end-get;

set SomeString
&SomeString = &NewValue;
end-set;
```

Writing More Efficient Code Examples

These examples demonstrate more efficiently written code:

- Beware of the rowset Fill method. (Or, "What not to do in a Application Engine PeopleCode step.")

  Sometimes you need to examine the algorithm you are using. The following example is a PeopleCode program that adopts this approach: read all the data into a rowset, process it row by row, and then

Copyright © 1988, 2021, Oracle and/or its affiliates.

update as necessary. One of the reasons this is a bad approach is because you lose the general advantage of set-based programming that you get with Application Engine programs.

```
Local Rowset &RS;
Local Record &REC;
Local SQL &SQL_UPDATE;

&REC_NAME1 = "Record." | SOME_AET.SOME_TMP;
&RS = CreateRowset(@(&REC_NAME1));
&LINE_NO = 1;


&NUM_ROWS = &RS.Fill("WHERE PROCESS_INSTANCE = :1 AND BUSINESS_UNIT = :2 AND T⇒
RANSACTION_GROUP = :3 AND ADJUST_TYPE = :4 ", SOME_AET.PROCESS_INSTANCE, SOME_⇒
AET.BUSINESS_UNIT, SOME_AET.TRANSACTION_GROUP, SOME_AET.ADJUST_TYPE);


For &I = 1 To &NUM_ROWS
   &REC = &RS(&I).GetRecord(@(&REC_NAME1));
   &REC.SOME_FIELD.Value = &LINE_NO;
   &REC.Update();
   &LINE_NO = &LINE_NO + 2;
End-For;
```

This code has the following problems:

- You might run out of memory in the Fill method if the Select gathers a large amount of data.
- The Fill is selecting all the columns in the table when all that is being updated is one column.

You can change this code to read in the data one row at a time using a SQL object or using a similar algorithm, but chunking the rowsets into a manageable size through the use of an appropriate Where clause.

The following are some approximate numbers you can use to see how large a rowset can grow. The overhead for a field buffer (independent of any field data) is approximately 88 bytes. The overhead for a record buffer is approximately 44 bytes. The overhead for a row is approximately 26 bytes. So a rowset with just one record (row) the general approximate formula is as follows:

memory_amount = nrows * (row overhead + nrecords * (rec overhead + nfields * (field overhead) + average cumulative fielddata for all fields))

- The following are some code examples to show isolating common expressions.

  In this example, a simple evaluation goes from happening three times to just once– `&RS_Level2(&I).PSU_TASK_EFFORT`. In addition, the rewritten code is easier to read.

  Example of code before being rewritten:

```
Local Rowset &RS_Level2;

Local Boolean &TrueOrFalse = (PSU_TASK_RSRC.COMPLETED_FLAG.Value = "N");

For &I = 1 To &RS_Level2.ActiveRowCount
   &RS_Level2(&I).PSU_TASK_EFFORT.EFFORT_DT.Enabled = &TrueOrFalse;
   &RS_Level2(&I).PSU_TASK_EFFORT.EFFORT_AMT.Enabled = &TrueOrFalse;
   &RS_Level2(&I).PSU_TASK_EFFORT.CHARGE_BACK.Enabled = &TrueOrFalse;
End-For;
```

  Example of code after being rewritten:

```
Local Boolean &TrueOrFalse = (PSU_TASK_RSRC.COMPLETED_FLAG.Value = "N");
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
For &I = 1 To &RS_Level2.ActiveRowCount
   Local Record &TaskEffort = &RS_Level2(&I).PSU_TASK_EFFORT;

   &TaskEffort.EFFORT_DT.Enabled = &TrueOrFalse;
   &TaskEffort.EFFORT_AMT.Enabled = &TrueOrFalse;
   &TaskEffort.CHARGE_BACK.Enabled = &TrueOrFalse;
End-For;
```

In the next example, the following improvements are made to the code:

Shorthand is used: &ThisRs(&J) instead of &ThisRs.GetRow(&J).

Eliminated all the autodeclared messages by declaring all the local variables. This action can improve your logic and possibly give you better performance.

Notice the integer declaration. If you know your variables will fit in an integer (or a float), then declare them that way. Runtime performance for Integers can be better than for variables declared as Number.

Fewer evaluation expressions.

Example of code before being rewritten:

```
Local Row &CurrentRow;
&TrueOrFalse = (GetField().Value = "N");
&CurrentRow = GetRow();
For &I = 1 To &CurrentRow.ChildCount
   For &J = 1 To &CurrentRow.GetRowset(&I).ActiveRowCount
      For &K = 1 To &CurrentRow.GetRowset(&I).GetRow(&J).RecordCount
         For &L = 1 To &CurrentRow.GetRowset(&I).GetRow(&J).GetRecord(&K).Fiel⇒
dCount
            &CurrentRow.GetRowset(&I).GetRow(&J).GetRecord(&K).GetField(&L).En⇒
abled = &TrueOrFalse;
         End-For;
      End-For;
   End-For;
End-For;
```

Example of code after being rewritten:

```
Local Row &CurrentRow;
Local integer &I, &J, &K, &L;

Local boolean &TrueOrFalse = (GetField().Value = "N");
&CurrentRow = GetRow();
For &i = 1 To &CurrentRow.ChildCount
/*   No specific RowSet, Record, or Field is mentioned!    */
   Local Rowset &ThisRs = &CurrentRow.GetRowset(&i);

   For &J = 1 To &ThisRs.ActiveRowCount
      Local Row &ThisRow = &ThisRs(&J);

      For &K = 1 To &ThisRow.RecordCount
         Local Record &ThisRec = &ThisRow.GetRecord(&K);

         For &L = 1 To &ThisRec.FieldCount
            &ThisRec.GetField(&L).Enabled = &TrueOrFalse;
         End-For;
      End-For;
   End-For;
End-For;
```

- Concatenating a large number of strings into a large string. Sometimes you need to do this.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The simplest approach is to do something like:

```
&NewString = &NewString | &NewPiece;
```

In itself this is not a bad approach but you can do this much more efficiently using an application class below.

```
class StringBuffer
   method StringBuffer(&InitialValue As string);
   method Append(&New As string) returns StringBuffer; // allows &X.Append("th⇒
is").Append("that").Append("and this")
   method Reset();
   property string Value get set;
   property integer Length readonly;
   property integer MaxLength;
private
   instance array of string &Pieces;
end-class;

method StringBuffer
   /+ &InitialValue as String, +/

   &Pieces = CreateArray(&InitialValue);
   &MaxLength = 2147483647; // default maximum size
   &Length = Len(&InitialValue);
end-method;

method Reset
   &Pieces.Len = 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(" | &⇒
MaxLength | ")");
   End-If;
   &Length = &TempLength;
   &Pieces.Push(&New);
   return %This;
end-method;

get Value
   /+ Returns String +/
   Local string &Temp = &Pieces.Join("", "", "", &Length);
   /* collapse array now */
   &Pieces.Len = 1;
   &Pieces[1] = &Temp; /* start out with this combo string */
   Return &Temp;
end-get;

set Value
   /+ &NewValue as String +/
   /* Ditch our current value */
   &Pieces.Len = 1;
   &Pieces[1] = &NewValue; /* start out with this string */
   &Length = Len(&NewValue);
end-set;
```

Use this code as follows:

```
Local StringBuffer &S = create StringBuffer("");

....
&S.Append(&line);
```

 Copyright © 1988, 2021, Oracle and/or its affiliates.

```
/* to get the value of string simply use &S.Value */
```

Preventing SQL Injection

The following functions and methods provide a way for SQL to be submitted to the database; they are, therefore, subject to SQL injection vulnerabilities:

- SQLExec function
- CreateSQL function
- Rowset class Select method
- Rowset class SelectNew method
- Rowset class Fill method
- Rowset class FillAppend method

Look at the following PeopleCode as an example:

```
rem Retrieve user input from the name field;
&UserInput = GetField(Field.NAME).Value;
SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='"
| &UserInput | "'", &Name, &Phone);
```

The code is meant to enable the user to type in a name and get the person's phone number. In the example, the developer expects that the user will input data such as *Smith,* in which case the resulting SQL would look like this:

```
SELECT NAME, PHONE FROM PS_INFO WHERE NAME='Smith'
```

However, if the user specified *"Smith' OR AGE > 55 --"*, the resulting SQL would look like this:

```
SELECT NAME, PHONE FROM PS_INFO WHERE NAME='Smith' OR AGE > 55 --'
```

Note the use of the comment operator (--) to ignore the trailing single quotation mark placed by the developer's code. This would allow a devious user to find everyone older than 55.

Use the following approaches to avoid SQL injection vulnerabilities:

- Where possible, avoid using string-building techniques to generate SQL.

  Note: String-building techniques cannot always be avoided. String-building does not pose a threat unless unvalidated user input is concatenated to SQL.

- Use bind variables where possible rather that string concatenation.

  The following example is vulnerable:

  ```
  SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
  &UserInput | "'", &Name, &Phone);
  ```

- Use the Quote PeopleCode function on the user input before concatenating it to SQL.

  This pairs the quotation marks in the user input, effectively negating any SQL injection attack.

Copyright © 1988, 2021, Oracle and/or its affiliates.

The following example is vulnerable:

```
SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
&UserInput | "'", &Name, &Phone);
```

This example is not vulnerable:

```
SQLExec("SELECT NAME, PHONE FROM PS_INFO WHERE NAME='" |
Quote(&UserInput) | "'", &Name, &Phone);
```

- Specify whether SQL errors appear to the user with the Suppress SQL Error setting in the PSTOOLS section of the application server configuration file. Normally, the SQL in error appears to the user in a number of messages. If you consider this a security issue, add the following line to your application server configuration file:

  ```
  Suppress SQL Error=1
  ```

  When this line is set, SQL errors do not display details; instead, they refer the user to consult the system log. The detail that was in the SQL message is written to the log file.

Related Links

"Quote" (PeopleTools 8.59: PeopleCode Language Reference)

"PSTOOLS Options" (PeopleTools 8.59: System and Server Administration)

Copyright © 1988, 2021, Oracle and/or its affiliates.

Appendix A

PeopleCode Editor Short Cut Keys

Short Cut Keys in the PeopleCode Editor

The following table lists all the short cut keys available in the PeopleCode Editor.

Note: The short cut keys for Application Designer are not listed.

Key	***Description***
Ctrl-A	Select all
Ctrl-C	Edit copy
Ctrl-F	Edit find
Ctrl-H	Edit replace
Ctrl-L	Line cut
Shift-Ctrl-L	Line delete
Ctrl-U	Selection lowercase
Shift-Ctrl-U	Selection uppercase
Ctrl-V	Paste
Backspace	Backspace and delete characters
Alt-Backspace	Edit undo
Ctrl-Backspace	Delete to start of word
Shift-Alt-Backspace	Edit redo
Delete	Delete
Ctrl-Delete	Delete to next word
Shift-Delete	Edit cut
↓ (down arrow)	Line down
Ctrl-↓	Scroll window down one line
Shift-↓	Line down with selection

Copyright © 1988, 2021, Oracle and/or its affiliates.

Key	*Description*
End	Position cursor at end of line
Ctrl-End	Position cursor at end of file
Shift-End	Select to end of line
Shift-Ctrl-End	Select to end of file
Enter	New line
Esc (escape)	Clear selection
F2	Next bookmark
Ctrl-F2	Toggle bookmark off and on
Shift-F2	Previous bookmark
Shift-Ctrl-F2	Remove all bookmarks
F3	Find next
Shift-F3	Find previous
F5	Go (Debug)
F8	Step (Debug)
F9	Toggle debug breakpoint
Atl-F9	Edit breakpoints (Debug)
Ctrl-F9	Break at start (Debug)
F10	Step over (Debug)
Home	Position cursor to first character of line
Ctrl-Home	Position cursor to start of file
Shift-Home	Select to start of line
Shift-Ctrl-Home	Select to start of file
Insert	Toggle insert mode
Ctrl-Insert	Copy
Shift-Insert	Paste
← (left arrow)	Position cursor left one character
Ctrl-←	Position cursor left one word

Copyright © 1988, 2021, Oracle and/or its affiliates.

Key	*Description*
Shift-←	Select one character left of cursor
Shift-Ctrl-←	Select next word left of cursor
Page Down	Page down
Page Up	Page up
→ (right arrow)	Position cursor right one character
Ctrl-→	Position cursor right one word
Shift-→	Select one character right of cursor
Shift-Ctrl-→	Select next word right of cursor
Tab	Tab
Shift-Tab	Back tab
↑ (up arrow)	Line up
Ctrl-↑	Scroll window up one line
Shift-Ctrl-W	Select word
Ctrl-X	Edit cut
Ctrl-Y	Edit redo
Ctrl-Z	Edit undo
Shift-Ctrl-Z	Edit redo

Copyright © 1988, 2021, Oracle and/or its affiliates.

Copyright © 1988, 2021, Oracle and/or its affiliates.