Agile Product Lifecycle Management

SDK Developer Guide

v9.3.0.3

Part No. E23054_01
August 2011

Oracle Copyright

Copyright © 1995, 2011, Oracle and/or its affiliates. All rights reserved.

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

U.S. GOVERNMENT RIGHTS
Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software 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 which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

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

This software and documentation may provide access to or information on 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. 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. The RMW product includes software developed by the Visigoth Software Society.

CONTENTS

Oracle Copyright ii
New in Release 9.3.0.3 xx

Introduction 1

About Agile SDK 1
SDK Components 2
Client-Side Components 2
Server-Side Components 2
SDK Architecture 2
System Requirements 3
Java Requirements 4
Java Virtual Memory Parameters (JVM) to Prevent Out of Memory Exceptions 4
Agile SDK Installation Folders 5
Checking Your Agile PLM System 5
Agile PLM Business Objects 5

Getting Started with Agile API 7

Agile API Overview 7
Types of Agile API Classes and Interfaces 7
Network Class Loading 8
Single-Threaded versus Multi-Threaded Applications 9
Packaging an Agile API Program 9
Distributing Agile API Files 9
Sample Programs 9
Starting an Agile API Program 10
Setting the Class Path for the Agile API Library 10
Importing Agile API Classes 10
Creating a Session and Logging In 11
Creating a Session by Accessing a Password Protected URL 12
Creating a Session from an Agile Web Service 13
Loading and Creating Agile PLM Objects 13
Loading Objects 14
Specifying Object Types 14
Specifying Object Parameters 14
Loading Different Types of Objects 15
Creating Objects 18

Working with Agile PLM Classes 19
Creating Objects of User-Defined Subclasses 20
Using AutoNumbers 21
Setting the Required Fields 22
Creating Different Types of Objects 25
Checking the State of Agile PLM Objects 29
Propagating Values to Related Objects 29
Saving an Object to a New Object 30
Sharing an Object 30
Deleting and Undeleting Objects 31
Closing a Session 33

Creating and Loading Queries 35

About Queries 35
Creating a Query 35
Saving a Query to a Folder 36
Generating Ordered (sorted) or Unordered Query Results 37
Creating a Parameterized Query 38
Specifying Query Attributes when Creating a Query 39
Specifying Workflow Queries 40
Specifying Search Criteria 41
Search Conditions 41
Query Language Keywords 42
Specifying Search Attributes 42
Retrieving Searchable Attributes 43
Using Relational Operators 44
Using Unicode Escape Sequences 45
Using Between, Not Between, In, and Not In Operators 45
Using the Nested Criteria to Search for Values in Object Lists 46
Using Criteria Selected from Criteria Library in SDK Queries 47
Using Relationships and Content in SDK Queries 47
Searching for Words or Phrases Contained in Attachments 49
Formatting Dates in Query Criteria 49
Using Logical Operators 50
Using Wildcard Characters with the Like Operator 50
Using Parentheses in Search Criteria 51
Setting Search Criteria for Lists Containing Large Number of Objects 51
Using SQL Syntax for Search Criteria 52
Using SQL Wildcards 54
Sorting Query Results Using SQL Syntax 54
Setting Result Attributes for a Query 55
Specifying Result Attributes 60

Retrieving CTO Originator Name 61
Duplicate Results for Site-Related Objects and AMLs 62
Working with Query Results 62
Sorting Query Results 62
Query Result Datatypes 62
Managing Large Query Results 63
Query Performance 63
Creating a Where-Used Query 63
Loading a Query 64
Deleting a Query 65
Simple Query Examples 65
Working with Tables 67
About Tables 67
Retrieving a Table 68
Accessing the New and Merged Relationships Tables 69
Accessing the Relationships Table 69
Accessing the Merged Tables 69
Accessing the Merged Relationships.AffectedBy Table 69
Accessing the Merged Relationships.Affects table 70
Accessing the Merged Relationships.References Table 70
Working with Read-only Tables 71
Retrieving the Metadata of a Table 71
Adding Table Rows 71
Adding an Item to the BOM Table 72
Adding an Attachment to the Attachments Table 72
Adding a Manufacturer Part to the Manufacturers Table 72
Adding an Item to the Affected Items Table 73
Adding a Task to the Schedule Table 73
Adding and Updating Multiple Table Rows 74
Adding Multiple Items to the BOM Table 74
Updating Multiple BOM Rows 75
Iterating Over Table Rows 76
Updating Objects in Query Results with Multiple Page Tables 77
Updating Table Rows when Iterating Large Query Results 78
Sorting Table Rows 78
Removing Table Rows 79
Retrieving the Referenced Object for a Row 80
Checking Status Flags of a Row 84
Working with Page 1, Page 2, and Page 3 85

Redlining....86
Removing Redline Changes....88
Removing Redline Changes in Bulk Mode....88
Identifying Redlined Rows and Redlined Cells....89
Using ICell.getOldValue....89
Working with Data Cells....91
About Data Cells....91
Data Types....91
Checking User's Discovery Privilege....92
Checking if the Cell is a Read-Only Cell....93
Getting Values....93
Understanding SDK Date Formats and User Preferences....95
Setting Values....95
Catching Exceptions for Locked Objects....96
Getting and Setting List Values....96
Getting and Setting Values for SingleList Cells....97
Getting and Setting Values for MultiList Cells....97
Getting and Setting Values for Cascading Lists....98
Using Reference Designator Cells....100
Working with Folders....101
About Folders....101
Using Level Separation Characters in Folder and Object Names....102
Loading a Folder....103
Creating a Folder....103
Setting the Folder Type....104
Adding and Removing Folder Elements....104
Adding Folder Elements....105
Removing Folder Elements....105
Getting Folder Elements....105
Deleting a Folder....108
Working with Items, BOMs, and AMLs....109
About Items, BOMs, and Tables....109
Working with Items....109
Getting and Setting the Revision of an Item....109
Changing the Incorporated Status of a Revision....111
Working with BOMs....112
Adding an Item to a BOM....112

Expanding a BOM 113
Copying one BOM into another BOM 114
Creating BOM-Related Product Reports 115
Redlining a BOM 117
Getting a Released Assembly Item 117
Creating a Change Order 117
Adding an Item to the Affected Items tab of a Change Order 118
Modifying the Redline BOM Table 118
Working with AMLs 119
Adding an Approved Manufacturer to the Manufacturers Table 120
Redlining an AML 121

Accessing PLM Metadata with APIName Field 123

About APIName Field 123
Assigning Names to APIName Fields 124
APIName Validation Rules 124
Accessing Metadata Using the APIName Field 125
APIs that Support the APIName Field 125
SDK APIs that Get the APIName Field 129
API Names of Root Administrator Nodes 130
API Name Examples 131

Subscribing to Agile PLM Objects 137

About User Subscriptions 137
Subscription Events 137
Subscribe Privilege 138
Subscription Notifications 138
Sending Notifications with SDK 138
Deleting Subscribed Objects 139
Getting Subscriptions for an Object 140
Modifying the Subscriptions for an Object 141
Making Attributes Available for Subscription 142
Parent and Child Attributes 143
Working with Subscription Tables 144

Managing Manufacturing Sites 147

About Manufacturing Sites 147
Controlling Access to Sites 147
Creating a Manufacturing Site 148
Loading a Manufacturing Site 148
Retrieving the Sites Table for an Item 149

Adding a Manufacturing Site to the Sites Table 149
Selecting the Current Manufacturing Site for an Item 150
Disabling a Site 152

Working with Lists 153

About Lists 153
List Library 153
SingleList Lists 154
Cascading Lists 155
MultiList Lists 156
Methods that Use IAgileList 156
Selecting a List Value 157
Working with Lifecycle Phase Cells 159
Working with Dynamic Lists 160
Enumerable and Non-Enumerable Lists 160
Non-Enumerable PG&C Lists 161
Selecting a List from the List Library 161
Creating Custom Lists 163
Creating a Simple List 163
Automatically Creating New Lists by Modifying Existing Lists 164
Creating a Cascading List 165
Checking the Data Type of a List 167
Modifying a List 168
Adding a Value to a List 168
Making List Values Obsolete 169
Setting the List Name and Description 169
Setting Level Names for a Cascading List 170
Enabling or Disabling a List 170
Deleting a List 170
Modifying and Removing List Values 171
Printing Contents of IAgileList Objects 172

Working with Attachments and File Folder Objects 173

About Attachments and File Folders 173
Working with File Folders 174
File Folder Classes and Subclasses 174
File Folder Tables and Constants 175
Creating File Folder Objects 175
Creating File Folder Objects by Adding Rows to Attachments Table 178
Working with the Files Table of a File Folder 178

Accessing Files in Agile PLM File Vault with IAttachmentFile 179
Working with Attachments Table of an Object 180
Checking In and Checking Out Files with ICheckoutable 181
Specifying the Revision of the Item 181
Checking whether the Revision Is Incorporated 182
Checking Out a File Folder 183
Canceling a File Folder Checkout 183
Adding Files and URLs to the Attachments Table 184
Deep Cloning Attachments and Files from One Object to Another 186
Specifying the File Folder Subclass When Adding Attachments 187
Retrieving Attachment Files 188
Deleting Attachments and File Folders 189
Working with Thumbnails 189
Accessing Thumbnails 190
Regenerating Thumbnails 190
Setting Master Thumbnails 191
Replacing Thumbnails 191
Sequencing Thumbnails 192
Generating Thumbnails while Adding Files to Attachments Tab 193
Working with Design Objects 193
Adding and Loading Design Objects 193
Managing Version Specific Relationships between Design Objects 193
Adding Relationships for Specific Versions of Design Objects 194
Removing Relationships for Specific Versions of Design Objects 194
Getting Relationships for Specific Versions of Design Objects 194
Editing Relationships for Specific Versions of Design Objects 195
Purging Specific Versions of Design Objects 195
Searching Design Object Deployments with Where-Used Queries 195

Importing and Exporting Data with SDK 197

About Importing and Exporting Data 197
Validating Import Data and Importing Data 197
Validating Data and Importing Data with SDK 197
Exporting Data from the SDK 200
Invoking SDK's Export Function 200

Managing Workflow 203

About Workflow 203
The Change Control Process 203
Dynamics of Workflow Functionality 204
How the Status of a Change Affects Workflow Functionality 204
How User Privileges Affect Workflow Functionality 204
Selecting a Workflow 205

Adding and Removing Approvers 206
Setting the "Signoff User Dual Identification" Preference 208
Approving a Routable Object 208
Rejecting a Routable Object 209
Adding User Groups of Approvers and Users to Approve Routable Objects 210
Approving a Routable Object by Users on behalf of "Transferred from Users" 212
Adding Active Escalations for the Current User to Approve a Routable Object 212
Specifying a Second Signature to Approve a Routable Object 213
Adding User ID as Second Signature to Approve a Routable Object 215
Approving or Rejecting Change 215
Commenting a Change 216
Auditing a Change 217
Changing the Workflow Status of an Object 218
Sending an Agile Object to Selected Users 220
Sending an Agile Object to User Groups 221

Managing and Tracking Quality 223

About Quality Control 223
Quality-Related API Objects 223
Quality-Related Roles and Privileges 224
Working with Customers 224
About Customers 224
Creating a Customer 224
Loading a Customer 225
Saving a Customer as Another Customer 225
Working with Product Service Requests 226
About Problem Reports 226
About Nonconformance Reports 226
Creating a Product Service Request 226
Assigning a Product Service Request to a Quality Analyst 227
Adding Affected Items to a Product Service Request 227
Adding Related PSRs to a Product Service Request 228
Working with Quality Change Requests 229
Creating a Quality Change Request 229
Assigning a Quality Change Request to a Quality Administrator 230
Saving a Quality Change Request as a Change 230
Using Workflow Features with PSRs and QCRs 231
Selecting Workflows for PSRs and QCRs 231

Creating and Managing Projects 233

About Projects and Projects Objects 233
Differences in the Behavior of Projects Objects 234

Creating Projects 234
Adding Rules for PPM Objects 236
Loading Projects 237
Using Projects Templates 237
Creating New Projects Using Templates 237
Creating Projects and Changing Ownerships 238
Saving Projects as Templates 239
Scheduling Projects 240
Working with Projects Baselines 243
Delegating Ownership of a Projects to Another User 244
Adding Resources to a Projects Team 244
Substituting Projects Resources 247
Locking or Unlocking Projects 248
Working with Discussions 248
Creating a Discussion 249
Replying to a Discussion 250
Joining a Discussion 253
Creating an Action Item 253

Working with Product Cost Management 255

Overview 255
Working with Price Objects 256
Managing Pricing 256
Creating a Price Object 257
Defaults 257
Specifying Item Revision 257
Creating a Published Price 258
Loading a Price Object 258
Adding Price Lines 258
Creating a Price Change Order 260
Working with Suppliers 261
Loading a Supplier 261
Modifying Supplier Data 261
Working with Sourcing Projects 262
Supported API Methods 263
Loading an Existing Sourcing Projects 264
Creating Sourcing Projects by Quantity Breaks 264
Creating Sourcing Projects by Quantity Breaks and Price Periods 265
Accessing and Modifying Objects, Tables, and Attributes 266

Setting Cover Page Values for Sourcing Projects ... 266
Understanding Nested Tables in PCM ... 267
Sourcing Projects' Parent Table and Nested Child Table Constants ... 267
Accessing and Modifying Nested Tables in Sourcing Projects or RFQ ... 268
Accessing and Modifying the Status of Sourcing Projects ... 268
Managing Data in Sourcing Projects ... 269
Setting Quantity for Items in Sourcing Projects ... 269
Performing Quantity Rollup in Sourcing Projects ... 270
Performing Cost Rollup in Sourcing Projects ... 270
Performing Price Lookup in Sourcing Projects ... 271
Setting Partners in a Sourcing Projects ... 276
Modifying the Target Price for Items in Sourcing Projects ... 277
Setting the Best Response for Items in Sourcing Projects ... 278
Working with RFQs ... 279
Supported API Methods ... 280
Creating RFQs for Sourcing Projects ... 280
Loading Existing RFQs ... 281
Loading RFQs from Sourcing Projects' RFQ Table ... 281
Accessing and Modifying RFQ Objects, Tables, Nested Tables, and Attributes ... 282
Performing Price Lookup in RFQs ... 282
Working with RFQ Responses ... 283

Managing Product Governance & Compliance ... 287

About Agile Product Governance and Compliance ... 287
Agile PG&C Interfaces and Classes ... 288
Agile PG&C Roles ... 288
Creating Declarations, Specifications, and Substances ... 289
Creating Declarations ... 289
Creating Specifications ... 290
Creating Substances ... 291
Creating a Subpart ... 291
Creating a Substance Group ... 292
Creating a Material ... 292
Creating a Substance ... 293
Adding Items, Manufacturer Parts, and Part Groups to Declarations ... 293
Adding Substances to Declarations ... 294
Structure of Bill of Substances ... 295
Rules for Adding Substances ... 296
Adding Subparts and Materials that Do Not Exist ... 296
Adding Examples to Substances ... 297
Adding Substances to Manufacturer Part Composition Table of Homogeneous Material Declarations ... 297
Adding Substances to Manufacturer Part Composition Table of Substance Declarations ... 299
Adding Substances to a Specification ... 300
Adding Specifications to a Declaration ... 301
Rules for Adding Specifications ... 301

Routing Declarations 302
Completing a Declaration 303
Submitting Declarations to Compliance Managers 304
Publishing a Declaration 304
Getting and Setting Weight Values 305
Adding Substance Compositions for Manufacturer Parts 306
Rolling Up Compliance Data 308
Understanding the IPGCRollup Interface 309
Passing the Date Parameter 309
Using the IPGCRollup Interface 310
Rolling Up Assembled Data on Items 310
Rolling Up Assembled Data on MPNs 311
Setting Values in the Calculated Compliance Field for Item Objects 311
Setting Values in the Calculated Compliance Field for Declaration Objects 312

Handling Exceptions 313

About Exceptions 313
Exception Constants 314
Getting Error Codes 314
Disabling and Enabling Error Codes with Bulk APIs 314
Getting Error Messages 315
Disabling and Enabling Warning Messages 316
Checking if APIException is Warning and not Error 317
Saving and Restoring State Enabled and Disabled Warnings 317
Deleting Warnings Automatically Disabled by Agile API 318

Performing Administrative Tasks 319

About Agile PLM Administration 319
Privileges Required to Administer Agile PLM 320
Administrative Interfaces 320
Getting an IAdmin Instance 321
Working with Nodes 322
Working with the Classes Node 326
Managing Agile PLM Classes 326
Concrete and Abstract Classes 328
Referencing Classes 329
Identifying the Target Type of a Class 330
Working with Attributes 330
Referencing Attributes 331
Retrieving Attributes 332
Retrieving Individual Attributes 333

Editing the Property of an Attribute 333
Working with User-Defined Attributes 333
Working with Properties of Administrative Nodes 334
Managing Users 335
Getting All Users 335
Creating a User 335
Creating a Supplier User 336
Saving a User to a New User 337
Checking for Expired Passwords 337
Configuring User Settings 338
Resetting User Passwords 339
Deleting a User 339
Managing User Groups 340
Getting All Users Groups 340
Creating a User Group 340
Adding a User Group to the User's User Group Table 342
Listing Users in a User Group 342
Developing Process Extensions 343
About Process Extensions 343
Developing Custom Autonumber Sources 344
Defining a Custom Autonumber Source 344
Packaging and Deploying a Custom Autonumber Source 345
Configuring Custom Autonumber Sources in Java Client 346
Assigning Autonumber Sources to a Subclass 347
Developing Custom Actions 347
Defining a Custom Action 347
Formatting New Lines (Line Breaks) in PLM Clients 348
Custom Actions and User Sessions 348
Packaging and Deploying a Custom Action 349
Roles and Privileges for Custom Actions 349
User Privileges for Configuring Process Extensions 350
Configuring Custom Actions in Agile Java Client 350
Using the Process Extension Library 350
Assigning Process Extensions to Classes 352
Assigning Process Extensions to Workflow Statuses 352
Defining and Deploying URL-Based Process Extensions 353
Before Building a URL-Based Process Extension 354
Defining a URL-Based Process Extension 354
Passing Encoded Agile PLM Information to Other Applications 355
Creating an Agile PLM Session from the Target System 355

Retrieving an Agile PLM Object from an HTTP Request....357
Identifying Attributes for Agile PLM Classes....357
Creating an External Report....359
Deploying Process Extensions in a Clustered Environment....360
Best Practices for Copying third Party JAR Files....360
Process Extensions FAQ....363

Developing Web Service Extensions....367

About Web Service Extensions....367
Key Features....368
WSX Architecture....368
About Web Services....369
Web Services Architecture....370
Security....370
Tools....371
Finding Additional Information About Web Services....371
Developing and Deploying a Web Service....372
About Deployment Descriptors....372
Reserved Web Service Names....373
Using a Web Service....373
Defining a Web Service Entry Point....373
Authenticating Users....374
Using Single Sign-On Cookies for Client-Server Access....374
Deployment Architecture....375
Invoking the Web Service Client with a Single Sign-on Cookie....375
Retrieve the Single Sign-On Cookie....375
Modifying the SOAP Binding Stub Code....375
Preparing Environment for MyFirstWebService....376
Downloading Tools to Build the Sample....377
Installing the Java SDK....377
Installing Ant....377
Building MyFirstWebService Sample....378
About Web Service Clients....379
Client Programming Languages....379
Accessing a Web Service....380
Creating MyFirstClient....380
Generating the SOAP Request....380
Submitting the SOAP Request....381
Processing the SOAP Response....381
Running MyFirstClient....382

Creating an Agile Session inside WSX....382
Microsoft .NET Interoperability....382
Web Service Extensions FAQs....383
Developing Dashboard Management Extensions....387
About Dashboard Management Extensions....387
Roles and Privileges in Dashboard Management Extensions....388
Developing Custom Chart Dashboard Management Extensions....388
Understanding ChartDataModel and ChartDataSet....388
Defining a Custom Chart DX Data Source....388
Packaging and Deploying a Custom Chart DX Source....390
Configuring Chart DXs in Java Client....390
Displaying Optional Tabs in Agile Web Client....391
Developing Custom Table Dashboard Management Extensions....392
Understanding Collection and CustomTableConstants....392
Defining a Custom Table DX Data Source....393
Packaging and Deploying a Custom Table DX Source....396
To package and deploy a Table DX source:....396
Configuring Table DXs in Java Client....396
To Add a Table to a Tab:....396
To Add Data to Tables:....397
Defining Custom (URL) Extensions....398
Agile PLM Events and Event Framework....399
Understanding Agile PLM Events and Event Framework....399
Key Components of an Agile PLM Event....399
Event Types....400
Event Handler and Handler Types....401
Event Subscribers....401
Event Trigger and Trigger Types....401
Synchronous and Asynchronous Execution Modes....402
Event Error Handling Rule....402
Event Order....402
Event Generation, Processing, and Execution....403
Working with Event Context Objects....405
Understanding Event Context Objects....405
Persistent and Transient Data....405
Event Information Objects....406
Event Script Objects....408
Working with Event Information and Event Script Objects....409
Working with Base Event Actions....409

Base Event Information Object - Java PX 409
Base Event Script Objects - Script PX 411
Working with General Object Actions 412
General Object Actions - Java PX 412
Working with General Base Event Script Objects 415
Working with Table and Relationship Actions 418
Table and Relationship Actions - Java PX 418
Table and Relationship Actions - Script PX 420
Working with Workflow Object Actions 422
Workflow Object Actions - Java PX 422
Workflow Object Actions - Script PX 425
Working with Specific Object-Based Actions 428
Specific Object-Based Actions - Java PX 428
Specific Object-Based Actions - Script PX 428
Working with Files and Attachments Objects Actions 429
Files and Attachments Objects Actions - Java PX 429
Files and Attachments Objects Actions - Script PX 430
Working with Product Governance and Compliance Actions 431
Product Governance and Compliance Actions - Java PX 431
Product Governance and Compliance Actions - Script PX 431
Working with Miscellaneous Object Actions 431
Miscellaneous Object Actions - Java PX 431
Miscellaneous Object Actions - Script PX 432
Working with Event Integration Points in PLM Clients 432
Event Integration Points - Java PX 432
Event Integration Points - Script PX 432
Guidelines for Java PX and Script PX Handlers 433
Working with Agile PLM Administrator 433
Testing Event Java PX and Event Script PX 434
Triggering Guidelines for Java PX, Script PX, and Notification Handlers 434
General Object Actions 435
Workflow Actions 435
Files and Attachments Actions 436
Event FAQs 437

Mapping Agile PLM Client Features to Agile API 441

Login Features 441
General Features 442
Search Features 442
Attachment Features 443
Workflow Features 443
Manufacturing Site Features 444
Folder Features 444
Projects Features 445
Administrative Features 445

Migrating Release 9.2.1 and Older Table Constants to Release 9.2.2 or Later 447

Mapped Pre-Release 9.2.2 Table Constants to 9.2.2 Table Constants..... 447
Removed Pre-Release 9.2.2 Table Constants..... 450

Migrating Custom Process Extensions to Event Framework 451

About this Appendix 451
Understanding Custom PXs and Java PXs..... 451
Custom PXs in PX Framework.....451
Process Extensions in Event Framework.....452
Custom PXs You Can Migrate to Event Framework 452
Migration Task List 452
Task - 1: Modify the Custom PX Code 452
Custom PX Code..... 453
Java PX Code..... 453
Task - 2: Package and Deploy the Modified Code 454
Task - 3: Configure Event in Event Framework.....454
Create Event..... 454
Create Event Handler 456
Create Event Subscriber 457
Configure Trigger Type, Execution Mode, Order, and Error Handling Rule 459
Task - 4: Test the Migrated PX in Event Framework.....460
Task - 5: Remove Custom PX from Process Extension Library.....460
Task - 6: Inform PLM Administrator.....460

Groovy Implementation in Event Framework..... 461

About this Appendix 461
What is Groovy?..... 461
Sources of Information 461
Script PX or Java PX?.....462
Event Framework Implementation..... 462
Key implementation considerations.....462
Starting a Script.....463
Accessing SDK with Scripts 463
Use Cases.....463

Preface

Oracle's Agile PLM documentation set includes Adobe® Acrobat PDF files. The Oracle Technology Network (OTN) Web site http://www.oracle.com/technetwork/documentation/agile-085940.html contains the latest versions of the Agile PLM PDF files. You can view or download these manuals from the Web site, or you can ask your Agile administrator if there is an Agile PLM Documentation folder available on your network from which you can access the Agile PLM documentation (PDF) files.

Note	To read the PDF files, you must use the free Adobe Acrobat Reader version 9.0 or later. This program can be downloaded from the Adobe Web site http://www.adobe.com.

The Oracle Technology Network (OTN) Web site http://www.oracle.com/technetwork/documentation/agile-085940.html can be accessed through Help > Manuals in both Agile Web Client and Agile Java Client. If you need additional assistance or information, please contact My Oracle Support (https://support.oracle.com) for assistance.

Note	Before calling Oracle Support about a problem with an Agile PLM manual, please have the full part number, which is located on the title page.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Readme

Any last-minute information about Agile PLM can be found in the Readme file on the Oracle Technology Network (OTN) Web site http://www.oracle.com/technetwork/documentation/agile-085940.html.

Agile Training Aids

Go to the Oracle University Web page http://www.oracle.com/education/chooser/selectcountry_new.html for more information on Agile Training offerings.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

New in Release 9.3.0.3

There are no new features in the current release of the SDK Developer Guide.

New in Release 9.3.0.2

The following new features and enhancements were implemented in this release:

- Checking for expired passwords – The example in Managing Users on page 335 that checks for Agile API errors related to expired passwords is changed in this release. See Checking for Expired Passwords on page 337.
- Improving query performance for lists containing large number of objects – A new section that shows how to set the serach criteria to improve performance for lists containing large number of objects is added in this release. See Setting Search Criteria for Lists Containing Large Number of Objects on page 51.
- Deploying dependent/shared libraries in Weblogic server – Applicable procedures to deploy the dependent JAR files are modified in this release. See Deploying the Dependent JAR Files on page 360.
- Creating a Session and Logging In – A note describing the settings of JVM parameter when LDAP users log in to Agile PLM running on Weblogic servers. See Creating a Session and Logging In on page 11.

New in Release 9.3.0.1 Rev 2

The SiteMinder and SAP portal references that appeared in Using Single Sign-On Cookies for Client-Server on page 374 are removed in this revision.

New in Release 9.3.0.1

New features and enhancements implemented in this release address using the SDK to create and load queries and work with tables. These changes, summarized below, are documented in Chapter 3, Creating and Loading Queries on page 35 and Chapter 4, Working with Tables on page 67.

- SDK support for workflow-related queries – This release supports using workflow attributes to prepare queries that check the status of a workflow in the PLM process. See Specifying Workflow Queries on page 40.
- SDK support to use the Nested Criteria to perform object list searches – This release provides an enhancement in the behavior and performance of this feature. See Using the Nested Criteria to Search for Values in Object Lists on page 46.
- SDK support for Relationships/Content queries – As a result of enhancements made to the `IQuery` interface, you can use the SDK to prepare queries listed below. For information and usage examples, see Using Relationships and Content in SDK Queries on page 47.
  - An object's Relationships attribute
  - A Program object's Content attribute
  - A Transfer Order object's Selected Content attribute

  For information, procedures, and examples, see Using Relationships and Content in SDK

Queries on page 47.

- SDK support to query using criterion selected from PLM's Criteria Library – ThIs feature was supported in earlier releases of the SDK. In this release, the examples is modified. See Using Criteria Selected from Criteria Library in SDK Queries on page 47.
- SDK support for bulk removal of redline changes – A new interface called `IRedlinedTable` supports this operation, See Removing Redline Changes in Bulk Mode on page 88.

Note The PG&C constants and the relationships table functionality in `AgileAPI.jar` Release 9.2.2 and subsequent releases of Agile PLM are incompatible with the ones in the earlier versions of `AgileAPI.jar`.

Chapter 1

Introduction

This chapter includes the following:

- About Agile SDK 1
- SDK Components 2
- SDK Architecture 2
- System Requirements 3
- Java Requirements 3
- Agile SDK Installation Folders 5
- Checking Your Agile PLM System 5
- Agile PLM Business Objects 5

About Agile SDK

The Agile software development kit (SDK) is a collection of Java application programming interfaces (APIs), sample applications, and documentation that enable building custom applications to access, or extend the Agile Application Server functionalities. By using the Agile SDK, you can create

programs that perform tasks against the Agile PLM system.

The Agile SDK enables the following operations:

- Integrate the Agile PLM system with enterprise resource planning (ERP) applications or other custom applications
- Develop applications to process product data
- Perform batch operations against the Agile Application Server

The Agile SDK includes the following modules:

- Agile API – A Java API with interfaces that expose Agile business objects. The Agile API is used to create additional Agile PLM clients, or used as part of an extension developed using Java Script, Web Services, PX, sPX, or WSX defined below.
- Process extensions (Custom PXs) – This a framework that allows Agile PLM customers to extend the functionality of Agile PLM Clients by adding external reports, user-driven and Workflowdriven custom actions, custom tools, and custom autonumber sources. These PXs are implemented using the Java programming language.
- Event – This framework supports Java process extensions (Java PXs), and Script process extensions (Script PXs). Similar to Custom PXs, they make it easier for Agile PLM customers to extend the functionality of Agile PLM Clients to manage events. Java PXs are implemented using Java and Script PXs are implemented using a scripting language called Groovy. Groovy is an object-oriented programming language for the Java Platform as an alternative to the Java programming language.

- Web service extensions (WSX) – This is a framework that enables Agile PLM customers to extend the functionality of the Agile PLM server and expose the customer-specific solution as a Web service.

SDK Components

The Agile SDK has the following Client-side and sever-side components:

Client-Side Components

The contents of the Agile SDK Client-side components are:

Documentation

- *SDK Developer Guide* (this manual)
- API Reference files (these are the Javadoc generated HTML files that document the API methods)
- Sample applications

Note The API HTML reference files and Sample applications are in the SDK_samples.zip folder. This folder is found in the Oracle Agile PLM's Event and Web Services Samples Web site https://www.samplecode.oracle.com/tracker/tracking/linkid/prpl1004/remcurreport/true/template/IssueList.vm/action/Search?eventSubmit_doSelectquery=foo&go=224. For more information and procedures to access its contents, contact your system administrator, or refer to your PLM installation guide.

Installation

- Agile API library (`AgileAPI.jar`)
- Java Process Extensions API library (`pxapi.jar`)
- Apache Axis library (`axis.jar`)

Server-Side Components

Oracle's Agile Application Server contains the following SDK server-side components:

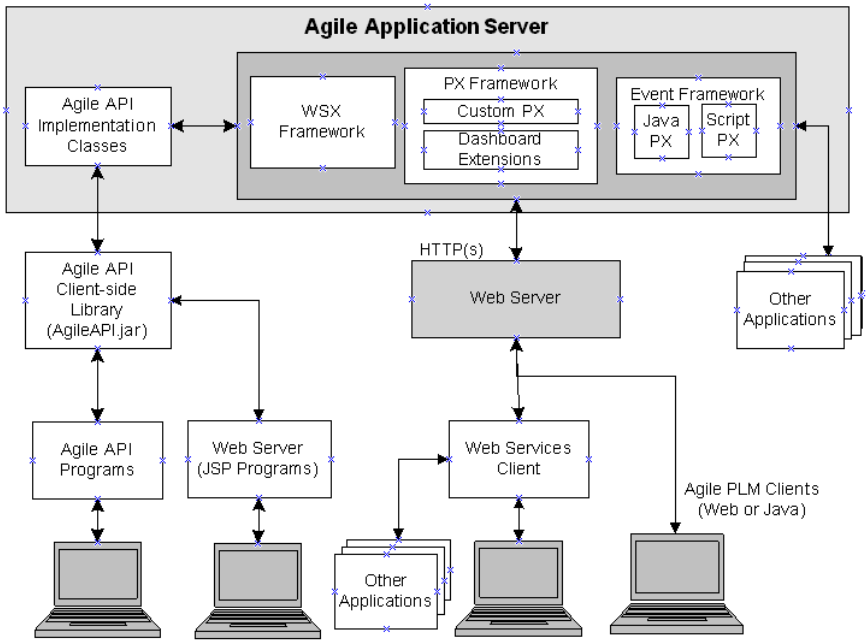
- Agile API implementation classes
- Java and Scripting process extensions framework
- Web service extensions frameworks

SDK Architecture

The SDK facilitates developing many types of programs to connect to the Agile Application Server.

If you are using only the Agile API, you can develop programs that connect directly to the server. If you are using WSX to develop Web service extensions, you can deploy the Web services inside the Agile Application Server container. You can locate the Web server used for WSX either inside or outside the company's demilitarized computing zone (DMZ) or perimeter network. When the Agile PLM Client initiates a custom action, it either runs a program deployed on the server, or it connects to an external resource or URL, WSX, Java PX, and Script PX extensions can also use the Agile API. It's a tool available to all Agile SDK development projects. You can also develop extensions using APIs that are not provided by Agile.

Figure 1: Agile SDK architecture



Note Agile API programs connect to the Agile Application Server using non-secure means. Consequently, Agile API programs should be run only from within the corporate firewall. Web service Clients, however, can connect to the server through the corporate *firewall* using standard HTTP(S) technology.

System Requirements

For Agile SDK system requirements, refer to *PLM Capacity Planning and Deployment Guide*.

Java Requirements

The Agile API must be compatible with the version of Java that the application server supports. To avoid problems, an Agile API Client must use the same version of Java that the connecting application server is using. Oracle Application Server 10g must use Sun Java Runtime Environment (JRE) 1.5.0_06 and Oracle WebLogic Server 10.3 must use Sun Java Runtime Environment (JRE) 1.6 for interoperability and 2007 Daylight Saving Time compliance.

The following table lists the recommended Java Runtime Environment (JRE) to use with Agile API Clients on different application servers that Agile PLM supports.

Application Server	Operating System	Required Java Version for Agile API clients
Oracle Application Server 10g	Windows 2003	Sun JRE 1.5.0
Oracle WebLogic Server 10.3	Windows 2003	Sun JRE 1.6

Java Virtual Memory Parameters (JVM) to Prevent Out of Memory Exceptions

To prevent out of memory errors, add the following JVM options in the indicated locations.

Note This workaround is only applicable to single-threaded SDK programs.

- If the Client is a standalone SDK Client, add the JVM option as shown below:
  `java -Ddisable.agile.sessionID.generation=true pk.sample`
- If the Client is a PX and out of memory occurs in Agile Server, add the JVM option in `<OAS_HOME>/opmn/conf/opmn.xml`

```
<category id="start-parameters">
          <data id="java-options" value="-Xrs -server -
XX:MaxPermSize=256M -ms1280M -mx1280M -XX:NewSize=256M -
XX:MaxNewSize=256M -XX:AppendRatio=3 -
Doracle.xdkjava.compatibility.version=10.1.0 -
Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy  -
Dagile.log.dir=$ORACLE_HOME/j2ee/home/log -
Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.port=9899 -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -Djava.awt.headless=true -
Dhttp.webdir.enable=false -Duser.timezone=GMT -
Ddisable.agile.sessionID.generation=true"/>
         <data id="oc4j-options" value="-verbosity 10 -
userThreads"/>
</category>
```

- If the Client is a URL PX, add the following JVM option in the Server Start up (similar to `catalina.bat` in Tomcat):

```
-Ddisable.agile.sessionID.generation=true
```

Agile SDK Installation Folders

The Agile SDK files use the following folder structure on your computer:

`lib` – The `\agile_home\integration\sdk\lib` folder contains the following libraries:

Important Do not include the `axis.jar` file and AgileAPI.jar file in the same `classpath`. The SDK classpath does not support this setting and the SDK will not function properly.

- `AgileAPI.jar` – Agile API library, which contains Agile API classes and interfaces
- `axis.jar` – An Oracle-modified version of the Apache Axis library required for Web service Clients
- `pxapi.jar` – PX API library, which contains interfaces used to develop custom autonumber sources and custom actions

Checking Your Agile PLM System

Before trying to run the Agile SDK Clients on your Agile PLM system, make sure the system is configured and working properly. In particular, make sure the HTTP ports for your application server are set correctly. For more information, refer to the *Agile PLM Installation Guide*.

Agile PLM Business Objects

With any enterprise software system, you work with business objects to manage the company's data. The following table lists the Agile PLM business objects and their related Agile API interfaces.

Object	Related Agile API Interface
Changes	`IChange`
Customers	`ICustomer`
Declarations	`IDeclaration`
Design	`IDesign`
Discussions	`IDiscussion`
File Folders	`IFileFolder`
Items	`IItem`
Manufacturer parts	`IManufacturerPart`
Manufacturers	`IManufacturer`
Packages	`IPackage`
Part Groups (Commodity or Part Family)	`ICommodity`

Object	Related Agile API Interface
Prices	IPrice
Product Service Request	IServiceRequest
Projects	IProgram
Sourcing Project	IProject
Quality Change Request	IQualityChangeRequest
Reports	IProductReport
Requests for Quote (RFQ)	IRequestForQuote
RFQ Responses	ISupplierResponse*
Sites	IManufacturingSite
Specifications	ISpecification
Substances	ISubstance
Suppliers	ISupplier
Transfer Order	ITransferOrder
User Groups	IUserGroup
Users	IUser

* Agile does not support the API interfaces in the current release of the software.

The business objects that you can view and actions that you can perform on these objects are determined by the server components installed on your Agile Application Server and the roles, and assigned privileges. Privilege levels can vary from field to field. In addition to Users and User Groups, Agile PLM administrators work with administrative objects, such as administrative nodes and Agile PLM classes.

Note Not all Agile PLM business objects are exposed in the Agile API. For example, some Report objects are not accessible via the Agile API.

Chapter 2

Getting Started with Agile API

This chapter includes the following:

- Agile API Overview 7
- Starting an Agile API Program 10
- Loading and Creating Agile PLM Objects 13

Agile API Overview

This section provides an overview of the functionality provided by the Agile API. Topics covered include:

- Types of Agile API classes and interfaces
- Loading Agile API classes
- How the Agile API is thread-safe
- Packaging your Agile API applications
- Finding the sample programs

Types of Agile API Classes and Interfaces

The Agile API contains several different classes and interfaces in the `AgileAPI.jar` library. These files are further classified into the following groups according to functions that they support:

- Aggregate interfaces – These interfaces aggregate the applicable functional interfaces for a particular object type. For example, the `IItem` interface extends `IDataObject, IRevisioned, IManufacturingSiteSelectable, IAttachmentContainer, IHistoryManager`, and `IReferenced`. Most SDK functionalities fall within these interfaces. The Agile API's underlying implementation classes, which are not exposed, implement these interfaces.
- Functional Unit Interfaces – These interfaces hold units of functionality that are extended to other interfaces. For example, `IAttachmentContainer` provides a convenient way to access the attachments table for any object. Other interfaces in this group such as `IChange` and `IItem` extend the `IAttachmentContainer` interface. `IRoutable` is another class that serves as a functional unit; it provides methods for any object that you can route to another Agile PLM user; `IChange`, `IPackage`, and `ITransferOrder` all extend `IRoutable`.
- Metadata interfaces – This group of classes defines the metadata (and meta-metadata) for the Agile Application Server. Metadata is simply data that describes other data. The metadata interfaces include classes such as `IAgileClass, INode, IRoutableDesc, ITableDesc`, and `IWorkflow`.

- Factory classes – `AgileSessionFactory` is a factory class that is used to create a session (`IAgileSession`) and access transaction management. `IAgileSession` is also a factory object allowing you to instantiate other objects. Many Agile API objects, in turn, are factory objects for tables or other referenced objects. Tables, in turn, are factories for rows.
- Exception classes – There's only one Exception class, `APIException`.
- Constants – These classes contain IDs for attributes, tables, classes, and so on. All classes containing only constants have class names that end with "Constants," for example, `ChangeConstants, ItemConstants, UserConstants`, and so on.

Network Class Loading

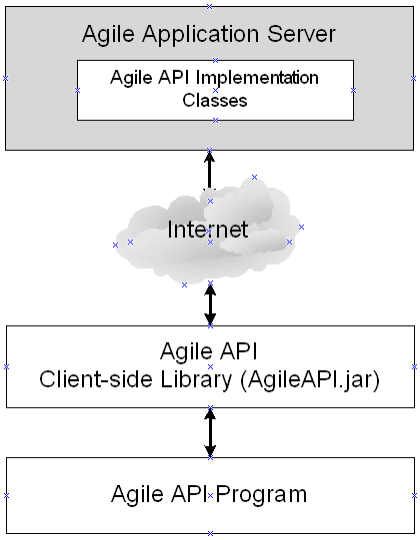
The Agile API has two main software components:

- Client-side library, `AgileAPI.jar`
- Server-side implementation classes

The server-side implementation classes are installed automatically with every instance of the Agile Application Server.

The Agile API Client-side library is composed almost entirely of interfaces; it's essentially a class loader. When you run an Agile API program, it connects to the Agile Application Server and automatically downloads whatever implementation classes it needs. For example, if your program uses methods of `IItem`, it downloads an implementation of `IItem` at run time.

Figure 2: Agile API architecture



Network class loading provides many benefits, including the ability to update client implementation classes by automatically downloading them from a server. Any Agile API classes that are downloaded from the server are automatically cached to a local disk. When an Agile API program needs to load a particular class, it retrieves it from the cache rather than downloading it again from the network. The cache results in faster loading of classes and reduced network load.

If the network class loader detects that its cache is stale, that is, its classes are older than the classes on the server, it invalidates the cache and reloads the necessary classes from the server. This allows you to update Agile SDK Clients to use the latest implementation classes without redeploying applications throughout the enterprise.

Single-Threaded versus Multi-Threaded Applications

The Agile API has been certified thread-compatible. It can be used for both Single-Threaded and Multi-Threaded application development. You can safely use Agile API calls concurrently, by surrounding each method invocation (or sequence of method invocations) with external synchronization.

Packaging an Agile API Program

After you develop a program that makes calls to the Agile API, you'll need to package its files so that you or other users can install it. Many development environments include tools for packaging and deploying applications.

You can also choose to package your program manually. If you choose to do this, you'll need to know the dependencies your project has. Again, many development environments include tools for generating dependency files. A dependency file lists the runtime components that must be distributed with your program's project files.

Distributing Agile API Files

You can freely distribute any Java applications or applets that you create that make calls to the Agile API. You can include the Agile API library, `AgileAPI.jar`, when you package your application's files.

Your development environment may require you to distribute other class files or libraries with your program. Check the documentation for your development environment to see which runtime files you must distribute with your program. Consult the applicable license agreement of the manufacturer for each of the files you plan to distribute to determine whether you are authorized to distribute the files with your application.

Sample Programs

The Agile SDK provides several sample programs that demonstrate how to use its APIs. These sample programs are in the `api`, `dx`, `px`, and `wsx` folders. You can find them in the SDK_samples (ZIP file). To access this file, see the Note in Client-Side Components on page 2.

Each sample program has its own `Readme.txt` file. Be sure to review the `Readme.txt` document before trying to run a sample program.

Starting an Agile API Program

When you create a program using the Agile API, follow this general approach for structuring your program:

1. At the top of each class file, add an import statement to import Agile API classes:

   ```
   import com.agile.api.*;
   ```

2. Get an instance of the Agile Application Server.
3. Create an Agile session.
4. Complete one or more business processes. This is where most of your program code goes.
5. Close the Agile session.

Setting the Class Path for the Agile API Library

When Java looks for a class referenced in your source, it checks the directories specified in the CLASSPATH variable. To create Agile API programs, you must include `AgileAPI.jar` in the class path.

If you are using a Java development environment, you usually can modify the class path for each project. If you don't let your development environment know where the Agile API library is located, it is not able to build the application.

Importing Agile API Classes

The only Java package your program has access to automatically is `java.lang`. To refer to Agile API classes, you should import the `com.agile.api` package at the beginning of each class file:

```
import com.agile.api.*;
```

Rather than importing the `com.agile.api` package, you can also refer to Agile API classes by their full package name, for example:

```
com.agile.api.IItem source =
(com.agile.api.IItem)m_session.getObject(com.agile.api.IItem.OBJECT_TYP
E, "1000-02");
```

As you can see, if you don't import the `com.agile.api` package, it's cumbersome to type the full package name whenever you refer to one of its classes. Also, when you don't import the `com.agile.api` package, or reference the Agile API classes by full package name, the Java compiler will return an error when you try to build your program.

Creating a Session and Logging In

Note Use the JVM parameter called `disable.agile.sessionID.generation=true` to create a session when the login user is a LDAP user. This is applicable only when Agile runs on Weblogic server. If the Client is a standalone SDK Client, add the JVM option `java -Ddisable.agile.sessionID.generation=true pk.sample`. Alternatively you can set the parameter in the code as follows:

```
System.setProperty("disable.agile.sessionID.generation", "true");
HashMap params = new HashMap();
params.put(AgileSessionFactory.USERNAME, USERNAME);
params.put(AgileSessionFactory.PASSWORD, PASSWORD);
AgileSessionFactory  factory =
AgileSessionFactory.getInstance(URL);
IagileSession session = factory.createSession(params);
```

To start an Agile API program, you must complete the following two tasks:

1. Get an instance of the Agile Application Server.

   Use the `AgileSessionFactory.getInstance()` method to get an instance of the Agile server. You must specify a connection URL for the server. The URL you specify depends on whether you connect directly to the Agile server or through a proxy Web server.

   - To connect directly to the Agile server, type this URL: http://appserver:port/virtualPath
   - To connect to the Agile server through a proxy Web server, type this URL: protocol://webserver:port/virtualPath

   where

   - *appserver* is the name of the Agile server computer.
   - *webserver* is the name of the Web server computer.
   - *virtualPath* is the virtual path for your Agile PLM server. The default value is `Agile`. The virtual path is specified when the Agile PLM system is installed. For more information, refer to the *Agile PLM Installation Guide*.
   - *protocol* is either HTTP or HTTPS.
   - *port* is the port number used for the specified protocol. The port is needed only if a nonstandard port number is being used. Otherwise, you can omit it.

2. Create a session for the Agile PLM server instance.

   Use the `AgileSessionFactory.createSession()` method to create a session. For the `params` parameter of `createSession()`, specify a `Map` object containing the login parameters (username and password).

The following example shows how an Agile API program creates a session and logs into the Agile PLM server.

Example: Creating a session and logging in

```
private IAgileSession login(String username, String password) throws
APIException {
//Create the params variable to hold login parameters
  HashMap params = new HashMap();
```

```
//Put username and password values into params
  params.put(AgileSessionFactory.USERNAME, username);
     params.put(AgileSessionFactory.PASSWORD, password);
//Get an Agile server instance. ("agileserver" is the name of the Agile
proxy server,
and "virtualPath" is the name of the virtual path used for the Agile
system.)
     AgileSessionFactory instance =
     AgileSessionFactory.getInstance
      ("http://<agileserver>/<virtualPath>");
//Create the Agile PLM session and log in
     return instance.createSession(params);
}
```

Your Oracle Agile PLM agreement determines the maximum number of concurrent open sessions to the Agile Application Server per user account. If you exceed this maximum number, the server prevents you from logging in. Therefore, it is important to use the `IAgileSession.close()` method to properly log out and close a session when your program is finished running. If the Agile PLM system is hosted on Oracle Application Servers, you are limited to only one session per thread.

Creating a Session by Accessing a Password Protected URL

To provide additional security for users accessing Agile PLM across a firewall, the proxy server may have a password-protected URL. If so, the normal method of obtaining a server instance and then creating a session will not work. Instead, you must use the `AgileSessionFactory.createSessionEx()` method to specify the username, password, and URL parameters needed to log in. The login code is simpler if you use `createSessionEx()` because you don't need to call the method `AgileSessionFactory.getInstance()` first to obtain a server instance. The `createSessionEx()` method obtains the server instance and creates the session in one call as shown in the following example.

Example: Creating a session by accessing a password-controlled URL

```
private IAgileSession securelogin(String username, String password)
throws APIException {
//Create the params variable to hold login parameters
  HashMap params = new HashMap();
//Put username, password, and URL values into params
     params.put(AgileSessionFactory.USERNAME, username);
     params.put(AgileSessionFactory.PASSWORD, password);
     params.put(AgileSessionFactory.URL,
"http://agileserver.agilesoft.com/Agile");
//Create the Agile PLM session and log in
    return AgileSessionFactory.createSessionEx(params);
}
```

The `createSessionEx()` method also works for URLs that are not password-protected, so you can use it instead of `createSession()` if you prefer.

Creating a Session from an Agile Web Service

If you developed a web service using web service extensions and deployed it in the same container as Agile PLM, you can take advantage of the Agile API to access Agile PLM server functionality from within the web service. To get an Agile PLM server instance for your web service, use the `AgileSessionFactory.getInstance()` method, but pass a `null` value for the `url` parameter.

Once you have retrieved an `AgileSessionFactory` object, you can also create a session. The web service request provides user authentication, so you don't need to specify a username or password when you create an Agile API session. Therefore, make sure you specify a `null` value for the `params` parameter of `AgileSessionFactory.createSession()`.

```
AgileSessionFactory factory = AgileSessionFactory.getInstance(null);
IAgileSession session = factory.createSession(null);
```

If you pass a null value for the `params` parameter of `createSession()`, the user authentication that took place when the Agile PLM server intercepted the web service request is reused for the Agile API session. You don't need to log in again. Do not attempt to close the session using `IAgileSession.close()`; the authorization handler will automatically close the session.

Specifying a null parameter for the `createSession()` method creates an `IAgileSession` corresponding to the session created by the authorization handler. If your web service doesn't use the authorization handler, or if you want to create a session for a different user than the one used for the authorization handler, you can still use `createSession(params)` to create a session. For the `params` parameter, specify a `Map` object containing the login parameters (username and password). If you don't use the authorization handler to create a session, you are responsible for closing it. Call the `IAgileSession.close()` method to close the session. For more information about web service extensions, see Developing Web Service Extensions on page 367.

Loading and Creating Agile PLM Objects

With every Agile API program, a basic requirement is the ability to get and create objects. The following interfaces map to objects that you can work with in the Agile API:

- `IChange`
- `ICommodity`
- `ICustomer`
- `IDeclaration`
- `IDesign`
- `IDiscussion`
- `IFileFolder`
- `IFolder`
- `IItem`
- `IManufacturer`
- `IManufacturerPart`
- `IManufacturingSite`
- `IPackage`
- `IPrice`
- `IProgram`
- `IProject`
- `IQualityChangeRequest`
- `IQuery`
- `IRequestForQuote`
- `IServiceRequest`
- `ISpecification`
- `ISubstance`
- `ISupplier`
- `ISupplierResponse`
- `ITransferOrder`
- `IUser`
- `IUserGroup`

To load and create these Agile PLM objects, you must first get an instance of the `AgileSessionFactory` object and then create an Agile PLM session. Then use `IAgileSession.getObject()` to load Agile PLM objects and `IAgileSession.createObject()` to create objects.

For more information about creating queries and folders, see Creating and Loading Queries on page 35 and Working with Folders on page 101.

Loading Objects

To load an Agile PLM object, use one of the `IAgileSession.getObject()` methods.

- `IAgileObject getObject(Object objectType, Object params)`
- `IAgileObject getObject(int objectType, Object params)`

Note If not specified by the user, objects will always load according to their base class which are derived from the subclass or class. Objects will also load correctly when the object's derived base class is correct. However, the SDK will load an object even if an invalid subclass is passed for that object when the derived base class of the invalid class and that of the object are both the same.

Specifying Object Types

The two `getObject()` methods let you specify the `objectType` parameter using these values:

- An `IAgileClass` instance that represents one of the Agile PLM classes.
- A class ID (for example, `ItemConstants.CLASS_PART` corresponds to the Part class). Predefined class IDs are available in the various *Constants files provided with the Agile API.
- An `OBJECT_TYPE` constant, such as `IItem.OBJECT_TYPE` or `IChange.OBJECT_TYPE`
- A class name (for example, “Part”). However, Oracle does not recommend using class names to instantiate objects because the class names can be modified and are not guaranteed to be unique.

Note When you use the `getObject()` method to load an object, you can specify abstract or concrete Agile PLM classes. For more information, see Concrete and Abstract Classes on page 328.

Specifying Object Parameters

The `params` parameter for the getObject() method can be a Map or String. If you specify a Map object for the params parameter, it must contain attributes (either attribute IDs or IAttribute objects) and their corresponding values. The Map must contain all identification related information. For example, when you load an IManufacturerPart, both the Manufacturer Name and Manufacturer Part Number must be specified.

If the `Map` object you specify for the `params` parameter contains additional attributes other than the identifying information, those attributes are ignored. The server uses only identifying information to retrieve an object. For a complete list of attributes used to uniquely identify Agile PLM objects, see "Identifying Attributes for Agile PLM Classes" in *SDK Developer Guide - Developing PLM Extensions*.

This example shows how to load part 1000-02 using a `Map` parameter that specifies the attribute (`ItemConstants.ATT_TITLE_BLOCK_NUMBER`) and a value.

Example: Loading a part using a Map

```
try {
 Map params = new HashMap();
 params.put(ItemConstants.ATT_TITLE_BLOCK_NUMBER,
   "1000-02");
     IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
params);
} catch (APIException ex) {
  System.out.println(ex);
}
```

If the object you're loading has a single attribute that serves as a unique identifier, you can enter the `String` value for that attribute as the `params` parameter. For example, the unique identifier for a part is a part number. Therefore, you can enter the part number as the parameter to load the object.

Note Not all objects have one attribute that serves as a unique identifier. For example, a manufacturer part is identified by both its manufacturer name and manufacturer part number. Therefore, to load a manufacturer part you must specify values for at least those two attributes.

This example shows how to load part 1000-02 by specifying a unique `String` identifier.

Example: Loading a part using a String

```
try {
     IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
"1000-02");
    } catch (APIException ex) {
  System.out.println(ex);
}
```

Loading Different Types of Objects

The following example shows how to load different types of Agile PLM objects.

```
try {
//Load a change
     IChange change = (IChange)m_session.getObject(IChange.OBJECT_TYPE,
"C00002");
     System.out.println("Change : " + change.getName());
//Load a commodity
     ICommodity comm =
(ICommodity)m_session.getObject(ICommodity.OBJECT_TYPE, "Res");
     System.out.println("Commodity : " + comm.getName());
//Load a customer
       ICustomer cust =
       (ICustomer)m_session.getObject(ICustomer.OBJECT_TYPE,
       "CUST00006");
       System.out.println("Customer : " + cust.getName());
//Load a declaration
       IDeclaration dec =
       (IDeclaration)m_session.getObject(IDeclaration.OBJECT_TYPE,
       "MD00001");
```

```
        System.out.println("Declaration : " + dec.getName());
//Load a discussion
        IDiscussion discussion =
        (IDiscussion)m_session.getObject(IDiscussion.OBJECT_TYPE,
        "D00002");
        System.out.println("Discussion : " + discussion.getName());
//Load a file folder
        IFileFolder ff =
        (IFileFolder)m_session.getObject(IFileFolder.OBJECT_TYPE,
        "FOLDER00133");
        System.out.println("File Folder : " + ff.getName());
//Load a folder
        IFolder folder =
        (IFolder)m_session.getObject(IFolder.OBJECT_TYPE, "/Personal
        Searches/MyTemporaryQueries");
        System.out.println("Folder : " + folder.getName());
//Load an item
        IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-
        02");
        System.out.println("Item : " + item.getName());
//Load a manufacturer
        Map params = new HashMap();
        params.put(ManufacturerConstants.ATT_GENERAL_INFO_NAME, "World
        Enterprises");
        IManufacturer mfr =
        (IManufacturer)m_session.getObject(IManufacturer.OBJECT_TYPE,
        params);
        System.out.println("Manufacturer : " + mfr.getName());
//Load a manufacturer part
  params.clear();
        params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURE
        R_NAME, "World Enterprises");
        params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURE
        R_PART_NUMBER, "WE10023-45");
        IManufacturerPart mfrPart =
        (IManufacturerPart)m_session.getObject(IManufacturerPart.OBJECT_T
        YPE, params); System.out.println("ManufacturerPart : " +
        mfrPart.getName());
//Load a manufacturing site
IManufacturinmporting and Exporting DtagSite siteHK =
        (IManufacturingSite)m_session.getObject(ManufacturingSiteConstant
        s.CLASS_SITE, "Hong Kong");
System.out.println("ManufacturingSite : " + siteHK.getName());
//Load a package
IPackage pkg =
(IPackage)m_session.getObject(PackageConstants.CLASS_PACKAGE,
"PKG00010");
System.out.println("Package : " + pkg.getName());
//Load a price
IPrice price =
        (IPrice)m_session.getObject(IPrice.OBJECT_TYPE, "PRICE10008");
System.out.println("Price : " + price.getName());
//Load a program
```

```
IProgram program =
       (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM10008");
System.out.println("Program : " + program.getName());
//Load a PSR
IServiceRequest psr =
(IServiceRequest)m_session.getObject(IServiceRequest.OBJECT_TYPE,
"NCR01562");
System.out.println("PSR : " + psr.getName());
//Load a QCR
IQualityChangeRequest qcr =
       (IQualityChangeRequest)m_session.getObject(IQualityChangeRequest.
       OBJECT_TYPE, "CAPA02021");
System.out.println("QCR : " + qcr.getName());
//Load a query
IQuery query =
       (IQuery)m_session.getObject(IQuery.OBJECT_TYPE,
    "/Personal Searches/Part Numbers Starting with P");
System.out.println("Query : " + query.getName());
  //Load an RFQ
IRequestForQuote rfq =
    IRequestForQuote)m_session.getObject(IRequestForQuote.OBJECT_TYPE,
    "RFQ01048");
System.out.println("RFQ : " + rfq.getName());
//Load an RFQ response
params.clear();
 params.put(SupplierResponseConstants.ATT_COVERPAGE_RFQ_NUMBER,
"RFQ01048");
 params.put(SupplierResponseConstants.ATT_COVERPAGE_SUPPLIER,
"SUP20013");
 ISupplierResponse rfqResp =
    (ISupplierResponse)m_session.getObject(ISupplierResponse.OBJECT_TYPE
    , params);
System.out.println("RFQ Response : " + rfqResp.getName());
//Load Sourcing Projects
IProject prj =
    (IProject)m_session.getObject(IProject.OBJECT_TYPE, "PRJACME_110");
System.out.println("Project : " + prj.getName());
//Load a specification
ISpecification spec =
    (ISpecification)m_session.getObject(ISpecification.OBJECT_TYPE,
    "WEEE");
System.out.println("Specification : " + spec.getName());
//Load a substance
ISubstance sub =
    (ISubstance)m_session.getObject(ISubstance.OBJECT_TYPE, "Cadmium");
System.out.println("Substance : " + sub.getName());
//Load a supplier
ISupplier supplier =
    (ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE, "SUP20013");
```

```
System.out.println("Supplier : " + supplier.getName());
//Load a transfer order
ITransferOrder to =
    (ITransferOrder)m_session.getObject(TransferOrderConstants.CLASS_CTO
    , "456602");
System.out.println("TransferOrder : " + to.getName());
//Load a user
params.clear();
params.put(UserConstants.ATT_GENERAL_INFO_USER_ID, "OWELLES");
IUser user =
    (IUser)m_session.getObject(IUser.OBJECT_TYPE, params);
System.out.println("User : " + user.getName());
//Load a user group
params.clear();
params.put(UserGroupConstants.ATT_GENERAL_INFO_NAME, "Designers");
IUserGroup group =
    (IUserGroup)m_session.getObject(IUserGroup.OBJECT_TYPE, params);
System.out.println("UserGroup : " + group.getName());
} catch (APIException ex) {
  System.out.println(ex);
}
```

Creating Objects

To create an Agile PLM object, use one of the `IAgileSession.createObject()` methods:

- `IAgileObject createObject(Object objectType, Object params)`
- `IAgileObject createObject(int objectType, Object params)`

Note The SDK does not support setting the Life Cycle Phase (LCP)/Workflow status attribute of an object while you are creating that object. The reason is that the necessary settings for LCP are not available until after the object is created. The same is also applicable in the UI. For example, I`Change` will not get any LCP values until a Workflow is selected. However, you can use the SDK to create objects, and then set and modify the LCP/Workflow status attribute. Also, you cannot get a list of values for this field, until the object is created, and the relevant actions are performed on the object.

The `objectType` and `params` parameters are identical to those used in the `IAgileSession.getObject()` methods; for more information, see Loading Objects on page 14. Except for `IFolder` and `IQuery` objects, you must specify a concrete class for the `objectType` parameter. For example, if you are creating a part, you can't specify `ItemConstants.CLASS_PARTS_CLASS` because that class is an abstract class that can't be instantiated. However, you can specify the class ID of any predefined or user-defined concrete class, such as `ItemConstants.CLASS_PART`.

If you are creating an object of a user-defined subclass, the `objectType` parameter of `createObject()` should be an `Integer` object corresponding to the subclass ID. You may wish to define constants for all user-defined subclasses available on your Agile PLM system.

In addition to a `Map` or `String` type, the `params` parameter for `IAgileSession.createObject()` can also be an `INode` object representing an autonumber source for the particular object class. The Agile Application Server queries the autonumber source for the next number in its sequence, and that number is used as the unique identifier.

Note You cannot specify an `INode` object for the `params` parameter for objects that don't have their autonumber sources available.

The following example shows how to create part 1000-02 using a `Map` parameter that specifies an attribute (`ItemConstants.ATT_TITLE_BLOCK_NUMBER`) and a value.

Example: Creating a part using a Map

```
try {
  Map params = new HashMap();
   params.put(ItemConstants.ATT_TITLE BLOCK NUMBER, "1000-02");
   IItem item = (IItem)m_session.createObject(ItemConstants.CLASS_PART,
   params);
} catch (APIException ex) {
    System.out.println(ex);
}
```

The following example shows how to create part 1000-02 by specifying a unique `String` identifier.

Example: Creating a part using a String

```
try {
   IItem item = (IItem)m_session.createObject(ItemConstants.CLASS_PART,
   "1000-02");
} catch (APIException ex) {
    System.out.println(ex);
}
```

Working with Agile PLM Classes

Because classes are customized for each Agile Application Server, you should avoid hard-coding references to class names, particularly if your program is going to be used on multiple Agile Application Servers or in different locales. Instead, you can retrieve the classes for each object type at run time. Your program can then provide a user interface to allow the user to select a class from the list.

The following example shows how to retrieve the list of classes for a particular object type at run time.

Example: Getting classes

```
try {
//Get the IAdmin interface for this session
  IAdmin m_admin = m_session.getAdminInstance();
//Get the Item base class
  IAgileClass itemClass =
    m_admin.getAgileClass(ItemConstants.CLASS_ITEM_BASE_CLASS);
// Clear the Item Type combo box
  comboItemType.removeAllItems();
```

```
// Get the Item subclass names and populate the Item Type combo box
  IAgileClass[] subclasses = itemClass.getSubclasses();
  for (int i = 0; i < subclasses.length; ++i) {
    comboItemType.addItem(subclasses[i].getName());
  }
} catch (APIException ex) {
    System.out.println(ex);
}
```

Creating Objects of User-Defined Subclasses

User-defined subclasses are classes created specifically for your Agile PLM system. Consequently, the Agile API doesn't provide predefined class ID constants for them. To specify a user-defined subclass for the `objectType` parameter of `createObject()`, pass an Integer corresponding to the class ID. To get the class ID for a user-defined class, use the `IAgileClass.getId()` method.

The following example shows how to create a Resistor object. In this example, Resistor is a user-defined subclass of the Parts class.

Example: Creating an object of a user-defined subclass

```
try {
//Define a variable for the Resistor subclass
  Integer classResistor = null;
//Get the Resistor subclass ID
 IAgileClass[] classes =
       m_admin.getAgileClasses(IAdmin.CONCRETE);
  for (int i = 0; i < classes.length; i++) {
    if (classes[i].getName().equals("Resistor")) {
      classResistor = (Integer)classes[i].getId();
      break;
    }
  }
//Create a Resistor object
  if (classResistor != null) {
 IItem resistor =
       (IItem)m_session.createObject(classResistor, "R10245");
  }
} catch (APIException ex) {
    System.out.println(ex);
}
```

Of course, you can also reference a user-defined subclass by name, as in the following example. However, class names are not necessarily unique. If there are two subclasses with the same name, the Agile API matches the first one found, which may not be the one you intended.

```
Creating an object by referencing the subclass name try {
  IItem resistor = (IItem)m_session.createObject("Resistor", "R10245");
} catch (APIException ex) {
```

```
   System.out.println(ex);
}
```

Using AutoNumbers

An Agile PLM class can have one or more `AutoNumber` sources. An `AutoNumber` source is a predefined sequence of numbers that automatically number an object. `AutoNumber` sources are defined in the administrative functionality of Agile Java Client.

Note The Manufacturers and Manufacturer Parts classes, and their user-defined subclasses, do not support automatic numbering.

You must configure your Agile Application Server to use `AutoNumber` when you create an object of a particular class. The `IAgileClass.isAutoNumberRequired()` method determines if automatic numbering is required for the object. However, this method is deprecated because the Agile API does not enforce automatic numbering of objects, even when it is required for a particular class. If your environment requires this capability, you must develop the necessary routine. Thus, if you develop a GUI program that allows users to create Agile PLM objects, make sure the user interface enforces automatic numbering when it is required. For an example of how a client program enforces automatic numbering, create a few objects using Agile Web Client and note how the user interface works.

To get the next available AutoNumber in the sequence:

Use the `IAutoNumber.getNextNumber(IAgileClass)` method to assign the next available `AutoNumber` in the sequence. This method will check to ensure the number is not used by another object. It will continue this process until it finds and returns the first available AutoNumber for the specified Agile subclass. This method will throw an exception if it fails to get the next available AutoNumber. The `IAutoNumber.getNextNumber()` method will not check and skip if the number is already used by another object.

The following example shows how to create a part using the next `AutoNumber`.

Example: Getting the next available AutoNumber

```
private void createPart(String partNumber) throw APIException {
   IAdmin admin;
   IAgileClass cls;
   IItem part;
   IAutoNumber[] numSources;
   String nextAvailableAutoNumber;

//Get the Admin instance
   admin = session.getAdminInstance();
//Get the Part class
   cls = admin.getAgileClass(ItemConstants.CLASS_PART);
//Check if AutoNumber is required
   if (isAutoNumberRequired(cls)) {
// Get AutoNumber sources for the Part class
     numSources = cls.getAutoNumberSources();
// Get the next available AutoNumber using the first autonumber source
```

```
    nextAvailableAutoNumber = numSources[0].getNextNumber(cls);
  // Create the part using the available AutoNumber
  part =
   (IItem)session.createObject(ItemConstants.CLASS_PART,
   nextAvailableAutoNumber);
  } else {
  // Create the part using the specified number
  // (if AutoNumber is not required)
   part = (IItem)session.createObject(ItemConstants.CLASS_PART,
   partNumber);
}
public boolean isAutoNumberRequired(IAgileClass cls) throws
APIException {
  if (cls.isAbstract()) {
    return false;
  }
  IProperty p =
((INode)cls).getProperty(PropertyConstants.PROP_AUTONUMBER_REQUIRED);
  if (p != null) {
    IAgileList value = (IAgileList)p.getValue();
    return ((Integer)(value.getSelection()[0]).getId()).intValue() ==
1;
  }
  return false;
}
```

To get the next AutoNumber in the sequence:

Use the `IAutoNumber.getNextNumber(IAgileClass)` method to increment or find the next AutoNumber in the sequence. This method generates the next AutoNumber, but does not check its availability. That is, if it is not used by another Agile object. The method will throw an exception if it fails to get the next AutoNumber.

For example, if you want to assign the next AutoNumber without checking its availability, modify Example 10 as follows:

- Replace `String nextAvailableAutoNumber` with `String nextAutoNumber`.
- Replace `nextAvailableAutoNumber = numSources[0].getNextNumber(cls);` with `nextAutoNumber = numSources[0].getNextNumber();`
- Replace `part = (IItem)session.createObject(ItemConstants.CLASS_PART, nextAvailableAutoNumber);` with `part = (IItem)session.createObject(ItemConstants.CLASS_PART, nextAutoNumber);`

Setting the Required Fields

A class can be defined with several required attributes. To make a particular attribute mandatory, the Agile PLM administrator sets the Visible and Required properties for the attribute to Yes. If you try to create an object in Agile Java Client or Agile Web Client without completing the required fields, the Client does not allow you to save the object until you set the values for all required fields.

Although the Agile PLM administrator can define whether an attribute is required for a class, the Agile API doesn't automatically enforce required fields when you set values. Consequently, you can use the API to create and save an object even if values aren't set for all required fields. If you want to enforce required fields in your Client program and make them behave the way they do in Agile Web and Java Clients, you have to write that code.

To check for required fields:

1. Call `ITable.getAttributes()` or `ITableDesc.getAttributes()` to get the list of attributes for a table.
2. For each attribute, call `IAttribute.getProperty(PropertyConstants.PROP_REQUIRED).getValue()` to get the value for the Required property.

The following example shows how to get the array of required attributes for Page One, Page Two, and Page Three for a class.

Example: Getting required attributes for a class

```
/**
 * Returns true if the specified attribute is required and visible.
 */
public boolean isRequired(IAttribute attr) throws APIException {
  boolean result = false;
  IProperty required =
attr.getProperty(PropertyConstants.PROP_REQUIRED);
  if (required != null) {
    Object value = required.getValue();
    if (value != null) {
      result = value.toString().equals("Yes");
    }
  }
  IProperty visible = attr.getProperty(PropertyConstants.PROP_VISIBLE);
  if (visible != null) {
    Object value = visible.getValue();
    if (value != null) {
      result &= value.toString().equals("Yes");
    }
  }
  return result;
}
/**
 * Returns an array containing the required attributes for the
specified class.
 */
public IAttribute[] getRequiredAttributes(IAgileClass cls) throws
APIException {
  //Create an array list for the results
  ArrayList result = new ArrayList();
```

```
  //Check if the class is abstract or concrete
  if (!cls.isAbstract()) {
    IAttribute[] attrs = null;
    //Get required attributes for Page One
    ITableDesc page1 =
cls.getTableDescriptor(TableTypeConstants.TYPE_PAGE_ONE);
    if (page1 != null) {
      attrs = page1.getAttributes();
      for (int i = 0; i < attrs.length; i++) {
        IAttribute attr = attrs[i];
        if (isRequired(attr)) {
          result.add(attr);
        }
      }
    }
    //Get required attributes for Page Two
    ITableDesc page2 =
cls.getTableDescriptor(TableTypeConstants.TYPE_PAGE_TWO);
    if (page2 != null) {
      attrs = page1.getAttributes();
      for (int i = 0; i < attrs.length; i++) {
        IAttribute attr = attrs[i];
        if (isRequired(attr)) {
          result.add(attr);
        }
      }
    }
    //Get required attributes for Page Three
    ITableDesc page3 =
cls.getTableDescriptor(TableTypeConstants.TYPE_PAGE_THREE);
    if (page3 != null) {
      attrs = page3.getAttributes();
      for (int i = 0; i < attrs.length; i++) {
        IAttribute attr = attrs[i];
        if (isRequired(attr)) {
          result.add(attr);
        }
      }
    }
  }
  return (IAttribute[])result.toArray(new IAttribute[0]);
}
```

Note Primary key fields that are used to create an object are required regardless of the setting for the Required property. For example, for items the `[Title Block.Number]` field must be specified to create a new item regardless whether the field is required.

Creating Different Types of Objects

The following example shows several different ways to create various types of Agile PLM objects. To simplify the code, `AutoNumbers` are not used.

Example: Creating different types of objects

```
try {
//Create a Map object to store parameters
    Map params = new HashMap();
//Create a change
IChange eco =
    (IChange)m_session.createObject(ChangeConstants.CLASS_ECO,
    "C00002");
    System.out.println("Change : " + eco.getName());
//Create a commodity
ICommodity comm =
    (ICommodity)m_session.createObject(CommodityConstants.CLASS_COMMODIT
    Y,"RES");
 System.out.println("Commodity : " + comm.getName());
//Create a customer
params.clear();
params.put(CustomerConstants.ATT_GENERAL_INFO_CUSTOMER_NUMBER,
"CUST00006");
params.put(CustomerConstants.ATT_GENERAL_INFO_CUSTOMER_NAME, "Western
Widgets");
ICustomer customer =
    (ICustomer)m_session.createObject(CustomerConstants.CLASS_CUSTOMER,
    params);
  System.out.println("Customer : " + customer.getName());
//Create a declaration
params.clear();
ISupplier supplier =
    (ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE, "SUP20013");
params.put(DeclarationConstants.ATT_COVER_PAGE_NAME, "MD00001");
params.put(DeclarationConstants.ATT_COVER_PAGE_SUPPLIER, supplier);
IDeclaration dec = (IDeclaration)
    m_session.createObject(DeclarationConstants.CLASS_SUBSTANCE_DECLARAT
    ION, params);
  System.out.println("Declaration : " + dec.getName());
//Create a discussion
params.clear();
params.put(DiscussionConstants.ATT_COVER_PAGE_NUMBER, "D000201");
params.put(DiscussionConstants.ATT_COVER_PAGE_SUBJECT, "Packaging
issues");
IDiscussion discussion =
    (IDiscussion)m_session.createObject(DiscussionConstants.CLASS_DISCUS
    SION, params);
    System.out.println("Discussion : " + discussion.getName());
//Create a file folder
IFileFolder ff =
```

```
    (IFileFolder)m_session.createObject(FileFolderConstants.CLASS_FILE_F
    OLDER, "FOLDER00133");
 System.out.println("File Folder : " + ff.getName());
//Create a folder
params.clear();
IFolder parentFolder =
    (IFolder)m_session.getObject(IFolder.OBJECT_TYPE, "/Personal
    Searches");
params.put(FolderConstants.ATT_FOLDER_NAME, "MyTemporaryQueries");
params.put(FolderConstants.ATT_PARENT_FOLDER, parentFolder);
IFolder folder = (IFolder)m_session.createObject(IFolder.OBJECT_TYPE,
params);
System.out.println("Folder : " + folder.getName());
//Create an item
IItem part =
    (IItem)m_session.createObject(ItemConstants.CLASS_PART, "1000-02");
System.out.println("Item : " + part.getName());
//Create a manufacturer
params.put(ManufacturerConstants.ATT_GENERAL_INFO_NAME, "World
Enterprises");
IManufacturer mfr =
    (IManufacturer)m_session.createObject(ManufacturerConstants.CLASS_MA
    NUFACTURER, params);
  System.out.println("Manufacturer : " + mfr.getName());
//Create a manufacturer part
params.clear();
params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_NAME
, "World Enterprises");
params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_PART
_NUMBER, "WE10023-45");
IManufacturerPart mfrPart =
    (IManufacturerPart)m_session.createObject
(ManufacturerPartConstants.CLASS_MANUFACTURER_PART, params);
System.out.println("ManufacturerPart : " + mfrPart.getName());
//Create a manufacturing site
IManufacturingSite siteHK =
    (IManufacturingSite)m_session.createObject(ManufacturingSiteConstant
    s.CLASS_SITE, "Hong Kong");
  System.out.println("ManufacturingSite : " + siteHK.getName());
//Create a package
IPackage pkg =
    (IPackage)m_session.createObject(PackageConstants.CLASS_PACKAGE,
    "PKG00010");
System.out.println("Package : " + pkg.getName());
//Create a price
params.clear();
params.put(PriceConstants.ATT_GENERAL_INFORMATION_NUMBER,
"PRICE10008");
```

```
params.put(PriceConstants.ATT_GENERAL_INFORMATION_CUSTOMER,
"CUST00006");
params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_NUMBER, "1000-
02");
params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_REV, "B");
params.put(PriceConstants.ATT_GENERAL_INFORMATION_PROGRAM,
"PROGRAM0023");
params.put(PriceConstants.ATT_GENERAL_INFORMATION_MANUFACTURING_SITE,
"San Jose");
params.put(PriceConstants.ATT_GENERAL_INFORMATION_SUPPLIER,
"SUP20013");
IPrice price =
    (IPrice)m_session.createObject(PriceConstants.CLASS_PUBLISHED_PRICE,
    params);
 System.out.println("Price : " + price.getName());
//Create a program
DateFormat df =
    new SimpleDateFormat("MM/dd/yy");
IAttribute attr =
    m_admin.getAgileClass(ProgramConstants.CLASS_PROGRAM).getAttribute(P
    rogramConstants.ATT_GENERAL_INFO_DURATION_TYPE);
IAgileList list = attr.getAvailableValues();
list.setSelection(new Object[] {"Fixed"});
params.clear();
params.put(ProgramConstants.ATT_GENERAL_INFO_NAME, "Wingspan Program");
params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE,
df.parse("06/01/05"));
params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
df.parse("06/30/05"));
params.put(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE, list);
IProgram program =
(IProgram)m_session.createObject(ProgramConstants.CLASS_PROGRAM,
params);
System.out.println("Program : " + program.getName());
//Create a PSR
IServiceRequest psr =
    (IServiceRequest)m_session.createObject(ServiceRequestConstants.CLAS
    S_NCR, "NCR01562");
  System.out.println("PSR : " + psr.getName());
//Create a QCR
IQualityChangeRequest qcr =
(IQualityChangeRequest)m_session.createObject(
    QualityChangeRequestConstants.CLASS_CAPA, "CAPA02021");
    System.out.println("QCR : " + qcr.getName());
//Create a query
params.clear();
IFolder parent =
    (IFolder)m_session.getObject(IFolder.OBJECT_TYPE, "/Personal
    Searches");
String condition =
```

```
    "[Title Block.Number] starts with 'P'";
params.put(QueryConstants.ATT_CRITERIA_CLASS,
ItemConstants.CLASS_ITEM_BASE_CLASS);
params.put(QueryConstants.ATT_CRITERIA_STRING, condition);
params.put(QueryConstants.ATT_PARENT_FOLDER, parent);
params.put(QueryConstants.ATT_QUERY_NAME, "Part Numbers Starting with
P");
IQuery query =
    (IQuery)m_session.createObject(IQuery.OBJECT_TYPE, params);
System.out.println("Query : " + query.getName());
//Create a specification
ISpecification spec = (ISpecification)
    m_session.createObject(SpecificationConstants.CLASS_SPECIFICATION,
    "WEEE");
  System.out.println("Specification : " + spec.getName());
//Create a substance
ISubstance sub =
    (ISubstance)m_session.createObject(SubstanceConstants.CLASS_SUBSTANC
    E, "Cadmium");
  System.out.println("Substance : " + spec.getName());
//Create a transfer order
ITransferOrder to =
    (ITransferOrder)m_session.createObject(TransferOrderConstants.CLASS_
    CTO, "456602");
  System.out.println("TransferOrder : " + to.getName());
//Create a user
params.clear();
params.put(UserConstants.ATT_GENERAL_INFO_USER_ID, "OWELLES");
params.put(UserConstants.ATT_LOGIN_PASSWORD, "agile");
IUser user =
    (IUser)m_session.createObject(UserConstants.CLASS_USER, params);
 System.out.println("User : " + user.getName());
//Create a user group
params.clear();
params.put(UserGroupConstants.ATT_GENERAL_INFO_NAME, "Designers");
IUserGroup group =
    (IUserGroup)m_session.createObject(UserGroupConstants.CLASS_USER_GRO
    UP, params);
System.out.println("UserGroup : " + group.getName());
} catch (APIException ex) {
    System.out.println(ex);
}
```

Note You cannot use the Agile API to create a `SupplierResponse`.

Checking the State of Agile PLM Objects

The `IStateful` interface supports Agile objects that have either Agile Workflow status or Agile life cycle states. Objects that support this interface are Item and routable objects.

Routable objects are:

- `IChange`
- `IDeclaration`
- `IFileFolder`
- `IPackage`
- `IProgram`
- `IQualityChangeRequest`
- `IServiceRequest`
- `ITransferOrder`

The following example returns an array that shows all states of the object, or null when they are not defined.

Example: Getting the array that defines the different states of an object

```
public interface IStateful {
public IStatus[] getStates()
throws APIException;
}
```

The following example returns the current state of the object, or null if it is not defined.

Example: Getting the current state of the object

```
public interface IStateful {
public IStatus getStatus()
throws APIException;
}
```

Propagating Values to Related Objects

Several objects in Agile PLM have related objects. For example, problem reports and nonconformance reports have a Related PSR table. On the Related PSR table, you can specify that a Workflow event should trigger a particular result in a related object, such as another problem report or noncomformance report. The triggered result does not occur instantaneously. In fact, there may be a noticeable delay—perhaps several seconds—in the time it takes Agile PLM to propagate values to related objects.

Saving an Object to a New Object

The Agile API lets you save an existing object as a new object. For example, in addition to a Save button, a dialog box in your program may have a Save As button, which saves the data to a new object. When you use the `IDataObject.saveAs()` method, you must specify the subclass that you are using to save the object and the object number. If the subclass supports it, you can use an AutoNumber.

This example shows how to save the current object to a new object using the next AutoNumber for the specified subclass.

Example: Saving an object as a new object

```
private void saveAsObject(IDataObject obj, IAgileClass sub) {
  String nextNum;
  try {
    // Get the next autonumber for the sublass
    IAutoNumber[] numSources = sub.getAutoNumberSources();
    nextNum = numSources[0].getNextNumber();
    // Save the object
    obj.saveAs(sub, nextNum);
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Sharing an Object

The `IShareable` interface is implemented by every Agile PLM business object that the Agile API exposes. Therefore, every business object can be shared. Sharing lets you grant one or more of your roles to another Agile PLM user or user group for a specific object. The roles you can assign when you share an object include your assigned or permanent roles and any roles assigned to you from membership in a user group.

Users that have been shared an object can perform actions permitted by the roles for that object only. They don't acquire the roles in a global fashion.

The `IShareable` interface has only two methods, `getUsersAndRoles()` and `setUsersAndRoles()`. The `getUsersAndRoles()` method returns a `Map` object. Each user in the `Map` has an associated array of roles. The `setUsersAndRoles()` method has one parameter, a `Map` object, which, like the `Map` returned by `getUsersAndRoles()`, maps each user to an array of roles. Each user can be assigned a different selection of roles.

Example: Sharing an object

```
private void getDataForSharing() throws Exception {
  //Get item
  IItem item =
(IItem)m_session.getObject(ItemConstants.CLASS_ITEM_BASE_CLASS,
"P10011");
  //Get users
  IUser user1 = (IUser)m_session.getObject(UserConstants.CLASS_USER,
"albertl");
```

```
  IUser user2 = (IUser)m_session.getObject(UserConstants.CLASS_USER,
"peterl");
  IUser[] users = new IUser[]{user1, user2};
  //Get roles
  INode nodeRoles =
(INode)m_session.getAdminInstance().getNode(NodeConstants.NODE_ROLES);
  IRole role1 = (IRole)nodeRoles.getChildNode("Component Engineer");
  IRole role2 = (IRole)nodeRoles.getChildNode("Incorporator");
  IRole[] roles = new IRole[]{role1, role2};
  //Share the item
  shareItem(item, users, roles);
}
private void shareItem(IItem item, IUser[] users, IRole[] roles) throws
Exception {
  Map map = new HashMap();
  for (int i = 0; i < users.length; i++) {
    map.put(users[i], roles);
  }
  IShareable shareObj = (IShareable)item;
  shareObj.setUsersAndRoles(map);
}
```

Note Each user and user group has a Share table that lists objects that have been shared and which roles have been granted for those objects.

Deleting and Undeleting Objects

The Agile API, like Agile Web Client, lets you delete and undelete objects. To delete and undelete an object, you must have Delete and Undelete privileges, respectively, for the particular object type.

The Agile API supports “soft” and “hard” deletes. The first time you delete an object, it is “soft-deleted.” Though it is marked “Deleted” in the database, it is not permanently removed. You can still retrieve a soft-deleted object; for example, you could use the `IAgileSession.getObject()` method to get a deleted object. When you run a query, soft-deleted objects are not included in the query results. However, Agile provides predefined queries (such as the Deleted Items query in the Change Analyst Searches folder) that let you find deleted objects.

To remove an object permanently, you delete it a second time, which is a “hard” delete. Once you hard-delete an object, you cannot restore it using the `IDataObject.undelete()` method.

Not all Agile PLM objects can be deleted. For example, the following objects cannot be deleted. If you attempt to delete one of these objects, the `delete()` method throws an exception.

- An item with a pending change
- An item with a revision history

- An item with a canceled change
- An item with an AML
- A released change
- A manufacturer part currently used on the Manufacturers tab of another object
- A manufacturer with one or more manufacturer parts

If you try to delete an Item that is used on the BOM tab of another item, the Agile PLM server throws an exception whose ID is `ExceptionConstants.APDM_DELETECOMPINUSE_WARNING`. The following example shows how to disable this warning and delete the item.

Example: Deleting an Item

```
private void deleteItem(IDataObject obj) {
  try {
    // Delete the Item
    obj.delete();
  } catch (APIException ex) {
    // Check for "Item is Used" warning
    if (ex.getErrorCode() ==
ExceptionConstants.APDM_DELETECOMPINUSE_WARNING) {
      int i = JOptionPane.showConfirmDialog(null, "This Item is used by
another Item. " +
      "Would you still like to delete it?", "Item is Used Warning",
JOptionPane.YES_NO_OPTION);
    }
    if (i == 0) {
      try {
        // Disable "Item is Used" warning

m_session.disableWarning(ExceptionConstants.APDM_DELETECOMPINUSE_WARNIN
G);
        // Delete the object
        obj.delete();
        // Enable "Item is Used" warning

m_session.enableWarning(ExceptionConstants.APDM_DELETECOMPINUSE_WARNING
);
      } catch (APIException exc) {
        System.out.println(exc);
      }
    } else {
      System.out.println(ex);
    }
  }
}
```

To restore an object that has been soft-deleted, use the `IDataObject.undelete()` method. Once again, to undelete an object, the user must have Undelete privileges for that object type. However, soft-deleted changes that have items on the Affected Items tab cannot be restored, regardless of the user's privileges. The following example shows how to undelete an object that has been deleted.

Example: Undeleting an object

```
private void undeleteObject(Object obj) throws APIException {
  // Make sure the object is deleted before undeleting it
  if (obj.isDeleted()) {
      // Restore the object
      obj.undelete();
  }
}
```

Closing a Session

Each Agile PLM user can open up to three concurrent sessions. Therefore, each session that you open using the Agile API should be closed properly. If you fail to close a session properly, you may not be able to log in with a new session until one of the concurrent sessions time out.

Example: Closing a session

```
public void disconnect(IAgileSession m_session) {
  m_session.close();
}
```

Chapter 3

Creating and Loading Queries

This chapter includes the following:

- About Queries 35
- Creating a Query 35
- Saving a Query to a Folder 36
- Generating Ordered (sorted) or Unordered Query Results 37
- Creating a Parameterized Query 38
- Specifying Query Attributes when Creating a Query 39
- Specifying Workflow Queries 40
- Specifying Search Criteria 41
- Using SQL Syntax for Search Criteria 52
- Setting Result Attributes for a Query 54
- Working with Query Results 62
- Creating a Where-Used Query 63
- Loading a Query 64
- Deleting a Query 65
- Simple Query Examples 65

About Queries

An `IQuery` is an object that defines how to search for Agile PLM data. It defines a search similar to the searches that you can use in Agile Web Client. The search can have multiple search criteria (like an Advanced Search in Agile Web Client), or it can be a simple search that specifies only one criterion.

Creating a Query

To create and execute a query, you must first create an `IQuery` object. As with other Agile API objects, you create the object using the `IAgileSession.createObject()` method.

In its simplest form, the parameters that you pass with the `createObject()` method to create a query are the `IQuery` object type and the query class used in the search. In the following example, the query class is the Item class.

Example: Creating a query

```
try {
  IQuery query =
    (IQuery)session.createObject(IQuery.OBJECT_TYPE,
  ItemConstants.CLASS_ITEM_BASE_CLASS);
  query.setCaseSensitive(false);
  query.setCriteria("[Title Block.Number] starts with 'P'");
```

```
   ITable results = query.execute();
    } catch (APIException ex) {
     System.out.println(ex);
}
```

The query class you specify with the `createObject()` method also includes objects from all of its subclasses. For example, if you search for objects in the Item class, the results include parts and documents. If you search for objects in the Change class, the results include objects from all Change subclasses (Deviation, ECO, ECR, MCO, PCO, SCO, and Stop Ship). If you want to search only a specific subclass, you should explicitly specify that class.

The following example shows how to create a query that searches for objects in a subclass named Foobar:

Example: Specifying the query class

```
IAdmin admin = m_session.getAdminInstance();
IAgileClass cls = admin.getAgileClass("Foobar");
IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE, cls);
```

Saving a Query to a Folder

After you name a query using the `IQuery.setName()` method, you can add it to a folder. The following example shows how to name a query and add it to the Personal Searches folder. You can retrieve the query from the folder later to reuse it.

Example: Naming a query and adding it to a folder

```
try {
   IQuery query =
    (IQuery)session.createObject(IQuery.OBJECT_TYPE,
     ItemConstants.CLASS_ITEM_BASE_CLASS);
   query.setCaseSensitive(false);
   query.setCriteria("[Title Block.Number] starts with 'P'");
   query.setName("Items Whose Number Starts with P");
  IFolder folder =
    (IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
       "/Personal Searches");
    folder.addChild(query);
} catch (APIException ex) {
   System.out.println(ex);
}
```

You can also use the `IQuery.saveAs()` method to name a query and save it to a folder.

Example: Using IQuery.saveAs() to save a query to a folder

```
try {
   IQuery query = (IQuery)session.createObject(IQuery.OBJECT_TYPE,
     ItemConstants.CLASS_ITEM_BASE_CLASS);
   query.setCaseSensitive(false);
   query.setCriteria("[Title Block.Number] starts with 'P'");
```

```
  IFolder folder = (IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
"/Personal Searches");
  query.saveAs("Items Whose Number Starts with P", folder);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Note Any query that you create without explicitly saving it to a folder is considered a temporary query. The Agile Application Server will automatically delete all temporary queries when the user session is closed.

Generating Ordered (sorted) or Unordered Query Results

As shown in examples for Creating a Query on page 35, executing `IQuery.execute()` and `IQuery.execute(Object[] params)` methods returns an ordered query result in `ITable`.

To improve query performance, the SDK provides the following methods to return results that are not sorted in the default order. However, if the query criteria has the `starts with` condition, then results are always sorted on that attribute and passing `skipOrdering` as `true` in `execute(boolean)` will not skip ordering.

Note To sort query results by other than the default order, see Sorting Query Results on page 62.

- `IQuery.execute(boolean skipOrdering)`
- `IQuery.execute(Object[] params, boolean skipOrdering)`

To skip or perform ordering, set the boolean `skipOrdering` to true or false as shown in the following example.

Example: Skip ordering in query results

```
try {
  IQuery query =
    (IQuery)session.createObject(IQuery.OBJECT_TYPE,
  ItemConstants.CLASS_ITEM_BASE_CLASS);
  query.setCaseSensitive(false);
  query.setCriteria("[Title Block.Number] starts with 'P'");
  // The boolean is set to true to skip ordering
  ITable results = query.execute(true);
   } catch (APIException ex) {
    System.out.println(ex);
}
```

Creating a Parameterized Query

When you specify criteria for a query, you can use a number preceded by a percent sign (%) to indicate a parameter placeholder. The parameter value is specified later, for example at runtime. Parameters provide a convenient way to pass values to a query, and they can save time and reduce extra coding. Parameterized queries can be saved and reused later.

Note The right hand operand query parameter supports one placeholder per each query operator, so if the query criteria have three query operators, then the query can have a total of three placeholders corresponding to the three operators. The `between` and `not between` query operations are different. For example, `[2091] contains none of (%0,%1);` is not allowed, but `[2091] contains none of (%0);` is allowed, and `query.execute(new Object[]{new Object[]{"B", "C"}});` is not allowed.

Indexes for query parameters are 0-based. Parameters are numbered 0, 1, 2, and so on. Always enumerate the parameters in ascending order. The following example shows a query with three parameters whose values are specified using the `IQuery.execute(Object[])` method.

Example: Parameterized query using IQuery.execute(Object[])

```
public ITable runParameterizedQuery() throws Exception {
  String condition = "[Title Block.Number] starts with %0 and" +
                     "[Title Block.Part Category] == %1 and" +
                     "[Title Block.Description] contains %2";
  IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
     ItemConstants.CLASS_PART);
   query.setCriteria(condition);
  ITable table = query.execute(new Object[] {"1", "Electrical",
"Resistor"});
  return table;
}
```

You can also specify query parameters using `IQuery.setParams()` method, as shown in the following example. Make sure you set the query parameter values before calling `IQuery.execute()`. Otherwise, when you run the query it will use previous parameter values. If parameters have not been set, the query uses null values. Similarly, if you do not pass any parameters to a query, then the `IQuery.getParams()` method returns null.

Example: Parameterized query using IQuery.setParams()

```
public ITable runParameterizedQuery() throws Exception {
  String condition = "[Title Block.Number] starts with %0 and" +
                     "[Title Block.Part Category] == %1 and" +
                     "[Title Block.Description] contains %2";
  IQuery query = (IQuery) m_session.createObject(IQuery.OBJECT_TYPE,
ItemConstants.CLASS_PART);
  query.setCriteria(condition);
  query.setParams(new Object[] {"1", "Electrical", "Resistor"});
  ITable table = query.execute();
  return table;
```

```
}
```

Do not use quote characters around parameterized queries because they will create a set of values (more than one element) for the query when parameters can only refer to a given value. The following examples show the proper use of quote characters when creating parameterized queries:

Example: Correct use of quote characters in a parameterized search query

```
String criteria = "[NUMBER] == %0";
query.execute(new Object[]{"P1000-02"});
String criteria = "[P2.LIST01] in %0";
query.execute(new Object[]{new Object[]{"A1", "B2"}});
```

Specifying Query Attributes when Creating a Query

Instead of passing only the query class when you create a query, you can use a more advanced form of the `createObject()` method and pass a `Map` object containing one or more attribute values. The `QueryConstants` class contains several constants for query attributes that you can set when you create a query. These are virtual attributes that do not exist in the Agile PLM database, but that can be used to define the query at run time.

Attribute Constant	Description
ATT_CRITERIA_CLASS	Query class.
ATT_CRITERIA_PARAM	Search condition parameter value (for a parameterized search condition).
ATT_CRITERIA_STRING	Search condition string.
ATT_PARENT_FOLDER	Parent folder where the query resides.
ATT_QUERY_NAME	Query name.

The following example shows how to set the query class, search condition, parent folder, and query name when you create the query.

Example: Specifying query attributes when you create a query

```
try {
String condition = "[Title Block.Number] starts with 'P'";
  IFolder parent = (IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
"/Personal Searches");
  HashMap map = new HashMap();
  map.put(QueryConstants.ATT_CRITERIA_CLASS,
ItemConstants.CLASS_ITEM_BASE_CLASS);
  map.put(QueryConstants.ATT_CRITERIA_STRING, condition);
  map.put(QueryConstants.ATT_PARENT_FOLDER, parent);
  map.put(QueryConstants.ATT_QUERY_NAME, "Part Numbers Starting with
P");
  IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
map);
  ITable results = query.execute();
} catch (APIException ex) {
  System.out.println(ex);
```

```
}
```

Specifying Workflow Queries

A workflow is a sequence of statuses that a routable object (such as a change) follows. Workflows automate many processes; for example, when a change is released, Agile users on the notification list receive automatic email with this information.

Routable objects are created from classes that represent processes, such as a change in the change control process, or a transfer order, or a product service request. These objects can be *routed* to Agile users for approval or other input via workflows.

Specifying workflow queries using workflow attributes enable searching and checking the status of a routable object in the assigned workflow. These searches are supported in both PLM's Web and Java Clients. Using the Clients, you can specify a single workflow attribute for the search criteria, or group one or more attributes in the search criteria. The *grouping* option in Web and Java Clients allow you to group search conditions. For information about workflow queries and grouping options, refer to workflow-related sections of *Getting Started with Agile PLM.*

The workflow attributes supported in the SDK query criteria are:

- Approver
- Approver Action
- Observer
- Observer Action
- Signoff User
- Notify User

The workflow attributes that require grouping in the criteria and are not supported in SDK are:

- Duration
- Next Status
- Workflow Status
- Status changed by

The following examples show how to specify a Workflow query.

Example: Using the workflow Approver attribute in query criteria

```
private void testWorkflowQuery(IAgileSession session) throws Exception
{
       IQuery query = (IQuery)session.createObject(IQuery.OBJECT_TYPE,
       ChangeConstants.CLASS_ECO);
       String criteria = "[Workflow.Approver] contains ([General
       Info.User ID] == 'admin')";
       query.setCriteria(criteria);
       ITable result = query.execute();
       System.out.println(result.size());
}
```

Note For nested query examples and information, see Using the Nested Criteria to Search for Values in Object Lists on page 46.

Example: Using the workflow Approver Action attribute in query criteria

```
private void testWorkflowQuery(IAgileSession session) throws Exception
{
        IQuery query = (IQuery)session.createObject(IQuery.OBJECT_TYPE,
        ChangeConstants.CLASS_ECO);
        String criteria = "[Workflow.Approver Action] contains
        'Approved'";
        query.setCriteria(criteria);
        ITable result = query.execute();
        System.out.println(result.size());
}
```

Specifying Search Criteria

You can narrow the number of objects returned from a search by specifying search criteria. If you don't specify search criteria, the query returns references to all objects in the specified query class. It's a good idea to limit the search criteria as much as possible, as the amount of data returned may be excessively large, resulting in decreased performance.

There are three different `setCriteria()` methods you can use to specify query criteria:

- `setCriteria(ICriteria criteria)` – Sets the query criteria from data stored in the Criteria administrative node. The Criteria administrative node defines reusable criteria for the Workflow, but the nodes can also be used as ordinary search criteria.

Note Workflow query is not supported in the current Release of the SDK.

- `setCriteria(java.lang.String criteria)` – Sets the search criteria from a specified `String`.
- `setCriteria(java.lang.String criteria, java.lang.Object[] params)` – Sets the search criteria from a specified `String` that references one or more parameters.

Unless you use the first `setCriteria()` method, which takes an `ICriteria` object for its parameter, the Agile API parses the search criteria as a `String`.

Search Conditions

The Agile API provides a simple yet powerful query language for specifying search criteria. The query language defines the proper syntax for filters, conditions, attribute references, relational operators, logical operators, and other elements.

Search criteria consist of one or more search conditions. Each search condition contains the following elements:

1. Left operand – The left operand is always an attribute enclosed in brackets, such as `[Title Block.Number]`. You can specify the attribute as an attribute name (fully qualified name or short name) or attribute ID number. The attribute specifies which characteristic of the object to use in the search.

2. Relational operator – The relational operator defines the relationship that the attribute has to the specified value, for example, “equal to” or “not equal to.”
3. Right operand – The matching value for the specified attribute in the left operand. The right operand can be a constant expression or a set of constant expressions. A set of constant expressions is needed if the relational operator is “between,” “not between,” “in,” or “not in.”

Following is an example of a search condition:

```
[Title Block.Description] == 'Computer'
```

This is another example where the right operand is a set of constant expressions:

```
[Page Two.Numeric01] between ('1000', '2000')
```

Query Language Keywords

When you specify a search condition, you must use proper keywords to construct the statement. The following keywords are available:

and	does	less	or	to
asc	equal	like	order	union
between	from	minus	phrase	where
by	greater	none	select	with
contain	in	not	start	word
contains	intersect	null	starts	words
desc	is	of	than	

Query language keywords are not localized. You must use English keywords, regardless of locale. You can use the keywords in lower case or upper case. In addition to keywords, you can use Agile PLM variables such as `$USER` (for current user) and `$TODAY` (for today’s date) in Agile API queries.

Note The "in" operator does not support MultiList in (set) query criteria.

Specifying Search Attributes

Every Agile PLM object that you can search for also has an associated set of attributes, which are inherent characteristics of the object. You can use these attributes as the left operand of a search condition. The right operand of the search condition specifies the attribute’s value(s).

A search attribute must be enclosed within brackets, for example, `[Title Block.Number]`. The brackets are needed because many attribute names have spaces. If a search attribute is not enclosed within brackets, your query will fail.

You can specify a search attribute in the following ways:

Attribute reference	Example
attribute ID number	`[1001]`
fully-qualified attribute name	`[Title Block.Number]`
short attribute name	`[Number]`

Note Because attribute names can be modified, Agile recommends referencing attributes by ID number or constant. However, many of the examples in this chapter reference attributes by name simply to make them more readable. If you choose to reference attributes by name, use the fully-qualified attribute name instead of the short name. Short attribute names are not guaranteed to be unique and could therefore cause your query to fail or produce unexpected results.

Attribute names, whether you use the long or short form, are case-insensitive. For example, `[Title Block.Number]` and `[TITLE BLOCK.NUMBER]` are both allowed. Attribute names are also localized. The names of Agile PLM attributes vary based on the locale of your Agile Application Server. If you are creating a query that is going to be used on servers in different locales, you should reference attributes by ID number (or the equivalent constant) instead of by name.

Note The APIName field, described in Accessing PLM Metadata with APIName Field on page 123, does not support specifying search attributes.

If the attribute name contains special characters, such as quotes or backslashes, you can type these characters using the backslash (\) as an escape character. For example, to include a quote character in your string, type \'. If you want to write a backslash, type two of them together (\\). If the attribute name contains square brackets, enclose the entire name in quotes:

```
['Page Two.Unit of Measure [g or oz]']
```

There are other less intuitive ways to specify attributes. For example, you can pass in an `IAttribute` reference using a parameter of the `setCriteria()` method. In the following example, `'%0'` references the attribute in the `Object` array parameter.

```
query.setCriteria("[%0] == 'Computer'", new Object[] { attr });
```

You can also use `String` concatenation to reference an attribute constant:

```
query.setCriteria("[" + ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION + "]
== 'Computer'");
```

Retrieving Searchable Attributes

The searchable attributes for a query depend on the specified query class or subclass. However, the searchable attributes for a subclass can differ greatly from searchable attributes for its parent class.

Due to database considerations, not all attributes are searchable. Generally, a few select Page One attribute (namely: Title Page, Cover Page, and General Info attributes) are searchable for each class.

If a tab is not configured in Java Client to be visible, you can still search for an attribute on that tab in the Agile SDK. However, you must search for the Table name that corresponds to the Tab name.

Note Because you use the table name to setup `IQuery`, it does not matter if an Agile administrator changes a Tab name from the name specified in Agile Java Client. Tab name changes do not affect SDK table names.

To find the searchable attributes for a query, use the `IQuery.getSearchableAttributes()` method.

Note Even though an attribute may not be searchable, it can still be included as a column in the query results. For more information, see Setting Result Attributes for a Query on page 54.

Using Relational Operators

Table below lists relational operators that are supported by the Agile API query language.

English operator	Notation	Description
`equal to`	`==`	Finds only an exact match with the specified value.
`not equal to`	`!=`	Finds any value other than an exact match with the specified value.
`greater than`	`>`	Finds any value greater than the specified value.
`greater than or equal to`	`>=`	Finds any value greater than or equal to the specified value.
`less than`	`<`	Finds any value less than the specified value.
`less than or equal to`	`<=`	Finds any value less than or equal to the specified value.
`contains, contains all`		Finds any value that includes the specified value.
`does not contain, does not contain all`		Finds any value that does not include the specified value.
`contains any`		Finds any value that includes the specified value.
`does not contain any`		Finds any value that does not include the specified value.
`contains none of`		Finds any value that includes none of the specified values.
`does not contain none of`		Behaves the same as `does not` contain any.
`starts with`		Finds values that begin with characters in the specified value.
`does not start with`		Finds values that do not begin with characters in the specified value.
`is null`		Finds objects where the selected attribute contains no value.
`is not null`		Finds objects where the selected attribute contains a value.
`like`		Performs a wildcard search, finding objects that match a single character or any string.

English operator	Notation	Description
`not like`		Performs a wildcard search, finding objects that do not match a single character or any string.
`between`		Finds objects that fall between the specified values.
`not between`		Finds objects that do not fall between the specified values.
`in`		Finds objects that match any of the specified values.
`not in`		Finds objects that do not match any of the specified values.
`contains phrase`		Finds objects with files that contain the specified phrase.
`contains all words`		Finds objects with files that contain all of the specified words.
`contains any word`		Finds objects with files that contain any of the specified words.
`contains none of`		Finds objects with files that contain none of the specified words.

Relational operators are not localized. You must use English keywords, regardless of locale. As with other query language keywords, you can use them in lower case or upper case.

Using Unicode Escape Sequences

Agile SDK Query language supports Unicode escape sequences. The primary usage of Unicode escape sequences in a query string is to search for *nonburnable* or foreign local character sets. A Unicode character is represented with the Unicode escape sequence `\uxxxx`, where `xxxx` is a sequence of four hexadecimal digits.

For example, to search for an item with Unicode 3458, use the following query:

```
Select * from [Items] where [Description] contains '\u3458'
```

There is another query operation for “contains’ usage in the case of MultiList.

Using Between, Not Between, In, and Not In Operators

The ‘between’, ‘not between’, ‘in’, and ‘not in’ relational operators are not supported directly by Agile PLM Java and Web Clients. These relational operators provide a convenient shorthand method for specifying ‘equal to’, ‘not equal to’, ‘greater than or equal to’, or ‘less than or equal to’ operations with a set of values.

Short form	Equivalent long form
`[Number] between ('1','6')`	`[Number] >= '1' and [Number] <= '6'`
`[Number] not between ('1','6')`	`[Number] < '1' and [Number] > '6'`
`[Number] in ('1','2','3','4',5','6')`	`[Number] == '1' or [Number] == '2' or [Number] == '3' or [Number] == '4' or [Number] == '5' or [Number] == '6'`
`[Number] not in ('1','2','3','4','5','6')`	`[Number]!= '1' and [Number] != '2' and [Number] != '3' and [Number] != '4' and [Number] != '5' and [Number] != '6'`

As shown in the preceding table, when you use the 'between', 'not between', 'in', and 'not in' relational operators, each value in the set of values must be enclosed in quotes and delimited by commas. Here are more criteria examples that use 'between' and 'in' relational operators:

```
[Title Block.Number] in ('1000-02', '1234-01', '4567-89')
[Title Block.Effectivity Date] between ('01/01/2001', '01/01/2002')
[Page Two.Numeric01] between ('1000', '2000')
```

Note The relational operators `any , all, none of`, and `not all` are not supported in the SDK.

Using the Nested Criteria to Search for Values in Object Lists

Several lists in Agile PLM contain business objects, such as Agile PLM users. To search for an object in such a dynamic list, you can specify nested query criteria. Nested criteria are enclosed in parentheses and separated from each other by a logical AND (&&) or OR (||) operator. A comma can also be used to separate nested criteria; it's equivalent to a logical OR.

The following criteria find a user with the first name Christopher OR the last name Nolan.

```
[Page Two.Create User] in ([General Info.First Name] == 'Christopher',
[General Info.Last Name] == 'Nolan')
```

The following criteria find a user with the first name Christopher AND the last name Nolan.

```
[Page Two.Create User] in ([General Info.First Name] == 'Christopher'
&&
[General Info.Last Name] == 'Nolan')
```

If `Part.Page Three.List01` is enabled and set to `Part Families` list, the following criteria finds a Part Family with the name `PartFamily_01`

```
[Page Three.List01] in ([General Info.Name] == 'PartFamily_01')
```

The parameter query is not supported in nested queries and multiple values for one placeholder in query parameters must be specified in two dimensional arrays as shown in the example below.

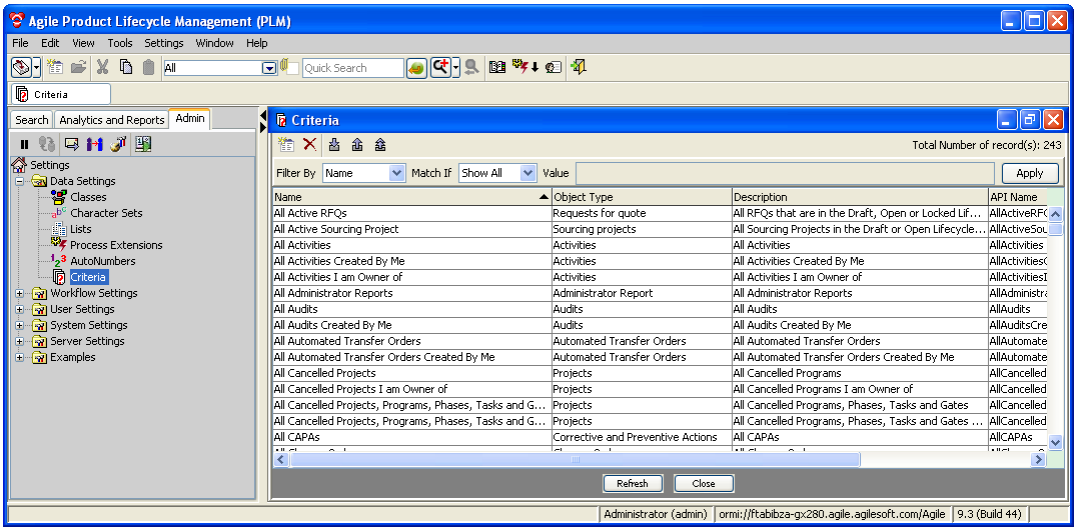
Example: Correct and incorrect parameter query in nested query criteria

- The parameter query specified in the following nested query criteria will fail to execute:
  ```
  [Page Two.User1] in ([General Info.First Name] == %0)
  ```
- However, when it is explicitly specified as a string value, instead of the placeholder, it will succeed:
  ```
  [Page Two.User1] in ([General Info.First Name] == 'Christopher')
  ```

Using Criteria Selected from Criteria Library in SDK Queries

Criteria nodes in Java or Web Client's Criteria library are `ICriteria` objects that you can use in SDK queries. To view a listing in Java Client as shown below, select Admin > Settings > Data Settings > Criteria.

Figure 3: Criteria nodes in Java Client's Criteria library



The following example gets a Criteria node from the Criteria library and loads and sets it as the SDK query criteria.

Example: Using criteria from the Criteria Library in SDK queries

```
IQuery query = (IQuery) session.createObject(IQuery.OBJECT_TYPE,
ItemConstants.CLASS_ITEM_BASE_CLASS);
    IAdmin admin = session.getAdminInstance();
// Get the Criteria Library node
    INode criteriaLibrary =
    admin.getNode(NodeConstants.NODE_CRITERIA_LIBRARY);
// Load the Criteria relevant to the query class (For example it is
Items base class)
    ICriteria criteria = (ICriteria) criteriaLibrary.getChild("All
    Released Items");
// Set the ICriteria  in SDK Query Criteria
    query.setCriteria(criteria);
```

Using Relationships and Content in SDK Queries

Agile SDK provides APIs to perform the Relationships and Content Search using the `IQuery` interface. The query criteria can contain the attributes of both the base search class and the related class.

To search using an object's Relationships:

1. Set `searchType` to `QueryConstants.RELATIONSHIPS` using `IQuery.setSearchType(int searchType)`.
2. Set the related class using `IQuery.setRelatedContentClass(Object relatedClass)`.

Example: Using an object's Relationships as the query criteria

```
IQuery query1 = (IQuery) session.createObject(IQuery.OBJECT_TYPE,
ItemConstants.CLASS_PART);
    query1.setSearchType(QueryConstants.RELATIONSHIPS);
    query1.setRelatedContentClass("Substance"); // ID or API Name
    query1.setCriteria("[Relationships.Name] Is Not Null and [Title
    Block.Number] equals 'P00001' and [Relationships.Substance.General
    Info.Name] Is Not Null");
```

To search using a Project object's Content:

1. Set `searchType` to `QueryConstants.RELATIONSHIPS` using `IQuery.setSearchType(int searchType)`.
2. Set the related class using `IQuery.setRelatedContentClass(Object relatedClass)`.

Example: Using a Project object's Content as the query criteria

```
IQuery query1 = (IQuery) session.createObject(IQuery.OBJECT_TYPE,
ProgramConstants.CLASS_ACTIVITIES_CLASS);
    query1.setSearchType(QueryConstants.RELATIONSHIPS);
    query1.setRelatedContentClass("ECO");   // ID or API Name
    query1.setCriteria("[Content.Criteria Met] Is Not Null and
    [Content.ECO.Cover Page.Originator] in ([General Info.First Name] ==
    'admin')
```

To search using a Transfer Orders object's Selected Content:

1. Set `searchType` to `QueryConstants.TRANSFER_ORDER_SELECTED_CONTENT` using `IQuery.setSearchType(int searchType)`.
2. Set the related class using `IQuery.setRelatedContentClass(Object relatedClass)`.

Example: Using a Transfer Orders object's Selected Content as the query criteria

```
IQuery query1 = (IQuery) session.createObject(IQuery.OBJECT_TYPE,
TransferOrderConstants.CLASS_CTO);
    query1.setSearchType(QueryConstants.TRANSFER_ORDER_SELECTED_CONTENT)
    ;
    query1.setRelatedContentClass("ECR");  // ID or API Name
    query1.setCriteria("[Selected Content.ECR.Cover Page.Number] equal
    to 'C0001'");
```

Searching for Words or Phrases Contained in Attachments

Two special attributes, `[Attachments.File Document Text]` and `[Files.Document Text]`, are used to index the content of files stored on the Agile file management server. If you are hosting your database on Oracle, you can take advantage of a feature that lets you search for words or phrases contained in attachments. When you create search criteria that use either of these attributes. There are four additional relational operators you can use:

- `contains phrase`
- `contains all words`
- `contains any word`
- `contains none of`

The following table shows several search conditions that search for words or phrases in attachments.

Search Condition	Finds
`[Attachments.File Document Text]` **`contains phrase`** `'adding new materials'`	Objects in which any of their attachments contain the phrase "`adding new materials.`"
`all [Attachments.File Document Text]` **`contains all words`** `'adding new materials'`	Objects in which all their attachments contain the words "adding," "new," and "materials."
`none of [Attachments.File Document Text]` **`contains any word`** `'containers BOM return output'`	Objects in which none of their attachments contain any of the words "containers," "BOM," "return," or "output."
`[Attachments.File Document Text]` **`contains none of`** `'containers BOM output'`	Objects in which any of their attachments do not contain the words "containers," "BOM," or "output."

Formatting Dates in Query Criteria

Several types of queries require date values. To pass a date as a String, use the `IAgileSession.setDateFormats()` method to specify a date format. The `setDateFormats()` method also applies to all Agile API values that you specify with `setValue()` methods.

Note If you don't set date formats explicitly using the `setDateFormats()` method, the Agile API uses the user's date format for the Agile PLM system. To see your date format in Agile Web Client, choose Settings > User Profile and then click the Preferences tab.

Example: Setting the date format for a query

```
m_session.setDateFormats(new DateFormat[] {new
SimpleDateFormat("MM/dd/yyyy")});
query.setCriteria("[Title Block.Rev Release Date] between" +
"('9/2/2001', '9/2/2003')");
query.setCriteria("[Title Block.Rev Release Date]
between (%0,%1)", new String[] {"9/2/2001", "9/2/2003"} );
```

Alternatively, if you use the `setCriteria(String criteria, Object[] params)` method, you can pass `Date` objects as parameters to the method.

Example: Passing Date objects as parameters of setCriteria()

```
DateFormat df = new SimpleDateFormat("MM/dd/yyyy");
query.setCriteria("[Title Block.Rev Release Date] between (%0,%1)",
new Object[] { df.parse("9/2/2001"), df.parse("9/2/2003") });
```

Using Logical Operators

You can use logical operators to combine multiple search conditions into a complex filter. When you have two or more conditions defined in a set of query criteria, the relationship between them is defined as either ‘and’ or ‘or’.

- and narrows the search by requiring that both conditions are met. Each item in the results must match both conditions. The ‘and’ logical operator can also be specified using two ampersands, ‘&&’.
- or broadens the search by including any object that meets either condition. Each item in the results table needs to match only one of the conditions, but may match both. The ‘or’ logical operator can also be specified using two vertical bars, ‘||’.

Logical operators are case-insensitive. For example, ‘and’ or ‘AND’ are both allowed.

The following query criteria finds parts that have both a part category equal to Electrical and a lifecycle phase equal to Inactive.

```
[Title Block.Part Category] == 'Electrical' and
[Title Block.Lifecycle Phase] == 'Inactive'
```

If you replace the ‘and’ operator with ‘or’, the query locates all parts with either a part category of Electrical or a lifecycle phase of Inactive, which could be a large number of parts.

```
[Title Block.Part Category] == 'Electrical' or
[Title Block.Lifecycle Phase] == 'Inactive'
```

Note The Agile API provides three where-used set operators. For more information, see Creating a Where-Used Query on page 63.
Logical operators, including the where-used set operators, are not localized. You must use English keywords, regardless of locale.

Using Wildcard Characters with the Like Operator

If you define a search condition using the ‘like’ operator, you can use two wildcard characters: the asterisk (*) and question mark (?). The asterisk matches any string of any length, so *at finds cat, splat, and big hat. For example, `[Title Block.Description] like '*book*'` returns all objects that contain the word “book,” such as textbook, bookstore, books, and so on.

The question mark matches any single character, so ?at finds hat, cat, and fat, but not splat. For example, `[Title Block.Description] like '?al*'` matches any word containing “al” that is preceded by a single letter, such as tall, wall, mall, calendar, and so on.

Using Parentheses in Search Criteria

Where-used, set operators have higher priority than and and or logical operators, as shown by the following table.

Priority	Operator(s)
1	▫ union ▫ intersection ▫ minus
2	▫ and ▫ or

Therefore, search conditions joined by union, intersection, and minus operators are evaluated before conditions joined by and or or.

If you use where-used set operators (‘union’, ‘intersect’, or ‘minus’) in search criteria, you can use parentheses to change the order that criteria are evaluated. If only ‘and’ or ‘or’ logical operators are used in a search criteria, additional parentheses aren’t needed because they don't change the result of criteria evaluation.

The following two criteria, although they contain the same search conditions, provide different results because parentheses are placed differently:

```
(([Title Block.Part Category] == 'Electrical' and
  [Title Block.Description] contains 'Resistor') union
(([Title Block.Description] contains '400' and
  [Title Block.Product Line(s)] contains 'Taurus')
  [Title Block.Part Category] == 'Electrical' and
([Title Block.Description] contains 'Resistor' union
 [Title Block.Description] contains '400') and
  [Title Block.Product Line(s)] contains 'Taurus'
```

Setting Search Criteria for Lists Containing Large Number of Objects

When using the SDK to query lists that contain a large number of objects, you can improve performance if you use the object ID in the query criteria to set the value for the list

For example, you can replace this routine:

```
query.setCriteria("[Page Three.List25] equal to 'Administrator
(admin)'");
```

with the following for better performance:

```
IUser user = (IUser)session.getObject(IUser.OBJECT_TYPE, "admin");
query.setCriteria("[Page Three.List25] equal to
"+user.getObjectId());
```

Using SQL Syntax for Search Criteria

In addition to its standard query language, the Agile API also supports SQL-like syntax for search criteria. If you're familiar with how to write SQL statements, you may find this extended query language easier to work with, more flexible, and more powerful. It combines in one operation the specification of the query result attributes, the query class, the search condition, and the sort column(s).

This is a simple example that demonstrates the syntax:

- Query result attributes: `SELECT [Title Block.Number], [Title Block.Description]`
- Query class: `FROM [Items]`
- Search condition: `WHERE [Title Block.Number] starts with 'P'`
- Sort column(s): `ORDER BY 1 asc`

To improve readability, it's recommended that SQL key words such as SELECT and FROM are all typed using capital letters and each part of the statement appears on a separate line. This is merely a convention, not a requirement. SQL key words are not case-sensitive, and you can write the entire query string on one line if you prefer.

The best way to demonstrate the advantages of SQL syntax is to compare the code for a query that uses standard Agile API query syntax for search criteria with one that uses SQL syntax. The following example shows a query created using the standard Agile API query syntax:

Example: Query using standard Agile API query syntax

```
try {
  IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
"Items");
  query.setCriteria("[Page Two.Nummeric01] between (1000, 2000)");
  //Set result attributes
  String[] attrs = { "Title Block.Number", "Title Block.Description",
    "Title Block.Lifecycle Phase" };
  query.setResultAttributes(attrs);
  //Run the query
  ITable results = query.execute();
} catch (APIException ex) {
  System.out.println(ex);
}
```

This example shows the same query rewritten in SQL syntax. Although the example doesn't have fewer lines of code, you may find that it's more readable than Agile API query syntax, particularly if you're familiar with SQL.

Example: Query using SQL syntax

```
try {
  IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
  "SELECT " +
      "[Title Block.Number],[Title Block.Description], " +
      "[Title Block.Lifecycle Phase] " +
    "FROM " +
      "[Items] " +
    "WHERE " +
      "[Title Block.Number] between (1000, 2000)"
  );
  //Run the query
  ITable results = query.execute();
} catch (APIException ex) {
  System.out.println(ex);
}
```

The following example shows a query written with SQL syntax that specifies the search criteria using the `ATT_CRITERIA_STRING` query attribute. For more information about how to use query attributes, see Specifying Query Attributes when Creating a Query on page 39.

Example: Using SQL syntax to specify query attributes

```
try {
  String statement =
    "SELECT " +
      "[Title Block.Number], [Title Block.Description] " +
    "FROM " +
      "[Items] " +
    "WHERE " +
      "[Title Block.Description] like %0";
  HashMap map = new HashMap();
  map.put(QueryConstants.ATT_CRITERIA_STRING, statement);
  map.put(QueryConstants.ATT_CRITERIA_PARAM, new Object[] { "Comp*" }
);
  IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
map);
  ITable results = query.execute();
} catch (APIException ex) {
  System.out.println(ex);
}
```

Note Remember, the FROM part of the search condition specifies the query class. If you use the `ATT_CRITERIA_CLASS` attribute to also specify a query class, the query class specified in the SQL search condition takes precedence.

Although you can use the `IQuery.setCriteria()` method to specify a search condition in SQL syntax, the `IQuery.getCriteria()` method always returns the search condition in the standard Agile API query syntax.

Using SQL Wildcards

You can use both the asterisk (*) and question mark (?) wildcards in a query that uses SQL syntax. As in standard Agile API query language, the asterisk matches any string and the question mark matches any single character. You can use wildcards in the SELECT statement (the specified query result attributes) and the WHERE statement (the search condition). For example, "SELECT *" specifies all available query result attributes.

Sorting Query Results Using SQL Syntax

If you specify search criteria using SQL syntax instead of the standard Agile API query language, you can use the ORDER BY keyword to sort the query results. You can sort the results in ascending or descending order by any attributes specified in the SELECT statement.

In the ORDER BY statement, refer to attributes by the one-based numerical order in which they appear in the SELECT statement. To specify whether to sort in ascending or descending order, type “asc” or “desc” after the attribute number. If “asc” or “desc” is omitted, ascending order is used by default.

Example	Description
ORDER BY 1	Sort by the first SELECT attribute in ascending order (the default)
ORDER BY 2 desc	Sort by the second SELECT attribute in descending order
ORDER BY 1 asc, 3 desc	Sort by the first SELECT attribute in ascending order and the third SELECT attribute in descending order

Attributes not specified in the SELECT statement cannot be used to sort query results. Also, if you use "SELECT *" to select all available result attributes, the results cannot be sorted because the attribute order is undefined.

The following example sorts results in ascending order by [Title Block.Number] and [Title Block.Sites], the first and third attributes in the SELECT statement.

Example: Using SQL syntax to sort query results

```
IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
  "SELECT " +
    "[Title Block.Number],[Title Block.Description], " +
    "[Title Block.Sites],[Title Block.Lifecycle Phase] " +
  "FROM " +
    "[Items] " +
  "WHERE " +
    "[Title Block.Number] between (1000, 2000)" +
 "ORDER BY " +
    "1, 3"
);
```

Setting Result Attributes for a Query

When you run a query, it returns several output fields, which are also called result attributes. By default, there are only a few result attributes for each query class. You can add or remove result attributes using the `IQuery.setResultAttributes()` method.

The following table shows the default query result attributes for each predefined Agile PLM class.

Query class	Default result attributes
Changes Change Orders ECO Change Requests ECR Deviations Deviation Manufacturer Orders MCO Price Change Orders PCO Sites Change Orders SCO Stop Ships Stop Ship	Cover Page.Change Type Cover Page.Number Cover Page.Description Cover Page.Status Cover Page.Workflow
Customers Customers Customer	General Info.Customer Type General Info.Customer Number General Info.Customer Name General Info.Description General Info.Lifecycle Phase

Query class	Default result attributes
Declarations Homogeneous Material Declarations Homogeneous Material Declaration IPC 1752-1 Declarations IPC 1752-1 Declaration IPC 1752-2 Declarations IPC 1752-2 Declaration JGPSSI Declarations JGPSSI Declaration Part Declarations Part Declaration Substance Declarations Substance Declaration Supplier Declarations of Conformance Supplier Declaration of Conformance	Cover Page.Name Cover Page.Description Cover Page.Supplier Cover Page.Status Cover Page.Workflow Cover Page.Compliance Manager Cover Page.Due Date Cover Page.Declaration Type
Discussions Discussions Discussion	Cover Page.Subject Cover Page.Status Cover Page.Priority Cover Page.Type
File Folders File Folders File Folder	Title Block.Type Title Block.Number Title Block.Description Title Block.Lifecycle Phase
Items Parts Part Documentation Document	Title Block.Item Type Title Block.Number Title Block.Description Title Block.Lifecycle Phase Title Block.Rev

Query class	Default result attributes
Manufacturers Manufacturers Manufacturer	General Info.Name General Info.City General Info.State General Info.Lifecycle Phase General Info.URL
Manufacturer Parts Manufacturer Parts Manufacturer Part	General Info.Manufacturer Part Number General Info.Manufacturer Name General Info.Description General Info.Lifecycle Phase
Packages Packages Package	Cover Page.Package Number Cover Page.Description Cover Page.Assembly Number Cover Page.Status Cover Page.Workflow
Part Groups Part Groups Commodity Part Family	General Info.Name General Info.Description General Info.Lifecycle Phase General Info.Commodity Type General Info.Overall Compliance
Prices Published Prices Contracts Published Price Quote History Quote History	General Info.Price Number General Info.Description General Info.Rev General Info.Price Type General Info.Lifecycle Phase General Info.Projects General Info.Customer General Info.Supplier
Product Service Requests Non-Conformance Reports NCR Problem Reports Problem Report	Cover Page.PSR Type Cover Page.Number Cover Page.Description Cover Page.Status Cover Page.Workflow

Query class	Default result attributes
Projects Activities Phase Program Project Task Gates Gate	General Info.Name General Info.Description General Info.Status General Info.Health General Info.Owner General Info.Root Parent General Info.Workflow General Info.Type
Sourcing Projects Sourcing projects Sourcing Projects	General Info.Project Type General Info.Number General Info.Description General Info.Manufacturing Site General Info.Ship To Location General Info.Projects General Info.Customer General Info.Lifecycle Phase
Quality Change Requests Corrective Action/Preventive Action CAPA Audits Audit	Cover Page.QCR Type Cover Page.QCR Number Cover Page.Description Cover Page.Status Cover Page.Workflow
RFQ Responses RFQ Responses RFQ Response	Cover Page.RFQ Number Cover Page.RFQ Description Cover Page.Lifecycle Phase Cover Page.Requested Cover Page.Completed Cover Page.Due Date

Query class	Default result attributes
RFQs RFQs RFQ	Cover Page.RFQ Number Cover Page.RFQ Description Cover Page.MFG Site Cover Page.Ship-To Location Cover Page.Projects Cover Page.Customer Cover Page.Lifecycle Phase Cover Page.RFQ Type
Sites Sites Site	General Info.Name General Info.Contact General Info.Phone
Specifications Specifications Specification	General Info.Name General Info.Description General Info.Lifecycle Phase General Info.Jurisdictions General Info.Validation Type General Info.Specification Type
Substances Materials Material Subparts Subpart Substance Groups Substance Group Substances Substance	General Info.Name General Info.Description General Info.CAS Number General Info.Lifecycle Phase General Info.Substance Type
Suppliers Suppliers Component Manufacturer Contract Manufacturer Distributor Manufacturer Rep	General Info.Supplier Type General Info.Number General Info.Name General Info.Description General Info.Status

Query class	Default result attributes
Transfer Orders Content Transfer Orders CTO Automated Transfer Orders ATO	Cover Page.Transfer Order Type (See Retrieving CTO Originator Name on page 61) Cover Page.Transfer Order Number Cover Page.Description Cover Page.Status Cover Page.Workflow

Specifying Result Attributes

If you run a query and find that the resulting `ITable` object does not contain the attributes you expected, it's because you didn't specify result attributes. The following example shows how to specify the result attributes for a query.

Example: Setting query result attributes

```
private void setQueryResultColumns(IQuery query) throws APIException {
  // Get Admin instance
  IAdmin admin = m_session.getAdminInstance();

  // Get the Part class
  IAgileClass cls = admin.getAgileClass("Part");

  // Get some Part attributes, including Page Two and Page Three
attributes
  IAttribute attr1 =
cls.getAttribute(ItemConstants.ATT_TITLE_BLOCK_NUMBER);
  IAttribute attr2 =
cls.getAttribute(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  IAttribute attr3 =
cls.getAttribute(ItemConstants.ATT_TITLE_BLOCK_LIFECYCLE_PHASE);
  IAttribute attr4 =
cls.getAttribute(ItemConstants.ATT_PAGE_TWO_TEXT01);
  IAttribute attr5 =
cls.getAttribute(ItemConstants.ATT_PAGE_TWO_NUMERIC01);
  IAttribute attr6 =
cls.getAttribute(ItemConstants.ATT_PAGE_THREE_TEXT01);
  // Put the attributes into an array
  IAttribute[] attrs = {attr1, attr2, attr3, attr4, attr5, attr6};
  // Set the result attributes for the query
  query.setResultAttributes(attrs);

}
```

The `IQuery.setResultAttributes()` method takes an `Object[]` value for the `attrs` parameter, supporting `String, Integer,`, or `IAttribute` arrays. Therefore, instead of specifying an array of `IAttribute` objects, you can also specify an array of attribute names (such as `{"Title Block.Description", "Title Block.Number"}`) or attribute ID constants.

The following example shows how to specify result attributes using ID constants.

Example: Setting query result attributes by specifying ID constants

```
private void setQueryResultColumns(IQuery query) throws APIException {
  // Put the attribute IDs into an array
  Integer[] attrs = { ItemConstants.ATT_TITLE_BLOCK_NUMBER,
                      ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION,
                      ItemConstants.ATT_TITLE_BLOCK_LIFECYCLE_PHASE,
                      ItemConstants.ATT_PAGE_TWO_TEXT01,
                      ItemConstants.ATT_PAGE_TWO_NUMERIC01,
                      ItemConstants.ATT_PAGE_THREE_TEXT01 };
  // Set the result attributes for the query
  query.setResultAttributes(attrs);
}
```

When you use the `setResultAttributes()` method, make sure you specify valid result attributes. Otherwise, the `setResultAttributes()` method will fail. To get an array of available result attributes that can be used for a query, use `getResultAttributes()`, as shown in the following example.

Example: Getting the array of available result attributes

```
private IAttribute[] getAllResultAttributes(IQuery query) throws
APIException {
  IAttribute[] attrs = query.getResultAttributes(true);
  return attrs;
}
```

Retrieving CTO Originator Name

The Cover Page of the Content Transfer Order (CTO) includes the Originator field which specifies roles and site assignments of users who originate CTOs. To retrieve the user name, you can not query this field directly and need to retrieve data in `UserConstants`. For example, the following statement which attempts to retrieve the user name directly, will not work:

```
QueryString = ("[Cover Page.Originator] equal to '<Last_name>,
<First_name>'");
```

But the following statements which also specify the data in `UserConstants` will work:

```
QueryString = "[Cover Page.Originator] in
(["+UserConstants.ATT_GENERAL_INFO_USER_ID+"]=='<UserID>')";
```

Or,

```
QueryString = "[Cover Page.Originator] in
(["+UserConstants.ATT_GENERAL_INFO_LAST_NAME+"]=='<Last_name>'"+
                  "&&
["+UserConstants.ATT_GENERAL_INFO_FIRST_NAME+"]=='<First_name>');
```

The query criteria for any innumerable attribute type such as `IItem`, `IChange`, and so on, must be in a nested form. This applies to the Originator attribute which points to Agile All users.

Duplicate Results for Site-Related Objects and AMLs

The manufacturing sites functionality of the Agile Application Server can have unintended results when you search for items or changes. If you search for items or changes and include a sites attribute—`[Title Block.Site]` for items and `[Cover Page.Site(s)]` for changes—in the result attributes, the query results include duplicate objects for each site associated with the object. Similarly, if you search for items and include an AML attribute—such as `[Manufacturers.Mfr. Part Number]`—in the result attributes, the query results include duplicate items for each manufacturer part listed on an item's Manufacturers table.

For example, a part with the number 1000-02 has five sites associated with it. If you search for that part and include `Title Block.Site` in the result attributes, the resulting `ITable` object returned by the `IQuery.execute` method contains five rows, not one. Each row references the same object, part number 1000-02, but the Site cell has a different value. If you use `ITable.getReferentIterator` to iterate through referenced objects in the search results, the duplicate objects would be more apparent; in this example, you would iterate over the same item five times.

Working with Query Results

When you run a query, the Agile API returns an `ITable` object, which extends `java.Util.Collection`. You can use the methods of `ITable` and of `java.Util.Collection` to work with the results. For example, the following code shows how to use the `Collection.iterator()` method.

```
Iterator it = query.execute().iterator();
```

```
The ITwoWayIterator interface lets you traverse the list of rows in
either direction using the next() and previous() methods.
ITwoWayIterator it = query.execute().getTableIterator();
ITwoWayIterator it = query.execute().getReferentIterator();
```

For more information about using `ITwoWayIterator`, see Iterating Over Table Rows on page 76.

Sorting Query Results

Unlike other Agile API tables, you cannot create a sorted iterator for query results using the `ITable.ISortBy` interface. To sort query results, use SQL syntax and specify an `ORDER BY` statement with the search criteria. For more information, see Using SQL Syntax for Search Criteria on page 52.

Query Result Datatypes

Values in a query results table have the same datatype as their attributes. That is, if an attribute's datatype is an `Integer`, its value in a query results table is also an `Integer`.

Important	Remember that in Agile 9.0 SDK, all values in a query results table are strings. In post Agile 9.2, these values are integers.

Managing Large Query Results

Agile PLM has a system preference named Maximum Query Results Displayed that sets a limit on the maximum number of rows that can be returned from any query. However, that preference doesn't affect Agile SDK Clients. Queries that you run from an Agile SDK Client always return all results.

Although you can access the entire query result set with the returned `ITable` object, the Agile API internally manages retrieving partial results as necessary. For example, let's say a particular query returns 5000 records. You can use the `ITable` interface to access any of those 5000 rows. You don't need to worry about how many of the 5000 rows the Agile API actually loaded into memory.

Note Searches that you run from other Agile PLM Clients, such as Agile Web Client, adhere to the limit imposed by the Maximum Query Results Displayed preference.

Query Performance

The response time for running queries can be the biggest bottleneck in your Agile API program. To improve performance, you should try to construct queries that return no more than a few hundred results. A query that returns more than a 1000 results can take several minutes to finish processing. Such queries also eat up valuable processing on the Agile Application Server, potentially slowing down your server for all users.

Creating a Where-Used Query

Previous sections of this chapter described how to create queries that search for Agile PLM objects, for example, items or changes. You can also create where-used queries. In a where-used query, the search conditions define the items that appear on the BOMs of objects. You can use a where-used query to find the assemblies on which a particular part is used.

The interface for a where-used query is similar to a standard object query. With minor changes, you can turn an object query into a where-used query as long as the query class is an Item class.

Note Where-used queries are only defined for Item classes.

To define a where-used query, use the `IQuery.setSearchType()` method. You can also use the following logical operators, also called where-used set operators, to further define the relationships between grouped sets of search conditions. Only one logical operator can be used for each search condition.

Where Used set operator	Description
`intersect`	Produces records that appear in both result sets from two different groups of search conditions.
`minus`	Produces records that result from the first group of search conditions but not the second.
`union`	Produces records that are the combination of results from two groups of search conditions.

Note Where-used set operators have higher priority than other logical operators. Therefore, search conditions joined by where-used set operations are evaluated before those joined by ‘and’ or ‘or’ operators.

Example: Where-used query

```
void btnFind_actionPerformed(ActionEvent e) {
  try {
// Create the query
 IQuery wuquery =
    (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
    ItemConstants.CLASS_ITEM_BASE_CLASS);
// Set the where-used type

wuquery.setSearchType(QueryConstants.WHERE_USED_ONE_LEVEL_LATEST_RELEAS
ED);
// Add query criteria
 wuquery.setCriteria(
    "[Title Block.Part Category] == 'Electrical'" +
    "and [Title Block.Description] contains 'Resistor'" +
    "union [Title Block.Description] contains '400'" +
    "and [Title Block.Product Line(s)] contains 'Taurus'");
    // Run the query
    ITable results = wuquery.execute();

    // Add code here to display the results
  }
  catch (APIException ex) {System.out.println(ex);}
}
```

Loading a Query

There are two ways to load a query:

- Use the `IAgileSession.getObject()` method to specify the full path of a query.
- Use the `IFolder.getChild()` method to specify the location of a query relative to a folder.

The following example shows how to load a query by specifying its full path.

Example: Loading a query using IAgileSession.getObject()

```
try {

//Load the "Changes Submitted to Me" query
 IQuery query =
    (IQuery)m_session.getObject(IQuery.OBJECT_TYPE,
         "/Workflow Routings/Changes Submitted To Me");
} catch (APIException ex) {
 System.out.println(ex);
}
```

The following example shows how to load a query by specifying its path relative to a folder, in this case the user's Public In-box folder.

Example: Loading a query using IFolder.getChild()

```
try {
//Get the Workflow Routings folder
 IFolder folder =
    (IFolder)m_session.getObject(IFolder.OBJECT_TYPE, "/Workflow
    Routings");
//Load the "Changes Submitted to Me" query
 IQuery query =
    (IQuery)folder.getChild("Changes Submitted To Me");
} catch (APIException ex) {
    System.out.println(ex);
}
```

Deleting a Query

To delete a saved query, use the `IQuery.delete()` method. Temporary queries, that is, queries that are created but not saved to a folder are automatically deleted after the user session is closed. For lengthy sessions, you can use the `delete()` method to explicitly delete a temporary query after you're finished running it.

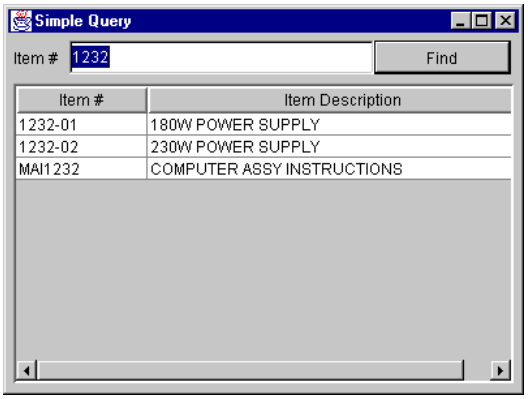
Example: Deleting a query

```
void deleteQuery(IQuery query) throws APIException {
  query.delete();
}
```

Simple Query Examples

Figure below depicts an example of dialog box that performs a simple query.

Figure 4: Simple Query dialog box



The Simple Query dialog box lets the user specify an item number to search for. When the user clicks the Find button, the program constructs a query to find all items that contain the specified text in the `Item Number` field. This example shows the code that runs the query when the user clicks the Find button.

Example: Simple Query code

```
void btnFind_actionPerformed(ActionEvent e) {
  try {
    // Create the query
    IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
      ItemConstants.CLASS_ITEM_BASE_CLASS);
    // Turn off case-sensitivity
    query.setCaseSensitive(false);

    // Specify the criteria data
    query.setCriteria("[Title Block.Number] contains (%0)",
    new String[] { this.txtItemNum.getText().toString() });
   // Run the query
   ITable queryResults = query.execute();
   Iterator i = queryResults.iterator();

   // If there are no matching items, display an error message.
   if (!i.hasNext()) {
     JOptionPane.showMessageDialog(null, "No matching items.", "Error",
     JOptionPane.ERROR_MESSAGE);
     return;
   }

   // Define arrays for the table data
   final String[] names = {"Item Number", "Item Description"};
   final Object[][] data = new Object[resultCount][names.length];
   int j = 0;
   while (i.hasNext()) {
     IRow row = (IRow)i.next();
     data[j][0] =
row.getValue(ItemConstants.ATT_TITLE_BLOCK_NUMBER).toString();
     data[j][1] =
row.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRITPION).toString();
     j++;
   }
   catch (APIException ex) {
    System.out.println(ex);
   }

   // Create a table model
   TableModel newDataModel = new AbstractTableModel() {
     // Add code here to implement the table model
   };

   // Populate the table with data from the table model
   myTable.setModel(newDataModel);
}
```

Chapter 4

Working with Tables

This chapter includes the following:

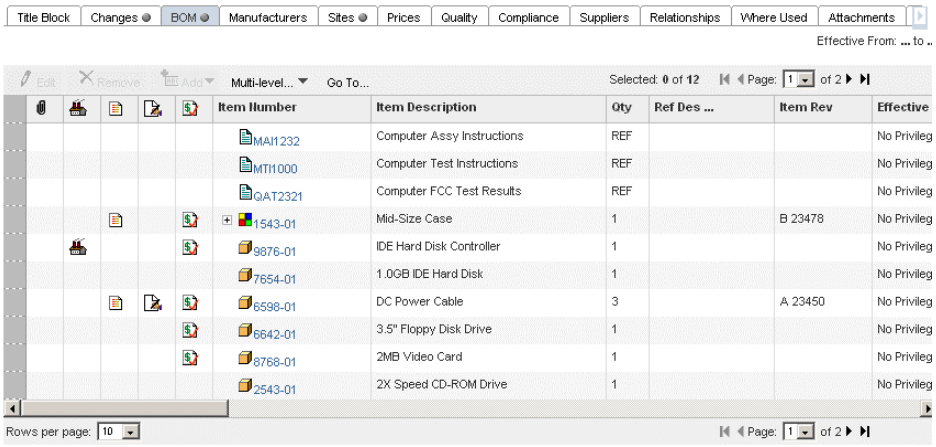
- About Tables 67
- Retrieving a Table 68
- Accessing the New and Merged Relationships Tables 69
- Retrieving the Metadata of a Table 71
- Adding Table Rows 71
- Adding and Updating Multiple Table Rows 74
- Iterating Over Table Rows 76
- Sorting Table Rows 78
- Removing Table Rows 79
- Retrieving the Referenced Object for a Row 80
- Checking Status Flags of a Row 84
- Working with Page 1, Page 2, and Page 3 85
- Redlining 85
- Removing Redline Changes 88
- Identifying Redlined Rows and Redlined Cells 89

About Tables

Whenever you work with an Agile PLM object in your program, you inevitably need to get and display the object's data. The data is contained in one or more tables. In Agile Web Client, these tables are equivalent to the separate tabs in a window, such as the Manufacturers and BOM tabs.

The following figure shows the BOM tab for an item in Agile Web Client.

Figure 5: BOM tab for Item



In some cases, a tab in Agile Web Client contains multiple tables. For example, the Changes tab for

an item contains the Pending Changes table and the Change History table. The tabs and the tables that they contain is not always the same for different Agile products. Also, they are not the same for each Agile PLM Dataobject. For example, tables for Parts objects are different from tables for Manufacturers objects. See Retrieving a Table on page 68.

To work with data in an Agile PLM table, follow these basic steps:

1. Create or get an object (for example, an item or a change order).
2. Retrieve a table (for example, the BOM table).
3. Iterate through the table rows to retrieve a row.
4. Get or set one or more attribute values for the selected row.

`ITable`, like `IFolder`, extends `java.util.Collection` and supports all the methods provided by that superinterface. This means that you can work with an `ITable` object as you would any Java `Collection`.

Interface	Inherited methods
`java.util.Collection`	`add(), addAll(), clear(), contains(), containsAll(), equals(), hashCode(), isEmpty(), iterator(), remove(), removeAll(), retainAll(), size(), toArray(), toArray()`

Retrieving a Table

After you create or get an object, you can use the `IDataObject.getTable()` method to retrieve a particular Agile PLM table. `IDataObject` is a general-purpose object that represents any Agile PLM object that contains tables of data. It is a superinterface of several other objects, including `IItem`, `IChange`, and `IUser`.

Note When retrieving PG&C's Supplier Declaration of Conformance (SDOC) tables, `IDataObject.getTable()` retrieves all 14 SDOC tables belonging to this base class. However, six of these tables (Items, Manufacturer Parts, Part Groups, Item Composition, Manufacturer Part Composition, Part Group Composition) are not enabled.

Tables vary for each Agile PLM dataobject. Tables for change objects are different from tables for items. Each table for a particular dataobject is identified by a constant in the constants class for that dataobject. Item constants are contained in the `ItemConstants` class, change constants are contained in the `ChangeConstants` class, and so on.

For information to use these tables, refer to the following Agile product administration documents:

- *Getting Started with Agile PLM*
- *Agile PLM Administrator Guide*
- *Agile PLM Product Governance & Compliance User Guide*
- *Agile PLM Product Portfolio Management User Guide*

Accessing the New and Merged Relationships Tables

In Release 9.2.2, the following tables were merged into a single table called the `Relationships` table.

- `Relationships.AffectedBy`
- `Relationships.Affects`
- `Relationships.Reference`

In addition, the constants that are used by these tables (`TABLE_REFERENCES`, `TABLE_RELATIONSHIPSAFFECTS`, and `TABLE_RELATIONSHIPSAFFECTEDBY`) were also removed. If you need these constants, you must rewrite them in your routines.

Note For a complete list of table constants that are merged and mapped into a single constants, or mapped into a new constant, see Migrating Release 9.2.1 and Older Table Constants to Release 9.2.2 on page 447.

For information to use these tables, refer to the following Agile documents:

- To use these tables in Agile PLM products, refer to *Getting Started with Agile PLM* and *Agile PLM Administrator Guide*
- To use these tables in Agile PPM products, refer to *Agile PLM Product Portfolio Management User Guide*

Accessing the Relationships Table

The `IRelationshipContainer` interface was implemented to access this table. Any Agile business object that contains the Relationships table implements this interface. You can access this table using IRelationshipContainer, or `IDataObject.getTable()` with `CommonConstants.TABLE_RELATIONSHIPS` constant.

```
IRelationshipContainer container = (IRelationshipContainer) object;
ITable relationship = container.getRelationship();
```

Accessing the Merged Tables

If you used these tables in previous releases of Agile PLM, and require the functionalities that they provided, modify your code as shown below.

Accessing the Merged Relationships.AffectedBy Table

- Code used in Release 9.2.1.x and earlier releases:

```
ITable affectedBy =
object.getTable(ChangeConstants.TABLE_RELATIONSHIPSAFFECTEDBY);
```

- Code recommended for this release:

```
ITable affectedBy =
object.getTable(CommonConstants.TABLE_RELATIONSHIPS)
.where("[2000007912] == 1", null);
```

Accessing the Merged Relationships.Affects table

- Code used in Release 9.2.1.x and earlier releases:

```
ITable affects =
object.getTable(ChangeConstants.TABLE_RELATIONSHIPSAFFECTS);
```

- Code recommended for this release:

```
ITable affects =
object.getTable(CommonConstants.TABLE_RELATIONSHIPS)
.where("[2000007912] == 2", null);
```

Accessing the Merged Relationships.References Table

- Code used in Release 9.2.1.x and earlier releases:

```
ITable references =
object.getTable(ChangeConstants.TABLE_RELATIONSHIPS_REFERENCES);
```

- Code recommended for this release:

```
ITable references =
object.getTable(CommonConstants.TABLE_RELATIONSHIPS)
.where("[2000007912] == 3", null);
```

Important The `ITable.where()` method is certified for deployment with these three tables only, and it may fail if it is used to access other tables from the SDK.

The following example shows how to retrieve and print the BOM table for an item.

Example: Retrieving the BOM table

```
//Load an item
private static IItem loadPart(String number) throws APIException {
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
number);
  return item;
}
//Get the BOM table
private static void getBOM(IItem item) throws APIException {
  IRow row;
  ITable table = item.getTable(ItemConstants.TABLE_BOM);
  Iterator it = table.iterator();
  while (it.hasNext()) {
    row = (IRow)it.next();
    //Add code here to do something with the BOM table
  }
}
```

Working with Read-only Tables

Several Agile PLM tables store history information or data about related objects. These tables are read-only and as such, you cannot modify these tables. When you write code to access a table, use the `ITable.isReadOnly()` method to check if the table is read-only.

Retrieving the Metadata of a Table

The `ITableDesc` is an interface that represents the metadata of a table which is the underlying data that describes a table's properties. `ITableDesc` is related to `ITable` in the same way that `IAgileClass` is related to `IDataObject`. At times you may need to identify the attributes for a particular table, its ID, or its table name without loading a dataobject. The following example shows how to use the `ITableDesc` interface to retrieve the collection of all attributes (including ones that aren't visible) for a table.

Example: Retrieving the metadata of a table

```
private IAttribute[] getBOMAttributes() throws APIException {
  IAgileClass cls = admin.getAgileClass(ItemConstants.CLASS_PART);
  ITableDesc td = cls.getTableDescriptor(ItemConstants.TABLE_BOM);
  IAttribute[] attrs = td.getAttributes();
  return attrs;
}
```

You can also use the API Name field to identify a table's name or ID. For information to use this field, see Accessing PLM Metadata with APIName Field on page 123. For information to use the Agile API to work with metadata, see Performing Administrative Tasks on page 319.

Adding Table Rows

To create a table row, use the `ITable.createRow(java.lang.Object)` method, which creates a new row and initializes it with the data specified in the `param` parameter. The `param` parameter of `createRow` is available to pass the following data:

- a set of attributes and values for the row's cells
- files or URLs to add to the Attachments table
- an Agile PLM object (such as an `IItem`) to add to the table

When you add a row to a table, it's not necessarily added at the end of the table.

Note	There is also a deprecated, parameter-less version of the `createRow()` method, which creates an empty row. Avoid using that method because it may not be supported in future Agile PLM releases. You must initialize a row with data when you create it.

You can also add table rows in batch format with `ITable.createRow()`. See Adding and Updating Multiple Table Rows on page 74.

Adding an Item to the BOM Table

The following example shows how to use the `ITable.createRow()` method to add an item to a BOM table.

Example: Adding a row and setting values

```
private static void addToBOM(String number) throws APIException {
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
number);
  ITable table = item.getTable(ItemConstants.TABLE_BOM);
  Map params = new HashMap();
  params.put(ItemConstants.ATT_BOM_ITEM_NUMBER, "1543-01");
  params.put(ItemConstants.ATT_BOM_QTY, "1");
  item.setManufacturingSite(ManufacturingSiteConstants.COMMON_SITE);
  IRow row = table.createRow(params);
}
```

Note To add a site-specific row to the BOM table, use `IManufacturerSiteSelectable.setManufacturingSite()` to select a specific site before calling `ITable.createRow()`.

Adding an Attachment to the Attachments Table

The following example shows how to use the `ITable.createRow(java.lang.Object)` method to add a row to the Attachments table. The code adds a row to the table and initializes it with the specified file. After adding the row, the code also sets the value of the File Description field.

Example: Adding a row to the Attachments table

```
private static void addAttachmentRow(String number) throws APIException
{
  File file = new File("d:/MyDocuments/1543-01.dwg");
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
number);
  ITable table = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
  IRow row = table.createRow(file);
}
```

Adding a Manufacturer Part to the Manufacturers Table

The following example shows how to use the `ITable.createRow(java.lang.Object)` method to add a row to the Manufacturers table of an item. The code adds a row to the table and initializes it with the specified `IManufacturerPart` object.

Example: Adding a row to the Manufacturers table

```
private static void addMfrPartRow(String number) throws APIException {
  HashMap info = new HashMap();

info.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_PART_N
UMBER, "TPS100-256");
```

```
info.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_NAME,
"TPS_POWER");
  IManufacturerPart mfrPart = (IManufacturerPart)m_session.getObject(
    ManufacturerPartConstants.CLASS_MANUFACTURER_PART, info
  );
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
number);
  item.setManufacturingSite(ManufacturingSiteConstants.COMMON_SITE);
  ITable table = item.getTable(ItemConstants.TABLE_MANUFACTURERS);
  IRow row = table.createRow(mfrPart);
}
```

Note To add a site-specific row to the Manufacturers table, use `IManufacturerSiteSelectable.setManufacturingSite()` to select a specific site before calling `ITable.createRow()`.

Adding an Item to the Affected Items Table

The following example shows how to use the `ITable.createRow(java.lang.Object)` method to add a row to the Affected Items table of a change order. The code adds a row to the table and initializes it with the specified `IItem` object.

Example: Adding a row to the Affected Items table

```
private static void addItemRow(String number) throws APIException {
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
"P522-103");
  IChange change =
(IChange)m_session.getObject(ChangeConstants.CLASS_ECO, number);
  ITable table = change.getTable(ChangeConstants.TABLE_AFFECTEDITEMS);
  IRow row = table.createRow(item);
}
```

Because the BOM table also references `IItem` objects, you can use code similar to those in Example 49 to add a row to a BOM table.

Adding a Task to the Schedule Table

The following example shows how to use the ITable.createRow(java.lang.Object) method to add a row to the Schedule table of a Project. The code adds a row to the table and initializes it with the specified `IProgram` object.

Example: Adding a row to the Schedule table

```
private static void addTaskRow(IProgram program, IProgram task) throws
APIException {
  // Get the Schedule table of the program
  ITable table = program.getTable(ProgramConstants.TABLE_SCHEDULE);
  // Add the task to the schedule
  IRow row = table.createRow(task);
}
```

Adding and Updating Multiple Table Rows

The `ITable` interface provides two convenient methods for adding and updating multiple table rows with one API call:

- `ITable.createRows()`
- `ITable.updateRows()`

Because these methods group multiple table rows in one API call, they can improve performance by reducing the number of Remote Procedure Calls (RPCs), particularly if you are connecting to the server across a Wide Area Network (WAN). However, these methods do not result in efficient batch operations on the Agile Application Server, which simply iterates through each row being added or updated.

Important The `ITable.createRows()` and `ITable.updateRows()` methods are supported only when you are adding or updating multiple rows on the BOM table of items, or the Affected Items table of Changes.

Adding Multiple Items to the BOM Table

The following example shows how to use the `ITable.createRows()` method to add multiple items to a BOM table.

Example: Adding multiple rows and setting values

```
private static void createBOMRows(String partNumber) throws
APIException {
  IItem[] child = new IItem [3];
  IItem parent = null;
  ITable tab = null;

  // Get the parent item
  parent = (IItem) m_session.getObject(IItem.OBJECT_TYPE, partNumber);
  // Get the BOM table
  tab = parent.getTable(ItemConstants.TABLE_BOM);

  // Create child items
  child[0] = (IItem) m_session.createObject(ItemConstants.CLASS_PART,
partNumber + "-1");
  child[1] = (IItem) m_session.createObject(ItemConstants.CLASS_PART,
partNumber + "-2");
  child[2] = (IItem) m_session.createObject(ItemConstants.CLASS_PART,
partNumber + "-3");
  // Create a row array
  IRow[] rowArray = new IRow[3];

  // Add the items to the BOM
```

```
  rowArray = tab.createRows(new Object[]{child[0], child[1],
child[2]});
}
```

Note To add a site-specific row to the BOM table, use `IManufacturerSiteSelectable.setManufacturingSite()` to select a specific site before calling `ITable.createRow()`.

Updating Multiple BOM Rows

To update multiple rows, use the `ITable.updateRows()` method. This method batches together multiple update operations into a single call. Instead of calling `IRow.setValues()` for multiple rows in a table, this API updates an entire table in one method call.

The `rows` parameter of `updateRow()` can be used to pass a `Map` containing `IRow` instances as keys with instances for values. The value Map objects should have attribute IDs as keys and replacement data for values.

Example: Updating multiple BOM rows

```
private static void updateBOMRows(String partNumber) throws
APIException {
  IItem parent = null;
  ITable tab = null;
  HashMap[] mapx = new HashMap[3];
  Map rows = new HashMap();
  IRow[] rowArray = new IRow[3];

  // Get the parent item
  parent = (IItem) m_session.getObject(IItem.OBJECT_TYPE, partNumber);
  // Get the BOM table
  tab = parent.getTable(ItemConstants.TABLE_BOM);

  // Create three items
  IItem child1 = (IItem)
m_session.createObject(ItemConstants.CLASS_PART, partNumber + "-1");
  IItem child2 = (IItem)
m_session.createObject(ItemConstants.CLASS_PART, partNumber + "-2");
  IItem child2 = (IItem)
m_session.createObject(ItemConstants.CLASS_PART, partNumber + "-3");
  // Add these items to BOM table rowArray = tab.createRows(new
Object[]{child1, child2, child3});
  // New values for child[0]
  mapx[0] = new HashMap();
  mapx[0].put(ItemConstants.ATT_BOM_FIND_NUM, new Integer(1));
  mapx[0].put(ItemConstants.ATT_BOM_QTY, new Integer(3));
  mapx[0].put(ItemConstants.ATT_BOM_REF_DES, "A1-A3");
  rows.put(rowArray[0], mapx[0]);
  // New values for child[1]
  mapx[1] = new HashMap();
```

```
    mapx[1].put(ItemConstants.ATT_BOM_FIND_NUM, new Integer(2));
    mapx[1].put(ItemConstants.ATT_BOM_QTY, new Integer(3));
    mapx[1].put(ItemConstants.ATT_BOM_REF_DES, "B1-B3");
    rows.put(rowArray[1], mapx[1]);
    // new values for child[2]
    mapx[2] = new HashMap();
    String strA = "BOM-Notes" + System.currentTimeMillis();
    mapx[2].put(ItemConstants.ATT_BOM_BOM_NOTES, strA);
    mapx[2].put(ItemConstants.ATT_BOM_FIND_NUM, new Integer(3));
    rows.put(rowArray[2], mapx[2]);
    // Update the BOM table rows
    tab.updateRows(rows);
  }
```

Iterating Over Table Rows

When you use the Agile API to get a table, such as a BOM table, your program often needs to browse the rows contained in the table. To access an individual row, you first have to get an iterator for the table. You can then iterate over each row to set cell values.

The Agile API does not support random access of rows in a table. This means that you can't retrieve a specific row by index number and then update it. When you add or remove a row, the entire table is resorted and the existing table iterator is no longer valid.

To browse the data in table, create an iterator for the table using one of these methods:

- `ITable.iterator()` – returns an `Iterator` object, allowing you to traverse the table from the first row to the last.
- `ITable.getTableIterator()` – returns an `ITwoWayIterator` object, allowing you to traverse the table rows forwards or backwards. You can also use `ITwoWayIterator` to skip a number of rows. ITwoWayIterator is preferred over Iterator if your program displays table rows in a user interface.
- `ITable.getTableIterator(ITable.ISortBy[])` – returns a sorted `ITwoWayIterator` object.
- `ITable.getReferentIterator()` – returns an `ITwoWayIterator` object for the objects referenced in the table.

When you work with an iterator for a table, you don't need to know the total number of rows in the table. Instead, you work with one row at a time. Although the `ITable` interface provides a `size()` method, which calculates the total number of rows in the table, it's considered a resource extensive operation performance-wise and as such, is not recommended for large tables, particularly if your code already uses an iterator to browse the table.

The following example demonstrates how to get an iterator for a table and use `ITwoWayIterator` methods to traverse forwards and backwards over the table rows.

Example: Iterating over table rows

```
try {
  // Get an item
```

```
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
"1000-02");
  // Get the BOM table
  ITable bom = item.getTable(ItemConstants.TABLE_BOM);
  ITwoWayIterator i = bom.getTableIterator();
  // Traverse forwards through the table
  while (i.hasNext()) {
    IRow row = (IRow)i.next();
    // Add code here to do something with the row
  }
  // Traverse backwards through the table
  while (i.hasPrevious()) {
    IRow row = (IRow)i.previous();
    // Add code here to do something with the row
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

The `ITwoWayIterator` object allows a user interface to display table rows on multiple pages, which is perhaps more practical than the use of `ITwoWayIterator` shown in the preceding example. For example, instead of displaying a single scrolling page of several hundred BOM items, you can break the table into pages displaying 20 BOM items per page. To navigate from page to page, your program should provide navigation controls such as those shown in the figure below.

Figure 6: Navigation controls in the Agile Web Client

Updating Objects in Query Results with Multiple Page Tables

When you invoke `getReferentIterator` to update objects in search results tables that contain more than 200 results, `getReferentIterator` will not update all the objects that are returned by the query. For example, when you run a query to match a value in a field, and then edit the same value while iterating through the results with `getReferentIterator`, the query completes the first page with no problem. However, when it queries the remaining pages, some table rows are not updated. There are several ways to overcome this limitation. The following is one such example.

Example: Updating all table rows when iterating large query results

1. Increase the table page size for this query so that it can contain the results in a single page.

2. Run the query several times and keep updating the results until query results are empty.

3. Do not query on the same field that you are updating.

Updating Table Rows when Iterating Large Query Results

Follow steps in the following example.

To update all table rows when iterating large query results:

1. Increase the table page size for this query so that it can contain the results in a single page.
2. Run the query several times and keep updating the results until query results are empty.
3. Do not query on the same field that you are updating.

Sorting Table Rows

To sort the rows in a table by a particular attribute, use `getTableIterator(ITable.ISortBy[])` to return a sorted iterator. The `ISortBy` parameter of `getTableIterator()` is an array of `ITable.ISortBy` objects. To create an `ISortBy` object, use `createSortBy(IAttribute, ITable.ISortBy.Order)`. The `order` parameter of `createSortBy()` is one of the `ITable.ISortBy.Order` constants either `ASCENDING` or `DESCENDING`.

Note The Agile API allows you to sort a table by only one attribute. Therefore, the `ISortBy` array that you specify for the `ISortBy` parameter of `getTableIterator()` must contain only one `ISortBy` object.

The following example sorts the BOM table by the BOM | Item Number attribute.

Example: Sorting a table iterator

```
try {
  // Get an item
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
"1000-02");
  // Get the BOM table
  ITable bom = item.getTable(ItemConstants.TABLE_BOM);
  // Get the BOM | Item Number attribute
  IAgileClass cls = item.getAgileClass();
  IAttribute attr =
cls.getAttribute(ItemConstants.ATT_BOM_ITEM_NUMBER);

  // Specify the sort attribute for the table iterator
  ITable.ISortBy sortByNumber = bom.createSortBy(attr,
ITable.ISortBy.Order.ASCENDING);
  // Create a sorted table iterator
  ITwoWayIterator i = bom.getTableIterator(new ITable.ISortBy[]
{sortByNumber});
  // Traverse forwards through the table
  while (i.hasNext()) {
    IRow row = (IRow)i.next();
    // Add code here to do something with the row
```

```
    }
  } catch (APIException ex) {
    System.out.println(ex);
  }
```

The following Product Sourcing and Projects Execution objects load tables a bit differently and therefore cannot be sorted using the `getTableIterator(ITable.ISortBy[])` method. For any tables of these objects, create an unsorted iterator using the `iterator()` or `getTableIterator()` methods.

- `IDiscussion`
- `IPrice`
- `IProgram`
- `IProject`
- `IRequestForQuote`
- `ISupplier`
- `ISupplierResponse`

The `ITable.ISortBy` interface is not supported for query result tables. To sort query results, use SQL syntax and specify an `ORDER BY` statement with the search criteria. For more information, see Using SQL Syntax for Search Criteria on page 52.

Removing Table Rows

To remove a row from a table, use the `ITable.removeRow()` method, which takes one parameter, an `IRow` object. You can retrieve a row by iterating over the table rows.

If a table is read-only, you can't remove rows from it. For more information, see Working with Read-only Tables on page 70. If you are working with a released revision of an item, you can't remove a row from the item's tables until you create a change order for a new revision.

Example: Removing a table row

```
try {
  // get an item
  IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART,
"1000-02");
  // get the BOM table
  ITable bom = item.getTable(ItemConstants.TABLE_BOM);
  ITwoWayIterator i = bom.getTableIterator();
  // find the bom component 6642-01 and remove it
  while (i.hasNext()) {
    IRow row = (IRow)i.next();
    String bomitem =
(String)row.getValue(ItemConstants.ATT_BOM_ITEM_NUMBER);
    if (bomitem.equals("6642-01")) {
      bom.removeRow(row);
      break;
```

```
      }
    }
  } catch (APIException ex) {
    System.out.println(ex);
  }
```

Because `ITable` implements the `Collection` interface, you can use `Collection` methods to remove table rows. To remove all rows in a table, use `Collection.clear()`.

Example: Clearing a table

```
public void clearAML(IItem item) throws APIException {
  // Get the Manufacturers table
  ITable aml = item.getTable(ItemConstants.TABLE_MANUFACTURERS);
  // Clear the table
  aml.clear();

}
```

Retrieving the Referenced Object for a Row

Several Agile PLM tables contain rows of information that reference other Agile PLM objects. For example, the BOM table lists all items that are included in a Bill of Material. Each row of the BOM table represents an item. While working with a row on a BOM table, your program can allow the user to open the referenced item to view or modify its data.

Table below lists Agile PLM tables that reference other Agile PLM objects. All Agile PLM objects are referenced by number (for example, Item Number, Change Number, or Manufacturer Part Number).

Object	Table	Referenced Object(s)
IChange	Affected Items Affected Prices Attachments Relationships	IItem IPrice IAttachmentFile Multiple object types
ICommodity	Attachments Compositions Parts Specifications Substances Suppliers	IAttachmentFile IDeclaration IItem ISpecification ISubstance ISupplier
ICustomer	Attachments Related PSR	IAttachmentFile IServiceRequest

Object	Table	Referenced Object(s)
IDeclaration	Attachments Item Composition Items Manufacturer Part Composition Manufacturer Parts Part Group Composition Part Groups Relationships Specifications	IAttachmentFile ISubstance IItem ISubstance IManufacturerPart ISubstance ICommodity Multiple object types ISpecification
IDiscussion	Attachments Where Used	IAttachmentFile Not supported
IFileFolder	Files Relationships Where Used	IAttachmentFile Multiple object types Multiple object types
IItem	Attachments BOM Change History Compositions Manufacturers Pending Change Where Used Pending Changes Prices Quality Redline BOM Redline Manufacturers Sites Specifications Substances Where Used	IAttachmentFile IItem IChange IDeclaration IManufacturerPart IItem IChange IPrice IServiceRequest or IQualityChangeRequest IItem IManufacturerPart IManufacturingSite ISpecification ISubstance IItem

Object	Table	Referenced Object(s)
IManufacturerPart	Attachments Compositions Prices Specifications Substances Suppliers Where Used	IAttachmentFile IDeclaration IPrice ISpecification ISubstance ISupplier IItem
IManufacturer	Attachments Where Used	IAttachmentFile IManufacturerPart
IManufacturingSite	Attachments	IAttachmentFile
IPackage	Attachments	IAttachmentFile
IPrice	Attachments Change History Pending Changes	IAttachmentFile IChange IChange
IProgram	Attachments Deliverables - Affected By Deliverables - Affects Dependencies - Dependent Upon Dependencies - Required For Discussion Links Schedule Team	IAttachmentFile Multiple object types Multiple object types IProgram IProgram IDiscussion Multiple object types IProgram IUser and IUserGroup
IProject	Attachments BOM Item Changes Items Manufacturer Items Pending Change Responses RFQs	IAttachmentFile IItem IChange IItem IManufacturerPart IChange ISupplierResponse IRequestForQuote

Object	Table	Referenced Object(s)
IQualityChangeRequest	Affected Items Attachments PSR Items Relationships	IItem IAttachmentFile IItem Multiple object types
IRequestForQuote	Attachments	IAttachmentFile
IServiceRequest	Affected Items Attachments Related PSR Relationships	IItem IAttachmentFile IServiceRequest Multiple object types
ISpecification	Attachments Substances	IAttachmentFile ISubstance
ISubstance	Attachments Composition Where Used	IAttachmentFile ISubstance Multiple object types
ISupplierResponse	Attachments	IAttachmentFile
ISupplier	Attachments Manufacturers PSRs	IAttachmentFile IManufacturer IServiceRequest
ITransferOrder	Attachments Selected Objects	IAttachmentFile Multiple object types
IUser	Attachments Subscription User Group	IAttachmentFile Multiple object types IUserGroup
IUserGroup	Attachments Users	IAttachmentFile IUser

The following example shows how to retrieve the referenced `IChange` object from the Pending Changes table for an item.

Example: Retrieving a referenced Change object

```
void getReferencedChangeObject(ITable changesTable) throws APIException
{
  Iterator i = changesTable.iterator();
  while (i.hasNext()) {
    IRow row = (IRow)i.next();
    IChange changeObj = (IChange)row.getReferent();
    if (changeObj != null) {
```

```
      //Add code here to do something with the IChange object
    }
  }
}
```

The following example shows how the code in Example 57 can be simplified by using the `ITable.getReferentIterator()` method to iterate through the table's referenced objects.

Example: Iterating through referenced objects

```
void iterateReferencedChangeObjects(ITable changesTable) throws
APIException {
  Iterator i = changesTable.getReferentIterator();
  while (i.hasNext()) {
    IChange changeObj = (IChange)i.next();
    if (changeObj != null) {
      //Add code here to do something with the IChange object
    }
  }
}
```

Checking Status Flags of a Row

Sometimes you may want to perform an action on an object only if it meets certain status criteria. For example, if the selected object is a released change order, your program may not allow the user to modify it. To check the status of an object, use the `IRow.isFlagSet()` method. The `isFlagSet()` method returns a boolean value `true` or `false`.

Status flag constants are defined in the following classes:

- `CommonConstants` – Contains status flag constants common to Agile PLM objects.
- `ChangeConstants` – Contains status flag constants for `IChange` objects.
- `ItemConstants` – Contains status flag constants for `IItem` objects.

The following example shows how to use the `isFlagSet()` method to determine whether an item has attachments.

Example: Checking the status flag of an object

```
private static void checkAttachments(IRow row) throws APIException {
  try {
    boolean b;
    b = row.isFlagSet(CommonConstants.FLAG_HAS_ATTACHMENTS);
    if (!b) {
      JOptionPane.showMessageDialog(null, "The specified row does not
      have attached files.", "Error", JOptionPane.ERROR_MESSAGE);
    }
  } catch (Exception ex) {}
}
```

Working with Page 1, Page 2, and Page 3

Page One (that is, Title Block, Cover Page, and General Info pages), Page Two, and Page Three contain a single row of data and are therefore not tabular in format. All other tables contain multiple rows. Consequently, the data on Page One, Page Two, and Page Three is directly accessible. To get and set values for these pages, you don't need to get a table and then select a row. Instead, get a specified cell, and then use the `getValue()` and `setValue()` methods to display or modify the data.

If you prefer accessing data cells in a consistent way throughout your program, you can still use the Page One, Page Two, and Page Three tables to get and set values. The following example shows two methods that edit the values for several Page Two fields for an item. The first method retrieves the Page Two table and then sets the values for several cells. The second method accesses the Page Two cells directly by calling the `IDataObject.getCell()` method. Either approach is valid, but you can see that the second approach results in fewer lines of code.

Example: Editing Page Two cells

```
// Edit Page Two cells by first getting the Page Two table
private static void editPageTwoCells(IItem item) throws Exception {
  ICell cell = null;
  DateFormat df = new SimpleDateFormat("MM/dd/yy");
  ITable table = item.getTable(ItemConstants.TABLE_PAGETWO);
  Iterator it = table.iterator();
  IRow row = (IRow)it.next();
  cell = row.getCell(ItemConstants.ATT_PAGE_TWO_TEXT01);
  cell.setValue("Aluminum clips");
  cell = row.getCell(ItemConstants.ATT_PAGE_TWO_MONEY01);
  cell.setValue(new Money(new Double(9.95), "USD"));
  cell = row.getCell(ItemConstants.ATT_PAGE_TWO_DATE01);
  cell.setValue(df.parse("12/01/03"));
}
// Edit Page Two cells by calling IDataObject.getCell()
private static void editPageTwoCells2(IItem item) throws Exception {
  ICell cell = null;
  DateFormat df = new SimpleDateFormat("MM/dd/yy");
  cell = item.getCell(ItemConstants.ATT_PAGE_TWO_TEXT01);
  cell.setValue("Aluminum clips");
  cell = item.getCell(ItemConstants.ATT_PAGE_TWO_MONEY01);
  cell.setValue(new Money(new Double(9.95), "USD"));
  cell = item.getCell(ItemConstants.ATT_PAGE_TWO_DATE01);
  cell.setValue(df.parse("12/01/03"));
}
```

Redlining

When you issue a change for a released item or a price agreement, the Agile API lets you redline certain tables affected by the change. In the Agile PLM Clients, redline tables visually identify values that have been modified from the previous revision. Red underlined text—thus the term “redline”—indicates values that have been added, and red strikeout text indicates values that have been deleted. People responsible for approving the change can review the redline data.

The Agile PLM system provides the following redline tables:

- Redline BOM
- Redline Manufacturers (AML)
- Redline Price Lines
- Redline Title Block

Note The Web Client supports redlining the Item's Cover Page, Page Two, and Page Three tables together. However, in the SDK, these operations are performed separately, using different tables for each each page.

To redline BOM, Manufacturers, or Price Lines tables:

1. Get a released revision of an item or price object.
2. Create a new change, such as an ECO, MCO, SCO, or PCO
   - ECOs lets you modify an item’s BOM or Manufacturers tables
   - MCOs lets you modify an item’s Manufacturers table
   - SCOs let you modify an item’s site-specific BOM, Manufacturers
   - PCOs lets you modify a price’s Price Lines table
3. Add the item or price to the Affected Items or Affected Prices table of the change.
4. For ECOs and PCOs, specify the new revision for the change.
5. SCOs and MCOs do not affect an item’s revision.
6. Modify a redline table, such as the Redline BOM, Redline Manufacturers (AML), Redline Price Lines.

To redline the Manufacturers table of an item:

This example shows how to redline the Manufacturers table (AML) of an item.

Example: Redlining the Manufacturers table of an item

```
 private void redlineAML() throws APIException {
   IAttribute attrPrefStat = null;
   IAgileList listvalues = null;
   Map params = new HashMap();

   // Get a released item
   IItem item = (IItem)m_session.getObject("Part", "1000-02");
```

```
  // Get the Preferrred status value
  IAgileClass cls = item.getAgileClass();
  attrPrefStat =
cls.getAttribute(ItemConstants.ATT_MANUFACTURERS_PREFERRED_STATUS);
  listvalues = attrPrefStat.getAvailableValues();
  listvalues.setSelection(new Object[] { "Preferred" });
  // Create an MCO
  IChange change =
(IChange)m_session.createObject(ChangeConstants.CLASS_MCO, "M000024");
  // Set the Workflow ID of the MCO
  change.setWorkflow(change.getWorkflows()[0]);

  // Get the Affected Items table
  ITable affectedItems =
change.getTable(ChangeConstants.TABLE_AFFECTEDITEMS);

  // Add a new row to the Affected Items table
  IRow affectedItemRow = affectedItems.createRow(item);
  // Get the Redline Manufacturers table
  ITable redlineAML =
item.getTable(ItemConstants.TABLE_REDLINEMANUFACTURERS);
  // Add a manufacturer part to the table
params.put(ItemConstants.ATT_MANUFACTURERS_MFR_NAME, "AMD");
  params.put(ItemConstants.ATT_MANUFACTURERS_MFR_PART_NUMBER, "1234-
009");
  params.put(ItemConstants.ATT_MANUFACTURERS_PREFERRED_STATUS,
listvalues);
  redlineAML.createRow(params);
  // Add another manufacturer part to the table
  params.clear();
  params.put(ItemConstants.ATT_MANUFACTURERS_MFR_NAME, "DIGITAL
POWER");
  params.put(ItemConstants.ATT_MANUFACTURERS_MFR_PART_NUMBER,
"355355");
  params.put(ItemConstants.ATT_MANUFACTURERS_PREFERRED_STATUS,
listvalues);
  redlineAML.createRow(params);
}
```

To redline the Title Block table of an item:

The following is an example of redlining the Title Block table of the item. It assumes Item.Page_Two and

the attribute Text01 are visible and Text01 is change controlled.

Example: Redlining the Title Block table of an item

```
    ITable page2Tab = item.getTable(ItemConstants.TABLE_REDLINEPAGETWO);
    Iterator it = page2Tab.getTableIterator();
    IRow redPage2Row = (IRow)it.next();
    ICell cell =
    redPage2Row.getCell(CommonConstants.ATT_PAGE_TWO_TEXT01);
```

```
System.out.println("old value, before update: " +
cell.getOldValue());
redPage2Row.getCell(CommonConstants.ATT_PAGE_TWO_TEXT01).setValue("u
pda
ted Text01")
```

Removing Redline Changes

When you make redline changes to a table such as a BOM table, you may want to undo the changes for a row and restore it to its original state. You can use the `IRedlinedRow.undoRedline()` method to undo any redline changes to a row.

If you undo the redlines for a row, any cells that are modified are restored to their original values. A redlined row can also be one that was added or deleted. If you undo the redlines for a row that was added, the entire row is removed from that revision. If you undo the redlines for a row that was deleted, the entire row is restored.

Example: Removing redline changes from the BOM table

```
private static undoBOMRedlines(IItem item, String rev) throws
APIException {
  item.setRevision(rev);
  ITable redlineBOM = item.getTable(ItemConstants.TABLE_REDLINEBOM);
  Iterator it = redlineBOM.iterator();
  while (it.hasNext()) {
    IRedlinedRow row = (IRedlinedRow)it.next();
    row.undoRedline();
  }
}
```

Removing Redline Changes in Bulk Mode

Removing Redline Changes in Bulk Mode

Agile SDK enables removing (undoing) redlines with the aid of `IRedlinedTable`. This interface provides the API to perform bulk redline undos as shown below.

```
IRedlinedTable.undoRedline(Collection rows);
IRedlinedTable.undoAllRedline();
```

Note See Redlining on page 85.

Example: Typecasting Redline tables to IRedlinedTable interface

```
IItem item = (IItem) session.getObject(ItemConstants.CLASS_PART,
"PART_001");
    item.setRevision("B"); // Unreleased change
    ITable bomTable = item.getTable(ItemConstants.TABLE_REDLINEBOM);
    Iterator it = bomTable.iterator();
    List rows = new ArrayList();
    while(it.hasNext()) {
           IRow row = (IRow) it.next();
           if(((IRedlined)row).isRedlineModified())
                  rows.add(row);
}
```

```
// Only Redline tables can be typecasted to IRedlinedTable interface
// Case 1:
    ((IRedlinedTable)bomTable).undoRedline(rows);
// Case 2:
    ((IRedlinedTable)bomTable).undoAllRedline();
```

Identifying Redlined Rows and Redlined Cells

The `IRedlined` interface is designed to identify redlined rows and redlined cells. It is only supported on redlined tables. The interface works in conjunction with the `isRedlineModified()` method to show if objects are redlined. The interface typecasts `IRow` and `ICell` objects as follows:

- `IRow` indicates if the row is redline modified
- `ICell` indicates if the cell is redline modified.

Example: Identifying redlined rows and cells

```
public interface IRedlined {
public boolean isRedlineModified()
throws APIException;
}
```

The `IRedlined.isRedlineModified()` method returns a boolean value. This value is TRUE when cells or rows are redlined.

Note `IRedlined.isRedlineModified()` returns a FALSE value for all cells on redline added or redline removed rows.

Using ICell.getOldValue

With the introduction of the `IRedlined` interface, the `ICell.getOldValue()` method is no longer defined for redline added and redline removed rows. The `ICell.getOldValue()` method has a meaningful result only when `FLAG_IS_REDLINE_MODIFIED` is true for the row.

Note Do not call this method for redline added or redlined removed rows.

Chapter 5

Working with Data Cells

This chapter includes the following:

- About Data Cells 91
- Data Types 91
- Checking User's Discovery Privilege 92
- Checking if the Cell is a Read-Only Cell 93
- Getting Values 93
- Setting Values 95
- Getting and Setting List Values 96
- Using Reference Designator Cells 99

About Data Cells

An `ICell` object is a data field for an Agile PLM object that you have loaded or created in your program. A cell can correspond to a field on a tab in Agile Web Client or a single cell on a table. The `ICell` object consists of several properties that describe the current state of a cell. Most of the data manipulation your Agile API programs perform will involve changes to the value or properties of cells.

Data Types

The type of objects associated with the `getValue()` and `setValue()` methods depends on the cell's data type. Table below lists the object types of cell values for `getValue()` and `setValue()` methods.

DataTypeConstants	Object type associated with getValue and setValue
TYPE_DATE	Date
TYPE_DOUBLE	Double
TYPE_INTEGER	Integer
TYPE_MONEY	Money
TYPE_MULTILIST	IAgileList
TYPE_OBJECT	Object
TYPE_SINGLELIST	IAgileList
TYPE_STRING	String
TYPE_TABLE	Table

Note There are other Agile PLM datatypes, such as `TYPE_WORKFLOW`, but they are not used for cell values.

Checking User's Discovery Privilege

The Discovery privilege is the most basic Agile PLM privilege. It allows users that an object exists. If you do not have the Discovery privilege for an object, you won't be able to view it.

For example, if a user does not have the Discovery privilege for Manufacturer Parts, your program will not allow the user to view several cells on the Manufacturers table. You can use the `ICell.hasDiscoveryPrivilege()` method to check if the user has the Discovery privilege for a particular cell, as shown in the following example.

Note When you get the value for a cell for which you don't have the Discovery privilege, the Agile API returns a null string (""). This behavior is different with other Agile PLM Clients. For example, Agile Web Client displays the value "No Privilege" when you try to view a field for which you don't have the necessary viewing privileges.

Example: Checking Discovery privilege

```
Object v;
Integer attrID = ItemConstants.ATT_MANUFACTURERS_MFR_NAME;
try {
// Get the Manufacturers table
 ITable aml =
   item.getTable(ItemConstants.TABLE_MANUFACTURERS);
// Get the first row of the Manufacturers table
 IIterator iterator =
   aml.getTableIterator();
  if (iterator.hasNext()) {
    IRow amlRow =
   (IRow)iterator.next();
  }

// Get the value for the Mfr. Name field.
// If the user does not have Discovery privilege, the value is a null
String.
 v = amlRow.getValue(attrID);
  txtMfrName.setText(v.toString());
// If the user does not have the Discovery privilege
// for the cell, make its text color red.
 ICell cell =
   amlRow.getCell(attrID);
 if (cell.hasDiscoveryPrivilege()==false) {
   txtMfrName.setForeground(Color.red);
 }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Checking if the Cell is a Read-Only Cell

Roles and privileges assigned to a user by Agile PLM administrators, determine the level of access the user has to Agile PLM objects and their underlying data. For example, users with only ReadOnly privileges can view Agile PLM objects but not modify them.

Whenever your program displays a value from a cell, you should check whether the cell is read-only for the current user. If it is, your program must not allow the user to edit the value. If a user tries to set a value for a read-only cell, the Agile API throws an exception

Example: Checking whether a field is read-only

```
// ID for "Title Block.Description"
Integer attrID = ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION;
// Set the value for the Description text field.
try {
  txtDescription.setText(item.getValue(attrID).toString());
  // Get the ICell object for "Title Block.Description" ICell cell =
item.getCell(attrID);
  // If the cell is read-only, disable it
  if (cell.isReadOnly()) {
    txtDescription.setEnabled(false);
    txtDescription.setBackground(Color.lightGray);
  }
  else {
    txtDescription.setEnabled(true);
    txtDescription.setBackground(Color.white);
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Getting Values

The following table lists Agile API methods for getting values for cells.

Method	Description
`ICell.getValue()`	Gets a cell value
`IRow.getValue()`	Gets a cell value contained within a row
`IRow.getValues()`	Gets all cell values contained within a row
`IDataObject.getValue()`	Gets a cell value on Page One, Page Two, or Page Three

Before working with a cell's value, you must select the cell. Agile PLM cells are instances of attributes. To specify the attribute for a cell, specify either the attribute's ID constant, it's fully qualified name (such as "Title Block.Description"), or an `IAttribute` object. For more information about referencing attributes, see Referencing Attributes.

Note You can use `ICell getAPIName()` to access Data Cell attribute values. For information to use this field, see Accessing PLM Metadata with APIName Field on page 123.

The following example shows how to reference a cell by attribute ID constant.

Example: Specifying a cell by ID

```
Object v;
Integer attrID = ItemConstants.ATT_TITLE_BLOCK_NUMBER;
try {
  v = item.getValue(attrID);
} catch (APIException ex) {
  System.out.println(ex);
}
```

The following example shows how to reference a cell by fully a qualified attribute name.

Example: Specifying a field by fully qualified name

```
Object v;
String attrName = "Title Block.Number";
try {
  v = item.getValue(attrName);
} catch (APIException ex) {
  System.out.println(ex);
}
```

The method that you use to get a cell value depends on the current object in use by your program. Use the `ICell.getValue()` method if you have already retrieved an `ICell` object and want to retrieve a value.

Example: Getting a value using ICell.getValue()

```
private static Object getCellVal(ICell cell) throws APIException {
  Object v;
  v = cell.getValue();
  return v;
}
```

Quite often, your program will first retrieve an object, such as an item, and then use the `IDataObject.getValue(java.lang.Object cellId)` method to retrieve values for it.

Example: Getting a value using IDataObject.getValue(Object cellID)

```
private static Object getDescVal(IItem item) throws APIException {

  Integer attrID = ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION;
  Object v;
  v = item.getValue(attrID);
  return v;
}
```

The object returned by the `getValue()` method is of the same data type as the Agile PLM attribute. For more information about data types, see Data Types on page 91.

Note All cells in a table returned by a query contain `String` values regardless of the datatypes associated with those cells. For more information about query result tables, see Working with Query Results on page 62.

If you are iterating over rows in an Agile PLM table, you can use the `IRow.getValues()` method to retrieve a `Map` object containing all cell values for a particular row in the table. The returned `Map` object maps attribute ID keys to cell values.

Understanding SDK Date Formats and User Preferences

In SDK, date is available as a Java Date object and does not format the date according to *User Preferences*. However, end users can convert it to their preferred format in GUI's *User Preferences*.

Important End users must use the GMT date format for PPM dates. For more information, refer to the *Agile PLM Product Portfolio Management User Guide.*

Setting Values

The following table lists Agile API methods for setting values for cells.

Method	Description
`ICell.setValue()`	Sets a cell value
`IRow.setValue()`	Sets a cell value contained within a row
`IRow.setValues()`	Sets multiple cell values contained within a row
`IDataObject.setValue()`	Sets a cell value on Page One, Page Two, or Page Three
`IDataObject.setValues()`	Sets multiple cell values on Page One, Page Two, or Page Three

The method you use to set a value depends on the current object in use by your program.

Use the `ICell.setValue()` method if you've already retrieved a `ICell` object and want to set its value.

Example: Setting a value using ICell.setValue()

```
private static void setDesc(ICell cell, String text) throws
APIException {
  cell.setValue(text);
}
```

If your program has already retrieved an object, such as a part, you can use the `IDataObject.setValue()` method to set values for it.

Example: Setting a value using IDataObject.setValue()

```
private void setDesc(IItem item, String text) throws APIException {
  Integer attrID = ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION;
  item.setValue(attrID, text);
}
```

If you are iterating over rows in an Agile PLM table, you can use the `IRow.setValues()` method to set the cell values for an entire row. You can also use the `IDataObject.setValues()` method to set multiple cell values on Page One, Page Two, or Page Three of an object. The `Map` parameter you specify with `setValues()` maps attributes to cell values.

Example: Setting multiple values in a row using IRow.setValues()

```
private void setBOMRow(IRow row) throws APIException {

   Map map = new HashMap();
   map.put(ItemConstants.ATT_BOM_ITEM_NUMBER, "23-0753");
   map.put(ItemConstants.ATT_BOM_QTY, "1");
   map.put(ItemConstants.ATT_BOM_FIND_NUM, "0");

   row.setValues(map);
}
```

When you set an Agile PLM value, you must know the cell's data type. If you try to set a cell's value using an object of the wrong data type, the method fails. You may need to cast the object to another class before using it to set a value.

Note If you don't explicitly demarcate transactional boundaries in your code, every `setValue()` operation your program performs is treated as a separate transaction.

Catching Exceptions for Locked Objects

If someone else is modifying an object, it is temporarily locked by that user. If you try to set the value for a cell when another user has the object locked, your program will throw an exception. Therefore, whenever your program sets values of cells, make sure you catch the following Agile exceptions related to locked objects:

- `ExceptionConstants.APDM_ACQUIRE_DBLOCK_FAILED`
- `ExceptionConstants.APDM_RELEASE_DBLOCK_FAILED`
- `ExceptionConstants.APDM_OBJVERSION_MISMATCH`

You should also catch exception 813, which is related to locked objects.

The typical exception message that Agile PLM returns for a locked object is "Someone is working on this object. Please try again later."

For more information about how to handle exceptions, see Handling Exceptions on page 313.

Getting and Setting List Values

There are two different datatypes for list cells. One for SingleList and one for MultiList cells. When you get the value for a SingleList or MultiList cell, the object returned is an `IAgileList` object. For that reason, list cells are slightly more complicated to work with than other cells. The `IAgileList` interface provides methods for getting and setting the current list selection. This section provides examples showing how to get and set values for different types of Agile PLM lists, including cascading lists.

When you use `ICell.getAvailableValues()` to get the available values for a list cell, the returned `IAgileList` object may include obsolete list values. Your program should not permit users to set the value for a list cell to an obsolete value. For information on how to check whether a list value is obsolete, see Making List Values Obsolete on page 169.

When a list contains String values, the values are case-sensitive. This means that whenever you set the value for a list cell you must ensure that the value is the right case.

Getting and Setting Values for SingleList Cells

A SingleList cell allows you select one value from the list. When you get the value for a SingleList cell, the object returned is an `IAgileList`. From that `IAgileList` object, you can determine what the currently selected value is. The following example shows how to get and set values for the "Title Block.Part Category" cell for an item.

Example: Getting and setting the value for a SingleList cell

```
private static String getPartCatValue(IItem item) throws APIException {

  // Get the Part Category cell
  ICell cell =
item.getCell(ItemConstants.ATT_TITLE_BLOCK_PART_CATEGORY);
  // Get the current IAgileList object for Part Category IAgileList cl
= (IAgileList)cell.getValue();
  // Get the current value from the list
  String value = null;
  IAgileList[] selected = cl.getSelection();
  if (selected != null && selected.length > 0) {
    value = (selected[0].getValue()).toString();
  }
  return value;
}
private static void setPartCatValue(IItem item) throws APIException {
  // Get the Part Category cell
  ICell cell =
item.getCell(ItemConstants.ATT_TITLE_BLOCK_PART_CATEGORY);
  // Get available list values for Part Category
  IAgileList values = cell.getAvailableValues();

  // Set the value to Electrical
  values.setSelection(new Object[] { "Electrical" });
  cell.setValue(values);
}
```

Getting and Setting Values for MultiList Cells

A MultiList cell behaves very similar to a SingleList cell except that it allows you to select multiple values. A MultiList cell cannot be a cascading list. The following example shows how to get and set values for a MultiList cell, "Title Block.Product Line(s)" for an item.

Example: Getting and setting the value for a MultiList cell

```
private static String getProdLinesValue(IItem item) throws APIException
{
  String prodLines;
  // Get the Product Lines cell
  ICell cell =
item.getCell(ItemConstants.ATT_TITLE_BLOCK_PRODUCT_LINES);
  // Get the current IAgileList object for Product Lines IAgileList
list = (IAgileList)cell.getValue();
  // Convert the current value from the list to a string prodLines =
list.toString();
  return prodLines;
}

private static void setProdLinesValue(IItem item) throws APIException {
  // Get the Product Lines cell
  ICell cell =
item.getCell(ItemConstants.ATT_TITLE_BLOCK_PRODUCT_LINES);
  // Get available list values for Product Lines
  IAgileList values = cell.getAvailableValues();

  // Set the Product Lines values
  values.setSelection(new Object[] {"Saturn","Titan","Neptune"});
  cell.setValue(values);
}
```

Getting and Setting Values for Cascading Lists

A SingleList cell can be configured to be a cascading list. A cascading list presents a list in multiple hierarchical levels, letting you drill down to a specific value in the list hierarchy. For more information about working with cascading lists, see Working with Lists on page 153.

When you get the value for a cascading list cell, a vertical bar (also called a piping character) separates each level in the cascading list. To select the value for a cascading list, use the `IAgileList.setSelection()` method. You can specify either an array of `IAgileList` leaf nodes or a `String` array containing one string delimited by vertical bars. After you select the value, save it using one of the `setValue()` methods.

The following example shows how to get and set the value for a cascading list.

Example: Getting and setting the value for a cascading list

```
private String getCascadeValue(IItem item) throws APIException {

  String value = null;
  // Get the Page Two.List01 value
  IAgileList clist =
    (IAgileList)item.getValue(ItemConstants.ATT_PAGE_TWO_LIST01);
  // Convert the current value from the list to a string value =
clist.toString();
```

```
  return value;
}
private void setCascadeValue(IItem item) throws APIException {
  String value = null;
  // Get the Page Two List01 cell
  ICell cell = item.getCell(ItemConstants.ATT_PAGE_TWO_LIST01);
  // Get available list values for Page Two List01
  IAgileList values = cell.getAvailableValues();

  // Set the value to "North America|United States|San Jose"
  values.setSelection(new Object[] {"North America|United States|San
Jose"});
  cell.setValue(values);
}
```

Although the previous example shows one way to set the value for a cascading list, there's another longer form you can use that illustrates the tree structure of the list. Instead of specifying a single `String` to represent a cascading list value, you can set the selection for each level in the list. The following example selects a value for a cascading list with three levels: continent, country, and city.

Example: Setting the value for a cascading list (long form)

```
private void setCascadeValue(IItem item) throws APIException{

String value = null;
// Get the Page Two List01 cell
ICell cell = item.getCell(CommonConstants.ATT_PAGE_TWO_LIST01);
// Get available list values for Page Two List01
IAgileList values = cell.getAvailableValues();

// Set the continent to "North America"
IAgileList continent = (IAgileList)values.getChildNode("North
America");
// Set the country to "United States"
IAgileList country = (IAgileList)continent.getChildNode("United
States");
// Set the city to "San Jose"
IAgileList city = (IAgileList)country.getChildNode("San Jose");
values.setSelection(new Object[]{city});
// Set the cell value
cell.setValue(values);
}
```

Using Reference Designator Cells

You can control how to use reference designator cells with Agile 9 SDK. You can make reference designator cells render collapsed or expanded depending on your system setting. The `IReferenceDesignatorCell` interface contains three public APIs that allow the end user to retrieve reference designator information in three formats:

- Collapsed–for example A1–A3; use `getCollapsedValue()`
- Expanded–A1, A2, A3; use `getExpandedValue()`
- Array of individual reference designators–[A1, A2, A3]; use `getReferenceDesignators[]`

The following table lists Agile API methods for retrieving reference designator values for cells.

Method	Description
`IReferenceDesignatorCell.getCollapsedValue()`	Gets a collapsed representation of the reference designators. For example, “A1,A2,A3” would be represented as “A1–A3”. Note that the range separator, (“–”) is defined as part of the system preferences.
`IReferenceDesignatorCell.getExpandedValue()`	Gets an expanded value of a reference designator. For example, for “A1-A3” the string, “A1, A2, A3” would be returned.
`IReferenceDesignatorCell.getReferenceDesignators()`	Gets the individual reference designators as an array of strings. For example, for “A1-A3” an array of these three strings, [“A1”, “A2”, “A3”] would be returned.

Note In previous releases of Agile SDK, the value of a reference designator was a comma-delimited list of reference designators. Because the functionality of `cell.getValue()` for a reference designator will depend on the system setting controlling reference designator presentation, the SDK user should not use `cell.getValue()` or `row.getValue()`. We recommend that you get the cell and cast it into an I`ReferenceDesignatorCell`; then call the method that corresponds to your desired data structure for processing or display reference designator information.

Chapter 6

Working with Folders

This chapter includes the following:

- About Folders 101
- Loading a Folder 103
- Creating a Folder 103
- Setting the Folder Type 104
- Adding and Removing Folder Elements 104
- Getting Folder Elements 105
- Deleting a Folder 108

About Folders

An `IFolder` is a general purpose container used to hold `IQuery` and `IFolder` objects as well as any of the main Agile PLM objects (`IChange`, `IItem`, `IManufacturer`, `IManufacturerPart`, and `IPackage`). Folders are used to organize queries, or searches.

Note A *file folder* is different from a folder. It is a subset of a folder and has its own interface called the `IFolder`. Files in a file folder holds can be referenced from the Attachments table of other objects. For more information about file folders, see Working with Attachments and File Folders on page 173.

There are several types of Agile PLM folders:

- Private – Folders that are accessible only to the user that created them. Users can create or delete their own Private folders.
- Public – Folders that are accessible to all Agile PLM users. Only users with the GlobalSearches privilege can create, delete, and modify Public folders.
- System – Predefined folders that ship with the Agile PLM system. Most users cannot modify or delete System folders.
- My Bookmarks (or Favorites) – A predefined folder containing each user's bookmarks to Agile PLM objects. You cannot delete the My Bookmarks folder.
- Home – The predefined Agile PLM home folder. You cannot delete the Home folder.
- Personal Searches – The predefined parent folder for each user's personal searches. You cannot delete the Personal Searches folder.
- Recently Visited – A predefined folder containing links to recently visited objects. The SDK does not populate this folder. It is only populated by Client applications. If required, you specify this in your application.

Note The recently visited folder is only flushed to the database periodically. Therefore, secondary connections like process extensions with portals, or standalone SDK applications will not see the same information that the user's GUI displays.

- Report – A folder containing reports. Although you cannot use the Agile API to create, modify, or delete report folders, you can create, modify, or delete them in Agile PLM Clients.

Note `FolderConstants` also includes a constant named `TYPE_MODIFIABLE_CONTENTS`, but it is currently unused.

Each user's selection of folders may vary. However, every user has a *Home* folder. From each user's Home folder, you can construct various subfolders and browse public and private queries. To retrieve the Home folder for a user, use the `IUser.getFolder(FolderConstants.TYPE_HOME)` method.

Folders are subject to the same transactional model as other Agile API objects. If you do not set a transaction boundary for a folder, it is automatically updated as soon as you add anything to, or remove anything from the folder.

`IFolder` extends `java.util.Collection` and `ITreeNode` support all the methods that are provided by those Superinterfaces. That is, you can work with an `IFolder` object as you would any Java `Collection`. Methods of `ITreeNode` allow you to deal with the hierarchical structure of a folder by adding and removing children, getting children, and getting the parent folder.

Interface	Inherited methods
`java.util.Collection`	`add(), addAll(), clear(), contains(), containsAll(), equals(), hashCode(), isEmpty(), iterator(), remove(), removeAll(), retainAll(), size(), toArray(), toArray()`
`ITreeNode`	`addChild(), getChildNode(), getChildNodes(), getParentNode(), removeChild()`

Using Level Separation Characters in Folder and Object Names

The SDK supports level separation characters '|' and '/' when naming `ITreeNode` objects as follows:

- '|' in `IAgileList` object names
- '/' in folder names

This feature primarily affects inherited `ITreeNode` methods shown in the table above. To use these characters, it is necessary to explicitly prefix them with the backslash character ('\').

- \|
- \/

Note To use the backslash character in Java string constants defined in SDK applications, you must specify it twice ("\\").

Loading a Folder

There are two ways to load a folder:

- Use the `IAgileSession.getObject()` method to specify the full path of a folder.
- Use the `IFolder.getChild()` method to specify the relative path of a subfolder.

Folder and query names are not case-sensitive. Therefore, you can specify a folder path using upper or lower case. For example, to load the Personal Searches folder, you can specify `/Personal Searches` or `/PERSONAL SEARCHES`.

The following example shows how to load a folder by specifying the full path to the folder.

Example: Loading a folder using IAgileSession.getObject()

```
try {
  //Load the Personal Searches folder
  IFolder folder = (IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
"/Personal Searches");
} catch (APIException ex) {
  System.out.println(ex);
}
```

The following example shows how to load a folder by specifying its path relative to another folder, in this case the user's Home Folder.

Example: Loading a folder using IFolder.getChild()

```
try {
  //Get the Home Folder
  IFolder homeFolder =
m_session.getCurrentUser().getFolder(FolderConstants.TYPE_HOME);
  //Load the Personal Searches subfolder
  IFolder folder = (IFolder)homeFolder.getChild("Personal Searches");
} catch (APIException ex) {
  System.out.println(ex);
}
```

Creating a Folder

To create a folder, use the **`IAgileSession.createObject()`** method. When you create a folder, you must specify the folder's name and its parent folder. The following example shows how to create a folder named "MyTemporaryQueries" in the Personal Searches folder.

Example: Creating a new folder

```
try {
  //Get an Admin instance
  IAdmin admin = m_session.getAdminInstance();

  //Load the Personal Searches folder
```

```
  IFolder parentFolder =
(IFolder)m_session.getObject(IFolder.OBJECT_TYPE, "/Personal
Searches");
  //Create parameters for a new folder
  Map params = new HashMap();
  params.put(FolderConstants.ATT_FOLDER_NAME, "MyTemporaryQueries");
  params.put(FolderConstants.ATT_PARENT_FOLDER, parentFolder);
  //Create a new folder
  IFolder folder = (IFolder)session.createObject(IFolder.OBJECT_TYPE,
params);
} catch (APIException ex) {
  System.out.println(ex);
```

Setting the Folder Type

By default, all new folders that you create are private folders unless otherwise specified. To change a private folder to a public folder, use the `IFolder.setType()` method. You must have the GlobalSearches privilege to be able to change a private folder to a public folder.

The two folder type constants you can use to set a folder's type are `FolderConstants.TYPE_PRIVATE` and `FolderConstants.TYPE_PUBLIC`. You cannot set a folder to any other folder type.

Example: Setting the folder type

```
try {
  //Get an Admin instance
  IAdmin admin = m_session.getAdminInstance();

  //Load the My Cool Searches folder
  IFolder folder = (IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
                   "/Personal Searches/My Cool Searches");

  //Make the folder public
  folder.setFolderType(FolderConstants.TYPE_PUBLIC);

} catch (APIException ex) {
  System.out.println(ex);
}
```

Adding and Removing Folder Elements

An Agile PLM folder can contain `IFolder` objects (subfolders), `IQuery` objects, and any kind of dataobject, such as `IChange`, `IItem`, `IManufacturer`, and `IManufacturerPart` objects. Use the `ITreeNode.addChild()` method to add objects to a folder.

Adding Folder Elements

The following example shows how to add objects to a table.

Example: Adding objects to a folder

```
public void addFolderItem(IFolder folder, Object obj) {
  try {
    folder.addChild(obj);
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Removing Folder Elements

To remove a single folder element, use the `ITreeNode.removeChild()` method. To clear all folder elements, use the `java.util.Collection.clear()` method.

Example: Removing objects from a Folder

```
void removeFolderElement(IFolder folder, Object obj) {
  try {
    folder.removeChild(obj);
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
void clearFolder(IFolder folder) {
  try {
    folder.clear();
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Getting Folder Elements

All objects contained in a folder, including subfolders, can be loaded by name. To retrieve an object from a folder, use the `IFolder.getChild()` method. Remember, the object type for folder elements can vary. Depending on the object, you could be getting a subfolder, a query, or a dataobject, such as an `IItem`.

Example: Getting a folder element

```
public void getFolderChild(IFolder folder, String name) {
  try {
    IAgileObject object = folder.getChild(name);
    //If the object is a query, run it
    if (object.getType()==IQuery.OBJECT_TYPE) {
      IQuery query = (IQuery)object;
```

```
      ITable results = query.execute();

      //Add code here to do something with the query results
    }
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

The following example shows how to use the `IFolder.getChildren()` method to return an `IAgileObject` array. In this case, the code checks the object type for each object in the array and then prints the object's name.

Example: Getting folder children

```
private void browseFolder(int level, IFolder folder) throws
APIException {
  IAdmin admin = m_session.getAdminInstance();
  Collection subObjects = folder.getChildNodes();

  for (Iterator it = subObjects.iterator();it.hasNext();) {
    IAgileObject obj = (IAgileObject)it.next();
    System.out.println(indent(level * 4));

    switch (obj.getType()) {
      case IItem.OBJECT_TYPE:
        System.out.println("ITEM: " + obj.getName());
        break;

      case IFolder.OBJECT_TYPE:
        System.out.println("FOLDER: " + obj.getName());
        browseFolder(level + 1, (IFolder)obj);
        break;

      case IQuery.OBJECT_TYPE:
        System.out.println("QUERY: " + obj.getName());
        break;
      default:
        System.out.println(
        "UNKNOWN TYPE: " + obj.getType() + ":" + obj.getName());
    }
  }
}
```

```
private String indent(int level) {
  if (level <= 0) {
    return "";
  }
  char c[] = new char[level*2];
  Arrays.fill(c, ' ');
  return new String(c);
}
private String indent(int level) {
  if (level <= 0) {
    return "";
  }
  char c[] = new char[level*2];
  Arrays.fill(c, ' ');
  return new String(c);
}
```

Another way to get a folder's children is to iterate over the folder elements, moving from one end of the folder to the other. To create an iterator for an `IFolder` object, use the `java.util.Collection.iterator()` method.

Note If you need to traverse the folder contents both forward and backward, use the `IFolder.getFolderIterator()` method to return an `ITwoWayIterator` object. `ITwoWayIterator` provides `previous(), next()`, and `skip()` methods, among others.

Example: Iterating over folder elements

```
try {
  //Load the Project X folder
  IFolder folder = (IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
                   "/Personal Searches/Project X");

  //Create a folder iterator
  Iterator it = folder.iterator();

  if (it.hasNext()) {
    //Get the next folder element
    Object obj = it.next();

    //Write code here to display each folder
    //element in your program's UI
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Deleting a Folder

To delete a folder, use the `IFolder.delete()` method. You can delete folders that are empty and that are not predefined Agile PLM system folders (such as the Global Searches and My Inbox folders).

Unlike other dataobjects, folders are not “soft-deleted” the first time you delete them. When you delete a folder, it is removed permanently from the system.

Example: Deleting a folder

```
void deleteFolder(IFolder folder) throws APIException {
  folder.delete();
}
```

Chapter 7

Working with Items, BOMs, and AMLs

This chapter includes the following:

- About Items, BOMs, and Tables 109
- Working with Items 109
- Working with BOMs 112
- Working with AMLs 119

About Items, BOMs, and Tables

Working with Items

An item is an object that helps define a product. Parts and documents are examples of types of items. A part is shipped as part of a product and has costs associated with it. A part can also be an assembly. A bill of material, or BOM, lists the separate components that make up the assembly. A document generally is an internal document, drawing, or procedure that references a part.

Items are different from other Agile PLM objects because they:

- Have a revision history, with a set of data for each revision.
- Can be incorporated, or locked from future changes.
- Can have site-specific BOMs or approved manufacturers lists (AMLs).

Getting and Setting the Revision of an Item

The revision for an item is a special type of Agile PLM attribute. The revision is always paired with another value, the number of its associated change object (such as an ECO). When you load an item, it's always loaded with the latest released revision.

Unlike other attributes, the "Title Block.Rev" field (whose ID constant is `ItemConstants.ATT_TITLE_BLOCK_REV`) for an item is not directly accessible. This means that you can't retrieve or set a revision value using `getValue()` and `setValue()` methods. For example, the `revValue` variable in the following code is always a `null String`.

Example: Failing to get a revision by accessing the `Title Block.Rev` field

```
IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-02");
IAgileList listRevValue =
  (IAgileList)item.getValue(ItemConstants.ATT_TITLE_BLOCK_REV);
   String revValue = listRevValue.toString();
if (revValue==null) {
  System.out.println("Failed to get the revision.");
```

```
}
```

The correct way to get and set the revision for an item is to use methods of the `IRevisioned` interface, as shown in the following example, which loads an item and then iterates through the item's revisions.

Example: Getting and setting the revision of an item

```
try {
  // Get an item
  IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-
02");
  // Print the item's current revision
  System.out.println("current rev : " + item.getRevision());
  // Get all revisions for the item
  Map revisions = item.getRevisions();

  // Get the set view of the map
  Set set = revisions.entrySet();

  // Get an iterator for the set
  Iterator it = set.iterator();

  // Iterate through the revisions and set each revision value
  while (it.hasNext()) {
    Map.Entry entry = (Map.Entry)it.next();
    String rev = (String)entry.getValue();
    System.out.println("Setting rev : " + rev + "....");
    item.setRevision(rev);
    System.out.println("current rev : " + item.getRevision());
  }

   catch (APIException ex) {
   System.out.println(ex);
   }
}
```

The `IRevisioned.setRevision()` method accommodates several different ways to specify a revision. The `change` parameter of the `setRevision()` method can be any of the following types of objects:

- a null object to specify an Introductory revision:

  ```
  item.setRevision(null);
  ```

- an `IChange` object associated with a particular revision:

  ```
  item.setRevision(changeObject);
  a change number (a String) associated with a particular revision:
  item.setRevision("C00450");
  revision identifier (a String such as "Introductory", "A", "B", "C",
  and so on): item.setRevision("A");
  ```

- a `String` containing both a revision identifier and a change number separated by eight spaces

(“A 23450”):

```
item.setRevision("A C00450");
```

The last type of `String` object that you can specify for the `change` parameter allows you to pass the same value used in other `Rev` cells in Agile PLM tables. For example, the “BOM.Item Rev” cell, unlike “Title Block.Rev,” is directly accessible. If you get the value for the cell, it returns a `String` containing the revision identifier and a change number separated by eight spaces.

Example: Setting the revision using “BOM.Item Rev”

```
try {
  // Get an item
  IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-
02");
  // Get the BOM table
  ITable bomTable = item.getTable(ItemConstants.TABLE_BOM);
  // Get part 1543-01 in the BOM
  ITwoWayIterator it = bomTable.getTableIterator();
  while (it.hasNext()) {
    IRow row = (IRow)it.next();
    String num =
(String)row.getValue(ItemConstants.ATT_BOM_ITEM_NUMBER);
    if (num.equals("1543-01")) {
      // Get the revision for this BOM item
      // (bomRev = revID + 8 spaces + changeNumber)
      String bomRev =
(String)row.getValue(ItemConstants.ATT_BOM_ITEM_REV);

      // Load the referenced part
      IItem bomItem = (IItem)row.getReferent();

      // Set the revision
      System.out.println("Setting rev : " + bomRev + "....");
      bomItem.setRevision(bomRev);
      System.out.println("current rev : " + bomItem.getRevision());
      break;
    }
  }
} catch (APIException ex) {
   System.out.println(ex);
}
```

Note If an item has no released revisions and no pending changes, the `IRevisioned.getRevision()` method returns a null `String` and the `IRevisioned.getRevisions()` method returns an empty `Map` object.

Changing the Incorporated Status of a Revision

Each revision of an item can be incorporated. When you incorporate the revision of an item, all attachments for that revision are locked and cannot be checked out. After an item is incorporated,

you can still use Agile Web Client to view the item's attachments, but you cannot modify them unless you submit a new Change.

To incorporate or unincorporate an item, use the `IAttachmentContainer.setIncorporated()` method. Special Agile PLM privileges are required to incorporate and unincorporate Items. If a user does not have the appropriate privileges, the `setIncorporated()` method throws an exception.

Only items that have revision numbers can be incorporated. Therefore, a preliminary item that has not been released cannot be incorporated. Once an ECO is submitted for that item and a pending revision number is specified, the revision can then be incorporated. Example 7-4 shows how to change the incorporated status of an item.

Example: Changing the incorporated status of an Item

```
try {
  // Get an item
  IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-
02");
  // Incorporate the item, or unincorporate it,
  // depending on its current state
  item.setIncorporated(!item.isIncorporated());
} catch (APIException ex) {
   System.out.println(ex);
}
```

Working with BOMs

A bill of material, or BOM, shows the components that make up a product. Each item that is listed on a BOM can be a single item or an assembly of several items.

The BOM table, like other Agile PLM tables, consists of columns, or fields, of data. Each column represents an Agile PLM attribute, such as "BOM.Item Number." Each row of the BOM table represents a separate item, either a part, a document, or a user-defined subclass.

In addition to the BOM table, there is also a redline BOM, which records redline changes to a BOM. When you load a BOM table using the `DataObject.getTable()` method, make sure you specify the correct table ID constant.

BOM Table	ID Constant
Current BOM table	`ItemConstants.TABLE_BOM`
Redline BOM table	`ItemConstants.TABLE_REDLINEBOM`

For an example showing how to retrieve a BOM table, see Retrieving a Table on page 68.

Adding an Item to a BOM

Before adding an item to the BOM table, specify the manufacturing site. A BOM item is either site-specific or common to all sites. Use the `IManufacturingSiteSelectable.setManufacturingSite()` method to specify the site. To add an item to the common BOM, use `ManufacturingSiteConstants.COMMON_SITE`.

Otherwise, specify a specific site, such as the user's default site.

Note You can't add rows to a BOM if the parent item is currently set to display all sites. Before adding a row to a BOM, make sure the item's site is not set to `ManufacturingSiteConstants.ALL_SITES`. Otherwise, the API throws an exception.

Example: Adding items to a BOM

```
//Add an item to the common BOM
public void addCommonBOMItem(IItem item, String bomnumber) throws
APIException {
  HashMap map = new HashMap();
  map.put(ItemConstants.ATT_BOM_ITEM_NUMBER, bomnumber);
  item.setManufacturingSite(ManufacturingSiteConstants.COMMON_SITE);
  item.getTable(ItemConstants.TABLE_BOM).createRow(map);
}
//Add a site-specific item to the BOM using the user's default site
public void addSiteBOMItem(IItem item, String bomnumber) throws
APIException {
 HashMap map = new HashMap();
 map.put(ItemConstants.ATT_BOM_ITEM_NUMBER, bomnumber);

item.setManufacturingSite(((IAgileList)m_session.getCurrentUser().getVa
lue()

UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0].getValu
e());
  );
  item.getTable(ItemConstants.TABLE_BOM).createRow(map);
}
```

For more information about manufacturing sites, see Managing Manufacturing Sites on page 147.

Expanding a BOM

The BOM table can be viewed as a table containing multiple levels even though the API doesn't present it that way. By default, the BOM table contains only top-level items. To expand a BOM to show its hierarchy, you need to recursively load each BOM item and its subassemblies. The following example shows how to print multiple levels of a BOM.

Example: Printing multiple levels of a BOM

```
private void printBOM(IItem item, int level) throws APIException {
  ITable bom = item.getTable(ItemConstants.TABLE_BOM);
  Iterator i = bom.getReferentIterator();
  while (i.hasNext()) {
    IItem bomItem = (IItem)i.next();
    System.out.print(indent(level));
    System.out.println(bomItem.getName());
    printBOM(bomItem, level + 1);
  }
```

```
}
private String indent(int level) {
  if (level <= 0) {
    return "";
  }
  char c[] = new char[level*2];
  Arrays.fill(c, ' ');
  return new String(c);
}
```

Copying one BOM into another BOM

Quite often the BOMs of two items will be very similar. Instead of creating a BOM from scratch, it is often easier to copy a BOM from one item to another and then make slight changes. The `Collection.addAll()` method can be used to copy the contents of one table into a target table. The `addAll()` method does not set a new revision for the item.

Note If you copy a BOM from one item to another, the target item must have the same associated manufacturing sites as the source item.

Example: Copying a BOM using Collection.addAll()

```
private static void copyBOM(IItem source, IItem target) throws
APIException {
  // Get the source BOM
  ITable sourceBOM = source.getTable(ItemConstants.TABLE_BOM);
  // Get the target BOM
  ITable targetBOM = target.getTable(ItemConstants.TABLE_BOM);
  // Add all rows from the source BOM to the target BOM
  targetBOM.addAll(sourceBOM);
}
```

Another way to copy a BOM is to iterate through the rows of a source BOM and copy each row to a target BOM.

Example: Copying a BOM by iteration

```
private static void copyBOM1(IItem source, IItem target) throws
APIException {
  // Get the source BOM
  ITable sourceBOM = source.getTable(ItemConstants.TABLE_BOM);
  // Get an iterator for the source BOM
  Iterator i = sourceBOM.iterator();

  // Get the target BOM
  ITable targetBOM = target.getTable(ItemConstants.TABLE_BOM);
  // Copy each source BOM row to the target BOM
  while (i.hasNext()) {
    targetBOM.createRow(i.next());
  }
}
```

Creating BOM-Related Product Reports

The SDK provides the `IProductReport` API with constants defined in `ProductReportConstants` to prepare the following BOM-related product reports. These reports are produced in the XML format.

- BOM Explosion reports – The BOM Explosion report displays the items that are in the bill of Material(BOM) for the one or more specified assembly, up to the desired number of levels.
- BOM Comparison reports – The BOM Comparison XML reports is the result of comparing two different BOMs up to the specified number of levels.

For example, when a base BOM compared with the target BOM the comparison will show:

- d shown in BOM node – indicates only base assembly has the BOM
- a shown in BOM node – indicates only target assembly has the BOM
- u shown in BOM node – indicates both root assemblies have the same BOM
- m shown in BOM node – indicates both root assemblies have the BOM but with some differences

All first level BOMs of both base and target assembly are categorized into another node BOMs. BOM nodes under BOMs are first sorted by FindNum and then by ItemNumber

There are several use cases for these reports. For example, archiving or comparative analysis with outputs of ERP systems.

To create a product report, you must use the `IAgileSession` object. The following examples show how to use `IAgileSession` and `ProductReportConstants` to prepare BOM Explosion and BOM Comparison reports.

Example: Creating an Agile Session

```
AgileSessionFactory factory =
AgileSessionFactory.getInstance("http://agileServer/virtualPath");
Map params = new HashMap();
params.put(AgileSessionFactory.URL,
"http://agileServer/virtualPath");
params.put(AgileSessionFactory.USERNAME, "username");
params.put(AgileSessionFactory.PASSWORD, "pwd");

IAgileSession session = factory.createSession(params);
```

Example: Preparing a BOM Comparison report

```
Map param = new HashMap();
param.put(ProductReportConstants.REPORTPARAM_REPORT_TYPE,
ProductReportConstants.REPORT_BOM_COMPARISON);
param.put(ProductReportConstants.REPORTPARAM_ITEMREVSITE,
"item1;item2");
param.put(ProductReportConstants.BOMCOMP_BOM_ATTRS,
ProductReportConstants.BOM_ATT_ITEM_NUM +";" +
ProductReportConstants.BOM_ATT_FIND_NUM);
param.put(ProductReportConstants.BOMCOMP_BOMLEVEL, "4");
IProductReport report = (IProductReport)
session.createObject(IProductReport.OBJECT_TYPE, "My BOM Comparison
Report");
String xmlReport = report.execute(param);
```

If the value for `ProductReportConstants.BOMCOMP_BOM_ATTRS` is not specified, then it is assumed to be `"Find Num;Item Number;Sites"`.

Example: Preparing a BOM Explosion report

```
Map param = new HashMap();
param.put(ProductReportConstants.REPORTPARAM_REPORT_TYPE,
ProductReportConstants.REPORT_BOM_EXPLOSION);
param.put(ProductReportConstants.BOMEXP_OBJTYPE, "Document;Part;");
param.put(ProductReportConstants.REPORTPARAM_ITEMREVSITE, "MM75-
01|23450|India;");
param.put(ProductReportConstants.BOMEXP_MAXLEVEL, "5");
IProductReport report = (IProductReport)
session.createObject(IProductReport.OBJECT_TYPE, "My BOM Explosion
Report");
String xmlReport = report.execute(param);
```

In BOM Explosion reports, the value for `ProductReportConstants.REPORTPARAM_ITEMREVSITE` can be as follows:

- `<Item_number>|<Change_number>|<Site_number> where <Change_Number>` and `<Site_number>` are optional if:
  - `<Change_number>` is not specified it is assumed to be the Latest revision
  - `<Site_number>` is not specified it is assumed as Common Sites
- The value can have one or more Items delimited by semicolon
- `Item1;Item2;Item3` are the Latest revision of `Item1,Item2` and `Item3` for Common Sites
- `Item1|ECO1;Item2;Item3` (`Item1` with `ECO1` revision and latest revision of `Item2, Item3`)
- `Item1|ECO1|Site1;Item2|ECO2 (Item1` with `ECO1` revision with Site1 Specific BOM and Item2 with ECO2 revision)
- `Item1|Site1;Item2` (`Item1` with `Site1` Specific BOM and Latest revision of `Item2` with Common Sites)

In BOM Comparison reports, the value for `ProductReportConstants.REPORTPARAM_ITEMREVSITE` can be as follows:

- `<Item_number>|<Change_number>|<Site_number> where <Change_Number>` and `<Site_number>` are optional when:
  - <Change_number> is not specified, then it is assumed as Latest revision.
  - <Site_number> is not specified, it is assumed as Common Sites.
- The value must have two Items delimited by a semicolon
- `Item1;Item2` (Latest revision of `Item1` and `Item2` and all Sites)
- `Item1|ECO1;Item2` (`Item1` with `ECO1` revision and Latest revision of `Item2`)
- `Item1|ECO1|Site1;Item2|ECO2` (`Item1` with `ECO1` revision with Site1 Specific BOM and `Item2` with `ECO2` revision)
- `Item1|Site1;Item2 (Item1` with `Site1` Specific BOM and Latest revision of `Item2` with Common Sites)

Redlining a BOM

To redline a BOM table, follow these steps:

1. Get a released assembly item.
2. Create a new Change Order, such as an ECO, for the item.
3. Add the item to the Affected Items table of the ECO. Also, specify the new revision for the change and set the item's revision to the associated change.
4. Modify the item's Redline BOM table.

In the following sections, there are code examples for each of these steps.

Note You can remove redlines from a row of the BOM table. See Removing Redline Changes on page 88.

Getting a Released Assembly Item

The following example shows how to load an assembly item from the Part subclass. Make sure the Part you specify is released and has a BOM.

Example: Getting a released assembly

```
// Load a released assembly item
private static IItem loadItem(IAgileSession myServer, Integer
ITEM_NUMBER) throws APIException {
  IItem item = (IItem)myServer.getObject("Part", ITEM_NUMBER);
  if (item != null) {
    //Check if the item is released and has a BOM
    if (item.getRevision().equals("Introductory") ||
      !item.isFlagSet(ItemConstants.FLAG_HAS_BOM)) {
      System.out.println("Item must be released and have a BOM.");
      item = null;
    }
  return item;
}
```

Creating a Change Order

To redline a BOM, you must create a Change Order, such as an ECO. Example below shows how to create an ECO and select a Workflow for the selected ECO.

Example: Creating an ECO

```
private static IChange createChange(IAgileSession myServer, Integer
ECO_NUMBER)
     throws APIException {
  IChange change =
(IChange)myServer.createObject(ChangeConstants.CLASS_ECO, ECO_NUMBER);
  // Set the Workflow ID
  change.setWorkflow(change.getWorkflows()[0]);
  return change;
}
```

Adding an Item to the Affected Items tab of a Change Order

After you create an ECO, you can add the Part you loaded to the Affected Items table of the ECO. Every ECO is associated with a revision. The following example shows how to specify the new revision for the ECO, and then set the revision for the Part to the one associated with the ECO.

Example: Adding an item to the Affected Items table of a change order

```
private static void addAffectedItems(IAgileSession myServer, IItem
item, IChange change)
  throws APIException {
  // Get the Affected Items table
  ITable affectedItems =
change.getTable(ChangeConstants.TABLE_AFFECTEDITEMS);

  // Create a Map object to store parameters
  Map params = new HashMap();

  // Set the value of the item number by specifying the item object
  params.put(ChangeConstants.ATT_AFFECTED_ITEMS_ITEM_NUMBER, item);
  // Specify the revision for the change
params.put(ChangeConstants.ATT_AFFECTED_ITEMS_NEW_REV, "B");
  // Add a new row to the Affected Items table IRow affectedItemRow =
affectedItems.createRow(params);
  // Select the new revision for the part
  item.setRevision(change);
}
```

Modifying the Redline BOM Table

After the Part has been added to the Affected Items table of an ECO and a revision has been specified, you can begin to modify the Part's Redline BOM table. The following example shows how to get the Redline BOM table, add and remove rows, and set specific cell values.

Example: Modifying the Redline BOM table

```
private static void modifyRedlineBOM(IAgileSession myServer, IItem
item) throws APIException {
  // Get the Redline BOM table
  ITable redlineBOM = item.getTable(ItemConstants.TABLE_REDLINEBOM);
  // Create two new items, 1000-002 and 1000-003
  IItem item1 = (IItem) myServer.createObject(ItemConstants.CLASS_PART,
"1000-002");
  IItem item2 = (IItem) myServer.createObject(ItemConstants.CLASS_PART,
"1000-003");
  // Add item 1000-002 to the table
  IRow redlineRow = redlineBOM.createRow(item1);
  redlineRow.setValue(ItemConstants.ATT_BOM_QTY, new Integer(50));
  redlineRow.setValue(ItemConstants.ATT_BOM_FIND_NUM, new
Integer(777));
  // Add item 1000-003 to the table
  redlineRow = redlineBOM.createRow(item2);
```

```
    redlineRow.setValue(ItemConstants.ATT_BOM_QTY, new Integer(50));
    redlineRow.setValue(ItemConstants.ATT_BOM_FIND_NUM, new
 Integer(778));
    // Remove item 1000-003 from the table
    IRow     delRow;
    String   itemNumber;
    Iterator it = redlineBOM.iterator();
    while (it.hasNext()) {
      delRow     = (IRow)it.next();
      itemNumber =
 (String)delRow.getValue(ItemConstants.ATT_BOM_ITEM_NUMBER);
      if (itemNumber.equals("1000-003")) {
        redlineBOM.removeRow(delRow);
        break;
      }
    }

    // Change the Qty value for item 1000-002
    IRow modRow;
    it = redlineBOM.iterator();
    while (it.hasNext()) {
      modRow     = (IRow)it.next();
      itemNumber =
 (String)modRow.getValue(ItemConstants.ATT_BOM_ITEM_NUMBER);
      if (itemNumber.equals("1000-002")) {
        modRow.setValue(ItemConstants.ATT_BOM_QTY, new Integer(123));
      }
    }
  }
```

Working with AMLs

The Manufacturers table for an item is also called the approved manufacturers list, or AML. It lists manufacturers that have been approved to supply a particular item. The list identifies the manufacturer part for that item. The Manufacturers table consists of columns, or fields, of data. Each column represents an Agile PLM attribute, such as “Manufacturers.Mfr. Name.” Each row of the Manufacturers table references a separate manufacturer part.

In addition to the Manufacturers table, there is also a redline Manufacturers table, which records redline changes. When you load a Manufacturers table using the `DataObject.getTable()` method, make sure you specify the correct table ID constant.

BOM Table	ID Constant
Current Manufacturers table	`ItemConstants.TABLE_MANUFACTURERS`
Redline Manufacturers table	`ItemConstants.TABLE_REDLINEMANUFACTURERS`

Adding an Approved Manufacturer to the Manufacturers Table

Similar to the BOM Table, the Manufacturers Table requires that you specify the manufacturing site before adding a new row to the table. An approved manufacturer is either site-specific or common to all sites. Use the `IManufacturingSiteSelectable.setManufacturingSite()` method to specify the site. To add an approved manufacturer to the common Manufacturers table, use `ManufacturingSiteConstants.COMMON_SITE`. Otherwise, select a specific site, such as the user's default site.

Note You can't add rows to an AML if the parent item is currently set to display all sites. Before adding a row to an AML, make sure the item's site is not set to `ManufacturingSiteConstants.ALL_SITES`. Otherwise, the API throws an exception.

Example: Adding approved manufacturers to an AML

```
//Add a MfrPart to the common AML
public void addCommonApprMfr(IItem item, String mfrName, String
mfrPartNum) throws APIException {
  HashMap map = new HashMap();

map.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_PART_NU
MBER, mfrPartNum);
  map.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_NAME,
mfrName);
  IManufacturerPart mfrPart = (IManufacturerPart)m_session.getObject(
    ManufacturerPartConstants.CLASS_MANUFACTURER_PART, map
  );
  item.setManufacturingSite(ManufacturingSiteConstants.COMMON_SITE);
  item.getTable(ItemConstants.TABLE_MANUFACTURERS).createRow(mfrPart);
}
//Add a site-specific MfrPart to the AML using the user's default site
public void addSiteApprMfr(IItem item, String mfrName, String
mfrPartNum) throws APIException {
  HashMap map = new HashMap();

map.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_PART_NU
MBER, mfrPartNum);
  map.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_NAME,
mfrName);
  IManufacturerPart mfrPart = (IManufacturerPart)m_session.getObject(
    ManufacturerPartConstants.CLASS_MANUFACTURER_PART, map
  );

item.setManufacturingSite(((IAgileList)m_session.getCurrentUser().getVa
lue(
    UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0]
  );
  item.getTable(ItemConstants.TABLE_MANUFACTURERS).createRow(mfrPart);
```

}

For more information about manufacturing sites, see Managing Manufacturing Sites on page 147.

Redlining an AML

Once an item is released, you can change the Manufacturers table only by issuing a new change order. The change order allows you to redline the Manufacturers table.

Note You can remove redlines from a row of the Manufacturers table. See Removing Redline Changes on page 88.

To redline a Manufacturers table:

1. Get a released revision of an item.
2. Create a new ECO, MCO, or SCO.
   - ECOs lets you modify an item's BOM or Manufacturers tables.
   - MCOs lets you modify an item's Manufacturers table.
   - SCOs let you modify an item's site-specific BOM or Manufacturers tables.
3. Add the item to the Affected Items table of the change.
4. For ECOs, specify the new revision for the change. SCOs and MCOs do not affect an item's revision.
5. Modify the Redline Manufacturers table.

Chapter 8

Accessing PLM Metadata with APIName Field

This chapter includes the following:

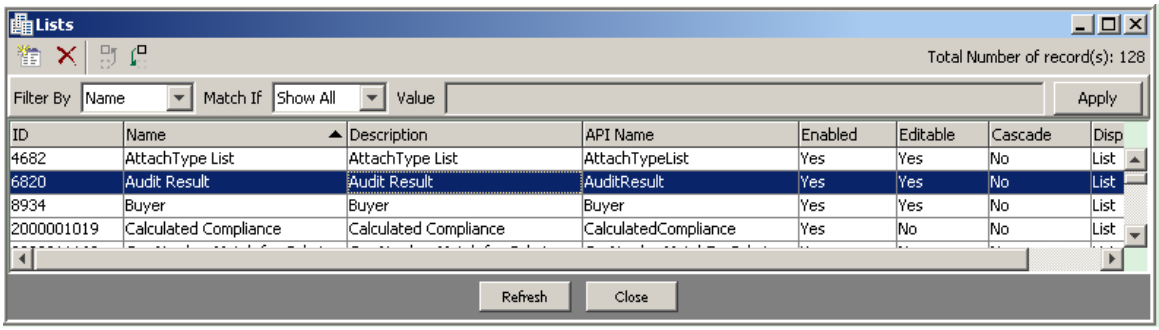
- About APIName Field 123
- Assigning Names to APIName Fields 124
- APIName Validation Rules 124
- Accessing Metadata Using the APIName Field 125

About APIName Field

The primary purpose of the APIName field is to facilitate SDK developers' access to internal Agile metadata when developing SDK applications. Prior to introduction of this field, display names or numeric IDs defined in the SDK Constants file are used to access Agile internal/metadata of objects in Classes, Tables, Attributes, Lists, and so on. The negative aspect of this approach is that numbers are difficult to remember and display names can change. Other hand, an object's APIName is unique, it is easier to remember, and unlike DisplayName it is not subject to change which can break your code.

For example, your SDK application can use "AuditResult" which is the APIName of the *List* instead of its ID which is 6820, or its display name which is "Audit Result" to look up its internal value.

Figure 7: Accessing attribute value via API Name field



The following paragraphs describe the rules that assign a name to APIName fields and SDK interfaces that you can in your SDK applications to access internal data with the APIName field.

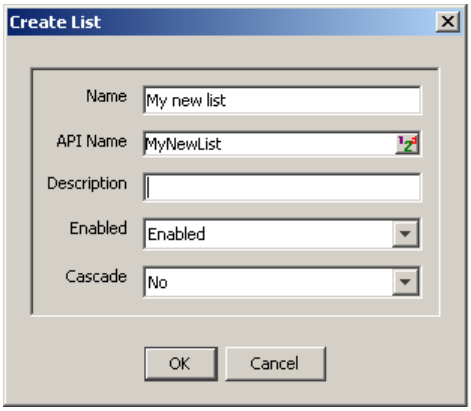
Note When upgrading to Release 9.3, it is possible to have duplicate APINames assigned to previously user defined fields. For example, if you have the user defined field "Foo" on P2 and P3, the upgrade tool will assign the APIName Foo to both fields. To avoid these duplications, change the APIName for one of these fields in the Java Client.

Assigning Names to APIName Fields

Names are automatically assigned to APIName fields by the PLM application when authorized users create new data objects in Java Client. Objects that support APIName have the additional "API Name" field in their *Create* dialog boxes which PLM immediately populates once a name is typed in the object's "Name" field.

The PLM assigned APIName converts the contents of the "Name" field using the *CamelCase* naming convention. The *CamelCase* convention is adhered to by Java JDK for all core APIs and closely resembles the name of the API. For example, the class “Manufacturer Parts” is converted to “ManufacturerParts" and the list "My new list" is converted to "MyNewList".

Figure 8: API name field



APIName Validation Rules

The APIName naming convention must adhere to the following rules:

- It can contain only characters such as Unicode or multi-byte characters are not allowed
- It must be a valid Java/XML identifier
  - Allowed characters are a-z, A-Z, 0-9, and _ (underscore)
- It must be between 1 and 255 characters long
- It is case-sensitive.
- It must be unique within a context, for example:
  - The attribute “Number” can exist in the cover page table for classes “Parts” as well as “Changes” (Different context, Parts & Changes)
  - Two attributes with APIName “Number” cannot exist in the cover page table of “Parts” (Same context, Parts cover page).
  - Two attributes with APIName "Number" cannot exist in Cover Page, Page Two and Page Three (Cover Page, Page Two and Page Three are a single context)

Accessing Metadata Using the APIName Field

You can use the API name of the Agile Metadata to:

- Access the metadata of Agile PLM (Node, Class, Attribute)
- Access/manipulate the value of the metadata (attributes and table attributes) of a data object

You can view the API name of Nodes, Classes, and Attributes in Java Client. SDK interfaces that support the APIName field are listed in the following tables.

APIs that Support the APIName Field

API	Example
IAdmin	
IAdmin	
getAgileClass(Object id)	getAgileClass("Parts")
getNode(Object id)	getNode ("Part.TitleBlock")
IAgileClass	
getAttribute (Object key)	getAttribute("TitleBlock.number") or, getAttribute("number")
getTableAttributes(Object tableId)	getTableAttributes("TitleBlock")
getTableDescriptor (Object id)	getTableDescriptor("TitleBlock")
isSubclassOf(Object cls)	isSubclassOf("Parts")
IAgileList	
getChild(Object child)	getChild("UNITED_STATES") UNITED_STATES is the APIName for entry 'United States' in 'Country' list
getChildNode(Object child)	getChildNode("UNITED_STATES")
setSelection (Object[] *childNodes*)	setselection(new Object[] {"UNITED_STATES" , "INDIA"})
IAgileSession	
createObject(Object objectType, Object params)	Map map = new HashMap();

	String partNumber = "P00001" map.put("TitleBlock.number", partNumber); IDataObject dObj = (IDataObject) (m_session.createObject("Part", map));
createObject(int objectType, Object params)	Map map = new HashMap(); String partNumber = "P00001" map.put("number", partNumber); IDataObject dObj = (IDataObject) (m_session.createObject("Part", map));
getObject(Object objectType, Object params)	Map map = new HashMap(); map.put("TitleBlock.number", "1000-01"); IDataObject dObj = (IDataObject) (m_session.getObject("Part", map));
getObject(int objectType, Object params)	Map map = new HashMap(); map.put("TitleBlock.number", "1000-01"); IDataObject dObj = (IDataObject)(m_session.getObject(IItem.OBJECT_TYPE, map));
IDataObject	
getCell(Object key)	getCell("PageTwo.list11") or getCell("list11")
getTable(Object tableId)	getTable("AffectedItems")
getValue(Object attribute)	getValue ("PageTwo.list11") or getValue ("list11")
setValue(Object key,Object value)	setValue("PageTwo.text01","test")
saveAs(Object type,Object params)	Map params = new HashMap(); params.put("number", number); IItem part2 = (IItem) part.saveAs("Document", params);

setValues(Map map)	Map map = new HashMap();
	map.put("TitleBlock.number", "1000-01");
	part.setValues(map);

ILibrary	

getAdminList(Object listId)	getAdminList("ActionStatus")

createAdminList(Map map)	map.put(IAdminList.ATT_NAME,"My List");
	map.put(IAdminList.ATT_APINAME, "MyList");
	map.put(IAdminList.ATT_DESCRIPTION, "My desc");
	map.put(IAdminList.ATT_ENABLED, new Boolean(false));
	map.put(IAdminList.ATT_CASCADED, new Boolean(false));
	IAdmin admin = m_session.getAdminInstance();
	IListLibrary listLibrary = admin.getListLibrary();
	IAdminList newList = listLibrary.createAdminList(map);

INode	
getChild(Object child)	IAdmin admin = m_session.getAdminInstance(); INode node = admin.getNode(NodeConstants.NODE_AGILE_CLASSES); INode partsClass = node.getChild("Parts");
getChildNode(Object child)	IAdmin admin = m_session.getAdminInstance(); INode node = admin.getNode("AgileClasses"); INode partsClass = node.getChildNode("Parts");
IProgram	

saveAs (Object type, Object[] tablesToCopy, Object params)	HashMap map = new HashMap();
	map.put("name", new_number);
	map.put("scheduleStartDate", new Date());
	Object[] objects = new Object[]{"PageTwo", "PageThree", "Team"};
	IProgram program2 =
	(IProgram)program.saveAs("Program",

	objects, map);

saveAs(Object type, Object[]tablesToCopy, Object params, boolean applyToChildren)	HashMap map = new HashMap();
	map.put("name", new_number);
	map.put("scheduleStartDate", new Date());
	Object[] objects = new Object[]{"PageTwo", "PageThree", "Team"};
	IProgram program2 =
	(IProgram)program.saveAs("Program", objects, map, true);

IProject	

assignSupplier(Object supplierParams)	HashMap map = new HashMap();
	map.put("Responses.itemNumber", item.getName());
	map.put("Responses.supplier", supplier.getName());
	rfq.assignSupplier(map);

IQuery	
setResultAttributes (Object[] attrs)	String[] attrs = new String[3]; attrs[0] = "TitleBlock.number"; attrs[1] = "TitleBlock.description"; attrs[2] = "TitleBlock.lifecyclePhase"; query.setResultAttributes(attrs);
IRequestForQuote	

assignSupplier (Object supplierParams)	HashMap map = new HashMap();
	map.put("Responses.itemNumber", item.getName());
	map.put("Responses.supplier", supplier.getName());
	rfq.assignSupplier(map);

ITable	
createRow(Object param)	HashMap params = new HashMap(); params.put("itemNumber", "P0001");

	params.put("newRev", "A");
	ITable affectedItems =
	change.getTable("AffectedItems");
	IRow affectedItemRow =
	affectedItems.createRow(params);

createRows(Object[] rows)	
getAvailableValues(Object attr)	getAvailableValues("PageTwo.list01")

updateRows (Map rows)	HashMap[] mapx = new HashMap[5];
	Map rows = new HashMap();
	mapx[0] = new HashMap();
	mapx[0].put("newRev", "A");
	mapx[0].put("text01", "sdk test1");
	rows.put(rowArray[0], mapx[0]);
	mapx[1] = new HashMap();
	mapx[1].put("newRev", "A");
	mapx[1].put("text01", "sdk test2");
	rows.put(rowArray[1], mapx[1]);
	tab.updateRows(rows);

ITableDesc	
getAttribute (Object key)	getAttribute("number")

SDK APIs that Get the APIName Field

Interface	Method
IAdminList	getAPIName()
IAgileClass	getAPIName()
IAgileList	getAPIName() addChild(Object child, String apiName)
IAttribute	getAPIName()
ICell	getAPIName()
INode	getAPIName()
IProperty	getAPIName()
ITableDesc	getAPIName()

API Names of Root Administrator Nodes

The following table lists the API names of the top level Administrator nodes which are not exposed in Agile Java Client. Top Level Admin Nodes are Admin Nodes that exist on their own. That is, no other Admin Node must exist in order for these Admin nodes to exist. For example, Class and Roles are top level nodes, but Life Cycle Phases and Attributes are not because they belong to another Admin Node. Similarly, Workflow Statuses are not top level nodes because they belong to Workflow.

Root Node	API Name
ACS Responses	`ACSResponses`
Account Policy	`AccountPolicy`
Activity Statuses	`ActivityStatuses`
ActivityHealths	`ActivityHealths`
Agile Classes	`AgileClasses`
Agile Workflows	`AgileWorkflows`
Agile eHubs	`AgileEHubs`
Attachment Purge Setting	`AttachmentPurgeSetting`
AutoNumbers	`AutoNumbers`
Catchers	`Catchers`
Character Set	`CharacterSet`
Cluster	`Cluster`
Company Profile	`CompanyProfile`
Criteria Library	`CriteriaLibrary`
Dashboard Management	`DashboardManagement`
Default Role Settings	`DefaultRoleSettings`
Destinations	`Destinations`
Event Handler Types	`EventHandlerTypes`
Event Handlers	`EventHandlers`
Event Subscribers	`EventSubscribers`
Event Types	`EventTypes`
Events	`Events`
Example Role/Privilege	`ExampleRolePrivilege`
Filters	`Filters`
Full Text Search Settings	`FullTextSearchSettings`
Import Preference Setting	`ImportPreferenceSetting`

LDAPConfig	LDAPConfig
LifeCycle Phases	LifeCyclePhases
My Assignments	MyAssignments
Notification Templates	NotificationTemplates
PGC SmartRules	PGCSmartRules
Package File Types	PackageFileTypes
Portals	Portals
Preferences	Preferences
Privileges	Privileges
Process eXtension Library	ProcessEXtensionLibrary
Query Cleanup	QueryCleanup
RFQ Terms and Conditions	RFQTermsAndConditions
Reports	Reports
Roles	Roles
Server Location	ServerLocation
Sign Off Message	SignOffMessage
SmartRules	SmartRules
Subscribers	Subscribers
Task Configuration	TaskConfiguration
UOM Families	UOMFamilies
Viewer and Files	ViewerAndFiles
wCM Servers	WCMServers

API Name Examples

The following example shows how to log in to an Agile PLM server, create two parts, enable Page Two text 01 and List 20, set values for them, and then add the second part to the BOM Table of the first part.

Example: Using the APIName field to access metadata

```
import com.agile.api.*;
import java.util.*;

/**
 * This sample code shows how to use the API name.
 * It uses some of the SDK APIs with the API name.
 * For a list of API names for attributes and classes,
```

```
 * refer to Agile Java Client.
 * Some API names in Agile Java Client may differ from the ones
 * in this example. This is because a duplicate conflict
 * was detected in the API name in the same context.
 * If detect this conflict, be sure change the API name
 * in this sample before compiling and executing the code.
*/

public class APIName
{
    public static final String        USERNAME = "admin";
    public static final String        PASSWORD = "agile";
    public static final String        URL      =
    "http://localhost:<>/Agile";

    public static IAgileSession    session    = null;
    public static IAdmin           admin      = null;
    public static AgileSessionFactory factory = null;
    public static IListLibrary listLibrary    = null;

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            // Create an IAgileSession instance
            session = connect(session);
            admin   = session.getAdminInstance();
            listLibrary = admin.getListLibrary();

            // Create two parts
            IItem  itemParent = createItem(getAutonumber());
            IItem  itemChild = createItem(getAutonumber());
            // enable Page Two tab for Part
            enableP2();

            // enable Page Two Text 01 and set value
            setP2Text(itemParent);

            // create a new AdminList
            createAdminList();

            // enable Page Two List 20 and set value
            setP2List(itemParent);

            // Add the child part to the BOM table of the parent part
            ITable bomTable = addBOM(itemParent, itemChild);
        }
       catch (Exception e) {
        e.printStackTrace();
        }
        finally {
            session.close();
        }
    }
```

```
/**
 * @throws APIException
 */
private static void createAdminList() throws APIException {
       Map listParams = new HashMap();
       listParams.put(IAdminList.ATT_APINAME, "MY_LIST"); // Specify
       the API name of the List
       listParams.put(IAdminList.ATT_NAME, "My List");
       listParams.put(IAdminList.ATT_ENABLED, new Boolean(true));
       IAdminList myList = listLibrary.createAdminList(listParams);
       IAgileList values = myList.getValues();
       values.addChild("Value A", "VAL_A"); // Specify the API name
       along with the value
       values.addChild("Value B", "VAL_B");
       values.addChild("Value C", "VAL_C");

       myList.setValues(values);
       System.out.println("Created Admin List " + myList.getName());
}

/**
 * @throws APIException
 */
private static void enableP2()  throws APIException {
       INode p2 = admin.getNode("Part.PageTwo"); // Fully qualified
       API name
       IProperty visible =
       p2.getProperty(PropertyConstants.PROP_VISIBLE);
       IAgileList values = visible.getAvailableValues();
       values.setSelection(new Object[]{ "Yes" });
       visible.setValue(values);
       System.out.println("Page two enabled for Part class");
}

/**
 * @param itemParent
 * @throws APIException
 */
private static void setP2Text(IItem itemParent) throws APIException
{
       IAgileClass clazz = itemParent.getAgileClass();
       ITableDesc p2TableDesc = clazz.getTableDescriptor("PageTwo");
       // 'PageTwo' is the API name of the Page Two tab
       IAttribute text01 = p2TableDesc.getAttribute("text01"); //
       'text01' is the API name of the Text01 field
       IProperty visible =
       text01.getProperty(PropertyConstants.PROP_VISIBLE);
       IAgileList values = visible.getAvailableValues();
       values.setSelection(new Object[]{ "Yes" });
       visible.setValue(values);
       itemParent.setValue("PageTwo.text01", "SDK test"); //
       'PageTwo.text01' is the fully qualified APIName for
       ItemConstants.ATT_PAGE_TWO_TEXT01
       System.out.println("Set P2 Text01 " +
       itemParent.getValue("PageTwo.text01") + " for Part " +
       itemParent.getName());
}

/**
```

```
 * @param itemParent
 * @throws APIException
 */
private static void setP2List(IItem itemParent) throws APIException
{
      IAgileClass clazz = itemParent.getAgileClass();
      ITableDesc p2TableDesc = clazz.getTableDescriptor("PageTwo");
      // 'PageTwo' is the API name of the Page Two tab
      IAttribute text01 = p2TableDesc.getAttribute("list20");  //
      'list20' is the API name of the List20 field
      IProperty visible =
      text01.getProperty(PropertyConstants.PROP_VISIBLE);
      IAgileList values = visible.getAvailableValues();
      values.setSelection(new Object[]{ "Yes" });
      visible.setValue(values);
      IAdminList myList = listLibrary.getAdminList("MY_LIST"); //
      MY_LIST is the API name of the List 'My List'
      IAgileList listValues = myList.getValues();
      listValues.setSelection(new Object[] { "VAL_B" } ); // VAL_B
      is the API name of the list value 'Value B'
      itemParent.setValue("PageTwo.list20", listValues); //
      'PageTwo.list20' is the fully qualified APIName for
      ItemConstants.ATT_PAGE_TWO_TEXT01
      System.out.println("Set P2 List20 " +
      listValues.getSelection()[0].getValue() + " for Part " +
      itemParent.getName());
}

/**
 * <p> Create an IAgileSession instance </p>
 * @param session
 * @return IAgileSession
 * @throws APIException
 */
private static IAgileSession connect(IAgileSession session)
           throws APIException {
      factory = AgileSessionFactory.getInstance(URL);
      HashMap params = new HashMap();
      params.put(AgileSessionFactory.USERNAME, USERNAME);
      params.put(AgileSessionFactory.PASSWORD, PASSWORD); session =
      factory.createSession(params);
      return session;
}

/**
 * <p> Create a part </p>
 * @param parent
 * @return IItem
 * @throws APIException
 */
private static IItem createItem(String number) throws APIException {
      HashMap map = new HashMap();
      map.put("TitleBlock.number", number);// 'number' or
      'TitleBlock.number' is the APIName for
      ItemConstants.ATT_TITLE_BLOCK_NUMBER
      map.put("description", "test");            // 'description'
      or 'TitleBlock.description' is the APIName for
      ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION
      String p = "P" + System.currentTimeMillis();
```

```
        IItem item = (IItem)session.createObject("Part",map); //
        'Part' is the API name of the Part class
        System.out.println("Created Part " + number);
        return item;
    }

    /**
     * <p> Add the child parts to the BOM table of the parent part </p>
     * @param itemParent
     * @param itemChild1
     * @param itemChild2
     * @return ITable
     * @throws APIException
     */
    private static ITable addBOM(IItem itemParent, IItem itemChild)
    throws APIException {
        ITable table  = itemParent.getTable("BOM");
        // 'BOM' is APIName for ItemConstants.TABLE_BOM
        IRow row1 = table.createRow();
        String number =
        (String)itemChild.getValue("TitleBlock.number");
        row1.setValue("itemNumber", number);
        // 'itemNumber' is APIName for 'Item Number' on BOM Table
        System.out.println("Added Part " + itemChild.getName() + " to
        BOM of the Part " + itemParent.getName());
        return table;
    }

    /**
     * @return
     * @throws APIException
     */
    private static String getAutonumber() throws APIException{
            IAgileClass cls = admin.getAgileClass("Part");
        // 'Part' is the API name of the Part class
        IAutoNumber auto[] = cls.getAutoNumberSources();
        String number = null;
        if(auto != null && auto.length > 0)
            number = auto[0].getNextNumber();
        else
            number = "PART" + System.currentTimeMillis();
        return number;
    }
}
```

Chapter 9

Subscribing to Agile PLM Objects

This chapter includes the following:

- About User Subscriptions 137
- Getting Subscriptions for an Object 139
- Modifying the Subscriptions for an Object 141
- Making Attributes Available for Subscription 142
- Working with Subscription Tables 143

About User Subscriptions

When you load an Agile PLM business object, such as an item or change, you can then subscribe to that object. Once you subscribe to the object, you will receive a Notification whenever a triggering event occurs for that object. You can specify which events trigger a Notification. Subscription events can be a lifecycle change, a change to attachment files, or a change to the value of any cell that is made available for subscription.

You can subscribe to both routable and nonroutable objects. The Agile API provides an interface called `ISubscribable`, which enables retrieving and modifying all subscriptions for an object. All objects that a user has subscribed to are listed on the user's Subscription table.

Subscription Events

Subscription events vary per object class. The full set of events you can subscribe to are listed in the following table.

Subscription Event	SubscriptionConstants
Status Change (for routable objects)	`EVENT_STATUS_CHANGE`
Lifecycle Phase Change (for nonroutable objects)	`EVENT_LIFECYCLE_CHANGE`
Field Change	`EVENT_FIELD_CHANGE`
Add File	`EVENT_ADD_FILE`
Delete File	`EVENT_DELETE_FILE`
Checkin File	`EVENT_CHECKIN_FILE`
Checkout File	`EVENT_CHECKOUT_FILE`
Cancel Checkout File	`EVENT_CANCELCHECKOUT_FILE`

Note There are additional subscription events for Projects Execution objects that are not supported by the Agile API.

Although most routable and nonroutable objects support the subscription events listed in the table above, there are some exceptions:

- User objects do not support the Lifecycle Change subscription event.
- File Folder objects do not support the Add File and Cancel Checkout File subscription events.

The Field Change subscription event is related to any attribute whose Available To Subscribe property has been set to “Yes.” Consequently, each class and subclass can have a different set of subscribable attributes.

Subscribe Privilege

To subscribe to an object, you must have the Subscribe privilege for that class. Many predefined Agile PLM roles, such as Creator, already have the Subscribe privilege for several object classes. To change your roles and privileges, see the administrator of your Agile PLM system.

Subscription Notifications

Subscription events trigger two types of Agile PLM Notifications:

- Email – Email Notifications are sent only if the user's Receive Email Notification preference is set to Yes. For information on user and system preferences, refer to *Agile PLM Administrator Guide*.
- Inbox – Inbox Notifications occur automatically regardless of user preferences

A user with Administrator privileges can create and configure these Notifications in Java Client which provides two very similar dialogs for this purpose. The reason for the two dialogs is due to the fact that there are two sets of Email and Inbox Notifications:

- Those that the "To" field is grayed out
- Those that the administrator user can select recipients who are notified when the subscription event is triggered

Sending Notifications with SDK

Notifications are briefly described in the SDK Guide under Event Notifications. SDK exposes the following API to send Notifications to designated notifiers with the specified template for Agile PLM objects. For information about Notifications, refer to *Agile PLM Administrator Guide*.

`sendNotification`	`(IDataObject`	*`object`*	Object the Notification is issued for
	`String`	*`template`*	Name of Notification template
	`Collection`	*`notifiers`*	List of notifiers
	`boolean`	*`urgent`*	True for urgent
	`String`	*`comments`*	Comments about the Notification
	`)`		`throws APIException`

These parameters are defined as follows:

- **`object`** – object on which Notification is to be issued
- **`template`** – name of the Notification template
- **`notifiers`** – a collection containing list of users as individual `IDataObjects` such as `IUser` and `IUserGroup`
- **`urgent`** – value of true indicates send urgently, set to false otherwise
- **`comments`** – comments about the Notification

For more information about these parameters and the API, refer to Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in SDK_samples (ZIP file). To access this file, see the Note in Client-Side Components on page 2.

The following example uses `sendNotification` to send a Notification from ECO C0001 to user1 and user2 with Notification Comment and template

Example: Sending a Notification with Agile SDK

```
IAgileSession session =
    AgileSessionFactory.createSessionEx(loginParams);
List notifyList =
    new ArrayList();
IDataObject user =
    (IDataObject)session.getObject(com.agile.api.IUser.OBJECT_TYPE,"John
    Doe");
notifyList.add(user);
user =
    (IDataObject)session.getObject(com.agile.api.IUser.OBJECT_TYPE,"Jane
    Doe");
notifyList.add(user);
IDataObject agileObject =
    (IDataObject)session.getObject(com.agile.api.IChange.OBJECT_TYPE,"C0
    001");
boolean urgent = true;
String comment =
    "Add ECO approver, Notify CA";
String template =
    "Automated SDK process added ECO approver";
session.sendNotification(obj,templateName,notifyList,urgent,comment);
```

Deleting Subscribed Objects

You can delete any Agile PLM business object using the `IDataObject.delete()` method. However, you can't delete an object until its subscriptions are removed. Users can remove their own subscriptions, but not the subscriptions of other users.

Getting Subscriptions for an Object

To retrieve the current subscriptions for an object, use `ISubscribable.getSubscriptions()`, which returns an array of all `ISubscription` objects, both enabled and disabled. The following example shows how to get subscriptions for an object.

Example: Getting subscriptions for an object

```
public void getSubscriptionStatus(IAgileObject obj) throws APIException
{
    ISubscription[] subs =
    ((ISubscribable)obj).getSubscriptions();
  for (int i = 0; i < subs.length; ++i) {
   if (subs[i].getId().equals(SubscriptionConstants.EVENT_ADD_FILE)) {
      if (subs[i].isEnabled()) {
   System.out.println("Add File subscription is enabled");
      }
    }
   else if
   (subs[i].getId().equals(SubscriptionConstants.EVENT_CANCELCHECKOUT_F
   ILE)) {
      if (subs[i].isEnabled()) {
   System.out.println("Cancel Checkout File subscription is enabled");
      }
    }
   else if
   (subs[i].getId().equals(SubscriptionConstants.EVENT_CHECKIN_FILE)) {
     if (subs[i].isEnabled()) {
   System.out.println("Checkin File subscription is enabled");
      }
    }
   else if
   (subs[i].getId().equals(SubscriptionConstants.EVENT_CHECKOUT_FILE))
   {
   if (subs[i].isEnabled()) {
    System.out.println("Checkout File subscription is enabled");
      }
    }
   else if
   (subs[i].getId().equals(SubscriptionConstants.EVENT_DELETE_FILE)) {
      if (subs[i].isEnabled()) {
        System.out.println("Delete File subscription is enabled");
      }
    }
   else if
   (subs[i].getId().equals(SubscriptionConstants.EVENT_FIELD_CHANGE)) {
      if (subs[i].isEnabled()) {
        IAttribute attr = subs[i].getAttribute();
        if (attr != null) {
        String attrName = attr.getFullName();
           System.out.println("Field Change subscription
```

```
            is enabled for " + attrName);
          }
        }
      }
else if
(subs[i].getId().equals(SubscriptionConstants.EVENT_LIFECYCLE_CHANGE))
{
    if (subs[i].isEnabled())
    System.out.println("Lifecycle Change subscription is enabled");
    }
    else if
    (subs[i].getId().equals(SubscriptionConstants.EVENT_STATUS_CHANGE))
    {
       if (subs[i].isEnabled())
      System.out.println
        ("Status Change subscription is enabled");
     }
  else
    System.out.println("Unrecognized subscription event: " +
    subs[i].getId());
  }
}
```

Modifying the Subscriptions for an Object

You can use the Agile API to modify subscriptions for the current user only. If you change your subscriptions for a particular business object, other users' subscriptions for that object remain unaffected.

The list of subscription events for any object is set at the server and cannot be modified by the Agile API. However, you can select the fields (attributes) you want subscribed. If you have Administrator privileges, you can also modify classes to define which fields are available for subscription. For more information, see the next section.

To work with a subscription, use the following `ISubscription` methods:

- `enable(boolean)` – Enables or disables the subscription
- `getAttribute()` – Returns the `IAttribute` object associated with a subscription. Only Field Change subscriptions have associated attributes
- `isEnabled()` – Returns true if the subscription is enabled, false otherwise
- `getId()` – Returns the subscription ID, which is equivalent to one of the `SubscriptionConstants`

`ISubscription` is a value object interface. Consequently, when you make changes to a subscription (for example, by enabling it), it's not changed in the Agile PLM system until you call `ISubscribable.modifySubscriptions()`.

The following example shows how to enable the Lifecycle Change and Field Change subscription events and subscribe to two Page Two fields. All other subscription events are disabled.

Example: Enabling and disabling subscriptions for an object

```
public void setSubscriptions(IAgileObject obj) throws APIException {
    ISubscription[] subs = ((ISubscribable)obj).getSubscriptions();
   for (int i = 0; i < subs.length; ++i) {
     // Enable the Status Change subscription event
    if (subs[i].getId().equals(SubscriptionConstants.EVENT_STATUS_CHANGE)) {
       subs[i].enable(true);
     }
     // Enable the Field Change subscription event for Page Two.Text01
and Page Two.List01
    else if
    (subs[i].getId().equals(SubscriptionConstants.EVENT_FIELD_CHANGE)){
      if (subs[i].getAttribute() != null)
    System.out.println(subs[i].getAttribute().getFullName() + ": " +
    subs[i].getAttribute().getId());
       if ((subs[i].getAttribute() != null) &&
    ((subs[i].getAttribute().getId().equals(CommonConstants.ATT_PAGE_TWO
    _LIST01)) ||
    (subs[i].getAttribute().getId().equals(CommonConstants.ATT_PAGE_TWO_
    TEXT01))) )
          subs[i].enable(true);
       else
          subs[i].enable(false);
     }
    // Disable all other subscription events
    else
      subs[i].enable(false);
   }
   ((ISubscribable)obj).modifySubscriptions(subs);
}
```

Making Attributes Available for Subscription

The attributes that are subscribable vary per Agile PLM class. In general, most Page One (Title Page, Cover Page, and General Info) attributes are subscribable and can therefore be made available for subscription. All Page Two attributes, except for `ATT_PAGE_TWO_CREATE_USER`, and all Page Three attributes are also subscribable.

When an attribute's Available To Subscribe property is set to Yes, users can subscribe to the attribute. When you call `ISubscribable.getSubscriptions()` for an object, the returned `ISubscription[]` array includes an `ISubscription` object for each subscription event. Although there is only one Field Change event—whose constant is `SubscriptionConstants.EVENT_FIELD_CHANGE`—each subscribed attribute is treated as a separate event that can trigger a subscription Notification. Depending on how your Agile PLM system has been configured, there could be dozens of attributes available for subscription for a particular object.

If an attribute isn't visible, it also isn't subscribable even if its Available To Subscribe property has been set to Yes. Therefore, before setting the Available To Subscribe property to Yes, make sure the Visible property is also set to Yes. The following example shows how to make all Page Two attributes for ECOs available for subscription.

Example: Making Page Two attributes available for subscription

```
try {
  // Get the ECO subclass
   IAgileClass classECO = m_admin.getAgileClass("ECO");
  // Get Page Two attributes
   IAttribute[] attr =
   classECO.getTableAttributes(ChangeConstants.TABLE_PAGETWO);
  // Make all visible Page Two attributes subscribable
  for (int i = 0; i < attr.length; ++i) {
    IProperty prop = null;
    IAgileList listValues = null;
    String strVal = "";

  // Check if the attribute is visible
   prop = attr[i].getProperty(PropertyConstants.PROP_VISIBLE);
   listValues = (IAgileList)prop.getValue();
   strVal = listValues.toString();

    // If the attribute is visible, make it subscribable
    if (strVal.equals("Yes")) {
      prop =
attr[i].getProperty(PropertyConstants.PROP_AVAILABLE_FOR_SUBSCRIBE);
      if (prop != null) {
        listValues = prop.getAvailableValues();
        listValues.setSelection(new Object[] { "Yes" });
        prop.setValue(listValues);
      }
    }
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Parent and Child Attributes

Several read-only attributes have a child relationship with a parent attribute. Child attributes derive values from their parent attribute. Consequently, parent attributes are available for subscription, but child attributes are not. Examples of child attributes include BOM table attributes like `BOM.Item List02` and `BOM.Item Text01`.

Working with Subscription Tables

A user's Subscription table lists all subscriptions the user has made. The Subscription table offers limited editing capabilities. For example, you can't add new rows to the table; the only way to add subscriptions using the Agile API is to call `ISubscribable.modifySubscriptions()` for a dataobject. However, you can remove subscriptions from the table.

The following example shows how to retrieve the Subscription table for the current user. It also shows how to remove a subscription for a part with the number 1000-02.

Example: Removing a subscription

```
try {
  // Get the current user
   IUser user = m_session.getCurrentUser();
  // Get the Subscription table
   ITable tblSubscriptions =
   user.getTable(UserConstants.TABLE_SUBSCRIPTION);
   Iterator i = tblSubscriptions.iterator();
  // Stop subscribing to part 1000-02
 while (i.hasNext()) {
   IRow row = (IRow)i.next();
   String n =
   (String)row.getValue(UserConstants.ATT_SUBSCRIPTION_NUMBER);
   if (n.equals("1000-02")) {
     tblSubscriptions.removeRow(row);
     break;
    }
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

In addition to removing individual rows from the Subscription table, you can also use the `Collection.clear()` method to clear the table.

Example: Clearing the Subscription table

```
public void clearSubscriptionTable(IUser user) throws APIException {
  // Get the Subscription table
   ITable tblSubscriptions =
   user.getTable(UserConstants.TABLE_SUBSCRIPTION);
  // Clear the table
  tblSubscriptions.clear();
}
```

The Subscription table doesn't list the events you've subscribed to for each object. To find that information, you need to open each referenced object. The following example shows how to use **`ITable.getReferentIterator()`** to iterate through the referenced objects in the table.

Example: Getting objects referenced in the Subscription table

```
try {
  // Get the current user
   IUser user = m_session.getCurrentUser();
```

```
  // Get the Subscription table
   ITable tblSubscriptions =
   user.getTable(UserConstants.TABLE_SUBSCRIPTION);
   Iterator i = tblSubscriptions.getReferentIterator();
  // Get each object referenced in the table
  while (i.hasNext()) {
   IAgileObject obj = (IAgileObject)i.next();
    if (obj instanceof ISubscribable) {
     ISubscription[] subscriptions =
     ((ISubscribable)obj).getSubscriptions();
      for (int j = 0; j < subscriptions.length; j++) {
       ISubscription subscription =
           subscriptions[j];
       System.out.println(subscription.getName());
        // Add code here to handle each subscription
      }
      System.out.println(obj.getName());
    }
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Chapter 10

Managing Manufacturing Sites

This chapter includes the following:

- About Manufacturing Sites 147
- Controlling Access to Sites 147
- Creating a Manufacturing Site 148
- Loading a Manufacturing Site 148
- Retrieving the Sites Table for an Item 149
- Adding a Manufacturing Site to the Sites Table 149
- Selecting the Current Manufacturing Site for an Item 150
- Disabling a Site 152

About Manufacturing Sites

Companies that practice distributed manufacturing use several different manufacturing sites for their products. Agile PLM site objects allow companies to maintain site-specific information for a product's parts. For example, the various manufacturing locations may have different effectivity dates for new revisions, different manufacturing instructions due to location, or different manufacturers from whom they buy components, due to location.

Changes can affect all manufacturing sites of an item or a specific site. The Affected Items table for a change lets you select the manufacturing sites that are affected. Items may have different effectivity dates and dispositions at each site. You specify effectivity dates and dispositions on the Affected Items tab of an ECO or SCO. To create a new revision when you assign the new effectivity date or disposition, use an ECO. To assign site-specific effectivity dates and dispositions without incrementing the revision, use an SCO.

For a more detailed overview of Agile PLM's manufacturing sites functionality, refer to the Agile PLM *Product Collaboration Guide*.

Controlling Access to Sites

In order to use manufacturing sites in your organization, the Sites module must be enabled in Agile Administrator. Your organization's agreement with Oracle determines the modules that are enabled in Agile PLM.

A user's access to manufacturing sites is controlled by his/her assigned roles and privileges and the Sites property in the user's profile. Your organization can create an unlimited number of manufacturing sites, however a user will not have access to every site unless all sites are specified in his user profile Sites property. Your organization may have implemented the Agile PLM system in such a way that users can access only the information pertaining to certain sites, as determined by their user profile Sites property.

To create a site-specific BOM for an item, the item's subclass must have the Site-specific BOM

property set to Allow. Otherwise, items of that subclass have BOMs that are common to all sites. For information on Sites and enabling sites, refer to the *Agile PLM Administrator Guide*.

Creating a Manufacturing Site

Manufacturing sites are identified uniquely by name. To create a manufacturing site, use the `IAgileSession.createObject` method, specifying both the class and the site name.

All users cannot create manufacturing sites. Only users who have the Create privilege for manufacturing site objects can create manufacturing sites.

Note When you create a manufacturing site, its Lifecycle Phase is set to `Disabled` by default. To use the site, make sure you enable it.

Example: Creating and enabling a manufacturing site

```
try {
  // Create a manufacturing site
  HashMap params = new HashMap();
  params.put(ManufacturingSiteConstants.ATT_GENERAL_INFO_NAME,
"Taipei");
  IManufacturingSite mfrSite =
(IManufacturingSite)m_session.createObject(
                              ManufacturingSiteConstants.CLASS_SITE,
params);
  // Enable the manufacturing site
  ICell cell = mfrSite.getCell(
    ManufacturingSiteConstants.ATT_GENERAL_INFO_LIFECYCLE_PHASE);
  IAgileList values = cell.getAvailableValues();
  values.setSelection(new Object[] { "Enabled" });
  cell.setValue(values);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Loading a Manufacturing Site

To load an `IManufacturingSite` object, use one of the `IAgileSession.getObject()` methods. The following example shows three different ways to specify the object type for a manufacturing site.

Example: Loading a manufacturing site

```
try {
  // Load the Hong Kong site
  IManufacturingSite siteHK =
    (IManufacturingSite)m_session.getObject(ManufacturingSiteConstants.C
    LASS_SITE, "Hong Kong");
  // Load the Taipei site
```

```
    IManufacturingSite siteTaipei =
      (IManufacturingSite)m_session.getObject(IManufacturingSite.OBJECT_TY
      PE, "Taipei");
    // Load the San Francisco site
      IManufacturingSite siteSF =
      (IManufacturingSite)m_session.getObject("Site", "San Francisco");
  } catch (APIException ex) {
    System.out.println(ex);
  }
```

Retrieving the Sites Table for an Item

Each item has a Sites table that lists the manufacturing sites where that item can be used. To retrieve the Sites table for an item, use the `DataObject.getTable()` method.

Example: Retrieving the Sites table

```
//Get the Sites table
private static void getSites(IItem item) throws APIException {
  IRow row;
  ITable table = item.getTable(ItemConstants.TABLE_SITES);
  ITwoWayIterator it = table.getTableIterator();
  while (it.hasNext()) {
    row = (IRow)it.next();
    //Add code here to do something with the Sites table
  }
}
```

To determine the manufacturing sites associated with an item, use the `IManufacturingSiteSelectable.getManufacturingSites()` method. Of course, you can also iterate over the Sites table to get the same information, but using the `getManufacturingSites()` method is easier and faster. See Selecting the Current Manufacturing Site for an Item on page 150 for an example that uses `getManufacturingSites()`.

Adding a Manufacturing Site to the Sites Table

Each row of the Sites table references a different `IManufacturingSite` object. To add a manufacturing site to the Sites table, use the `ITable.createRow()` method.

If a manufacturing site is not listed on an item's Sites table, then that item cannot be included in a parent item's BOM specific to that manufacturing site. For example, to add item P1001 to another item's Taipei-specific BOM, P1001 must have the Taipei site listed on its Sites table.

Example: Adding a row to the Sites table

```
private static void addSite(String itemNumber, IManufacturingSite site)
  throws APIException {
    //Load the item
    IItem item = (IItem)session.getObject(IItem.OBJECT_TYPE,
itemNumber);
```

```
    //Get the Sites table
    ITable table = item.getTable(ItemConstants.TABLE_SITES);
    //Add the manufacturing site to the table
    IRow row = table.createRow(site);
}
```

Selecting the Current Manufacturing Site for an Item

BOM and Manufacturers tables (or AMLs) can be different for each manufacturing site used for an assembly. When you retrieve a BOM or Manufacturers table for an item, you can display information for all sites or for a specific site. If you choose a specific site, only that site's information is included in the table.

The `IManufacturingSiteSelectable` interface provides methods for getting and setting the manufacturing site for an item. To get the current manufacturing site selected for an item, use the `IManufacturingSiteSelectable.getManufacturingSite()` method.

Example: Getting the currently selected manufacturing site for an item

```
private static IManufacturingSite getCurrentSite(IItem item)
  throws APIException {
    IManufacturingSite site = item.getManufacturingSite();
    return site;
}
```

The `IManufacturingSiteSelectable.getManufacturingSites()` method retrieves all available manufacturing sites that have been added to an item's Sites table.

Example: Getting all manufacturing sites associated with an item

```
private static void getItemSites(IItem item)
  throws APIException {
    IManufacturingSite[] sites = item.getManufacturingSites();
    //Print the name of each site
    for (int i = 0; i < sites.length; ++i) {
      String siteName = (String)sites[i].getValue(
        ManufacturingSiteConstants.ATT_GENERAL_INFO_NAME
        );
      System.out.println(siteName);
    }
}
```

The `IManufacturingSiteSelectable.setManufacturingSite()` method sets the current manufacturing site for an item. You can specify that an item has a specific manufacturing site, is not site-specific, or uses All Sites. To specify that an item is not site-specific, use `ManufacturingSiteConstants.COMMON_SITE`. To specify All Sites, pass the `ManufacturingSiteConstants.ALL_SITES` value.

When you set the manufacturing site for an item, the item is updated to reflect site-specific information. Consequently, your program should update the BOM and Manufacturers tables by iterating over the rows again to refresh them.

Example: Setting the current manufacturing site for an item

```
try {
  // Load sites
  IManufacturingSite siteSF =
(IManufacturingSite)m_session.getObject("Site", "San Francisco");
  IManufacturingSite siteHK =
(IManufacturingSite)m_session.getObject("Site", "Hong Kong");
  // Load an item
  IItem item  = (IItem)m_session.getObject("Part", "1000-02");
  // Set the Hong Kong site
  item.setManufacturingSite(siteHK);
  String desc =
(String)item.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  System.out.println("Hong Kong description = " + desc);
  // Set the San Francisco site
  item.setManufacturingSite(siteSF);
  desc =
(String)item.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  System.out.println("San Francisco description = " + desc);
  // Set the item to use all sites
  item.setManufacturingSite(ManufacturingSiteConstants.ALL_SITES);
  desc =
(String)item.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  System.out.println("All Sites description = " + desc);
  // Set the item to be common site (the item is not site-specific)
  item.setManufacturingSite(ManufacturingSiteConstants.COMMON_SITE);
  desc =
(String)item.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  System.out.println("Global description = " + desc);
  // Set the item to use the user's default site

item.setManufacturingSite(((IAgileList)m_session.getCurrentUser().getVa
lue(

UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0].getValu
e());
  desc =
(String)item.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  System.out.println("User's Default Site description = " + desc);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Disabling a Site

A manufacturing site can have one of two lifecycle phases, enabled or disabled. If a site is disabled, it can no longer be used to create site-specific BOMs, AMLs, and changes.

To disable a manufacturing site, set the value for the Lifecycle Phase attribute to `Disabled`.

Example: Disabling a manufacturing site

```
private static void disableSite(IManufacturingSite site)
  throws APIException {
  // Get the Lifecycle Phase cell
  ICell cell = site.getCell(
    ManufacturingSiteConstants.ATT_GENERAL_INFO_LIFECYCLE_PHASE
  );

  // Get available list values for Lifecycle Phase
  IAgileList values = cell.getAvailableValues();

  // Set the value to Disabled
  values.setSelection(new Object[] { "Disabled" });
  cell.setValue(values);
}
```

Chapter 11

Working with Lists

This chapter includes the following:

- About Lists 153
- Selecting a List Value 157
- Selecting a List from the List Library 161
- Creating Custom Lists 163
- Checking the Data Type of a List 167
- Modifying a List 168
- Printing Contents of IAgileList Objects 172

About Lists

Many attributes in the Agile PLM system are configured as lists. Agile provides two datatypes to support list fields:

- SingleList – a list in which only one value can be selected.
- MultiList – a list in which multiple values can be selected.

Attributes, properties, and cells can all be lists. The Agile API provides methods for working with lists in the `IAgileList` interface, a generalized data structure used for all Agile lists. Because `IAgileList` represents a tree structure of available list values, it extends the `ITreeNode` interface.

You can use `ITreeNode.addChild()` to add values to a list. All list values must be unique. After adding a list value, you can prevent its selection by making it obsolete.

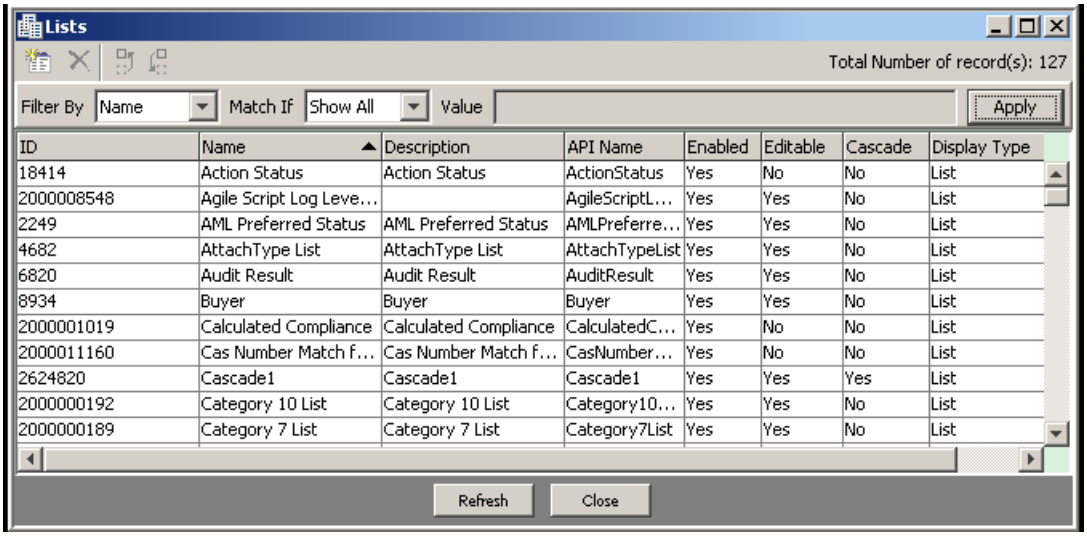
List Library

In Agile Java Client, administrators can define custom lists that can be used for Page Two and Page Three list attributes. You can also use the Agile API to define custom lists. The `IListLibrary` interface provides functionality equivalent to the list library in Agile Java Client. You can use the `IAdminList` interface to modify the values or properties of a list.

To retrieve the list library, use the `IAdmin.getListLibrary()` method. You can then use the `IListLibrary` interface to create new custom lists and work with existing lists. `AdminListConstants` provide IDs for each list in the list library.

Note The Agile API provides support for several internal Agile lists that are not exposed in the list library in Agile Java Client.

Figure 9: List Library

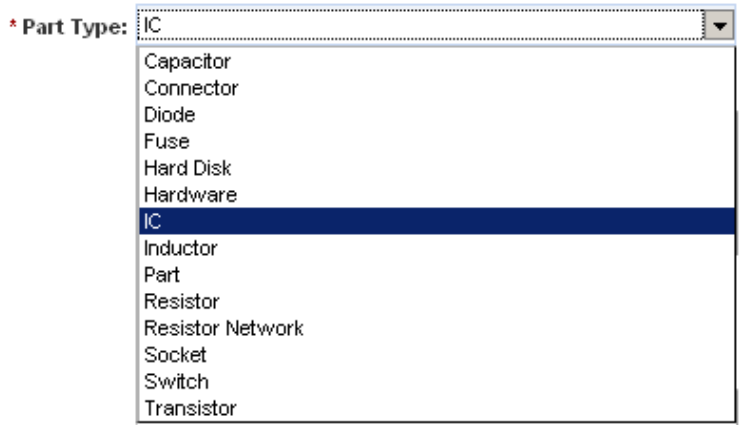


ID	Name	Description	API Name	Enabled	Editable	Cascade	Display Type
18414	Action Status	Action Status	ActionStatus	Yes	No	No	List
2000008548	Agile Script Log Leve...		AgileScriptL...	Yes	Yes	No	List
2249	AML Preferred Status	AML Preferred Status	AMLPreferre...	Yes	Yes	No	List
4682	AttachType List	AttachType List	AttachTypeList	Yes	Yes	No	List
6820	Audit Result	Audit Result	AuditResult	Yes	Yes	No	List
8934	Buyer	Buyer	Buyer	Yes	Yes	No	List
2000001019	Calculated Compliance	Calculated Compliance	CalculatedC...	Yes	No	No	List
2000011160	Cas Number Match f...	Cas Number Match f...	CasNumber...	Yes	No	No	List
2624820	Cascade1	Cascade1	Cascade1	Yes	Yes	Yes	List
2000000192	Category 10 List	Category 10 List	Category10...	Yes	Yes	No	List
2000000189	Category 7 List	Category 7 List	Category7List	Yes	Yes	No	List

SingleList Lists

A SingleList attribute or cell presents a list from which only one value can be selected. The following figure shows a SingleList cell for Part Types in Agile Web Client.

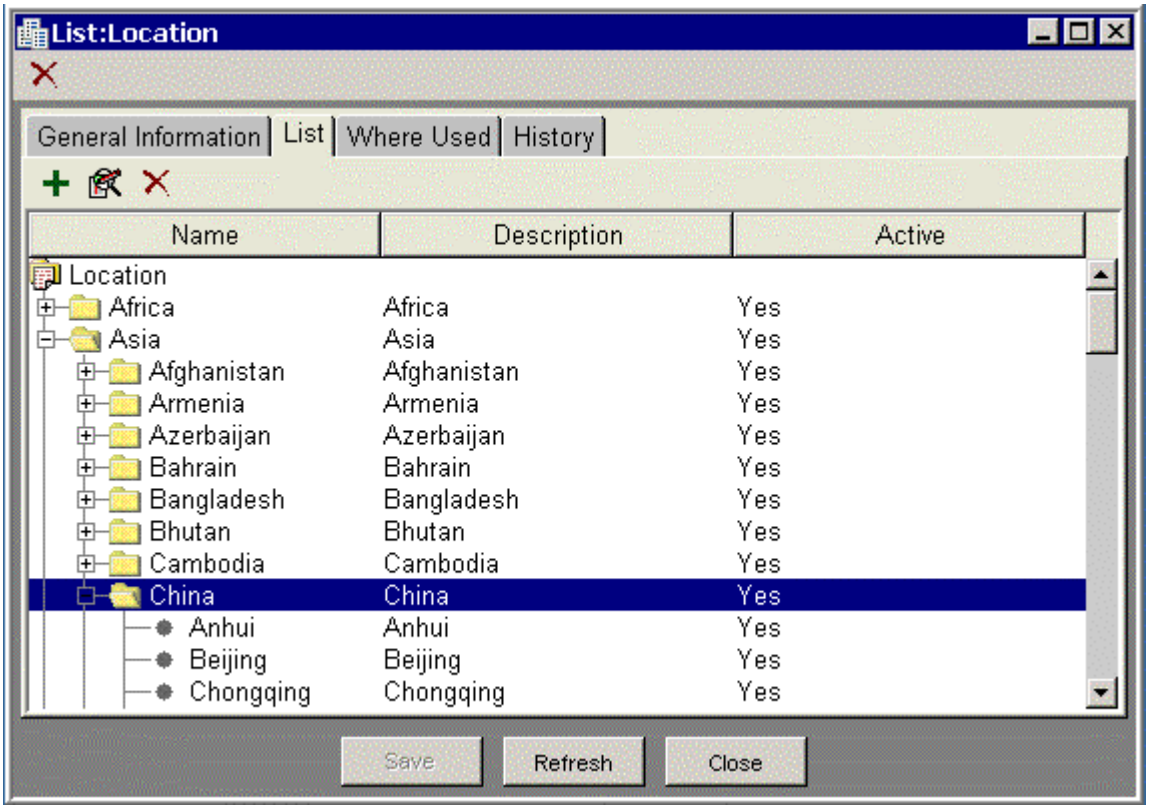
Figure 10: SingleList cell in the Agile Web Client



Cascading Lists

In Agile Java Client, you can configure a SingleList attribute to have multiple hierarchical levels. A list with multiple hierarchical levels is called a cascading list. The following figure shows the Location list, a cascading list, being configured in Agile Java Client. The list has separate levels for continent, country, and city.

Figure 11: Configuring a cascading list in the Agile Java Client

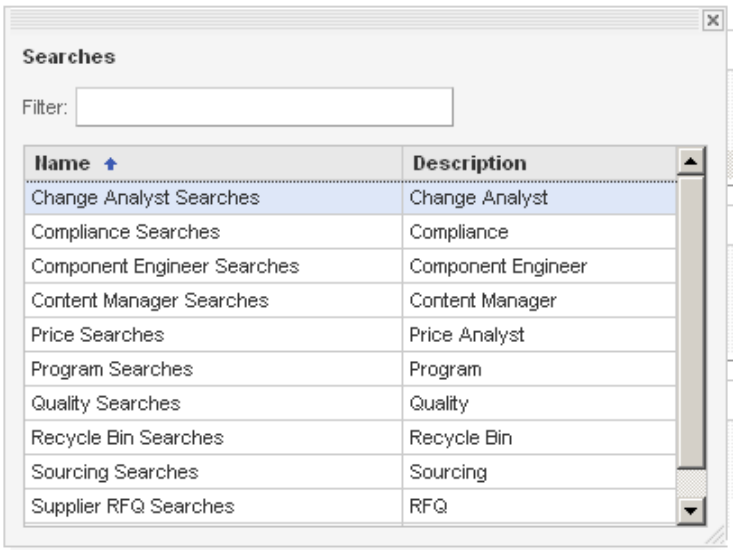


Note The Location list is the only cascading list that ships with Agile PLM. However, you can define your own cascading lists.

MultiList Lists

A MultiList attribute or cell presents a list from which multiple values can be selected. In Agile Web Client, you can select values for a MultiList cell using the Multiple Value Selection dialog, shown in the following figure.

Figure 12: Multiple Value Selection window in the Agile Web Client



Methods that Use IAgileList

The `IAgileList` interface provides the necessary methods to get and set the selected value(s) of a list. The `IAgileList` interface represents a value object with a tree structure, which is why the interface extends `ITreeNode`.

The following Agile API methods return an `IAgileList` object (or an array of `IAgileList` objects):

- `IAdminList.getValues()`
- `IAdminList.setValues(IAgileList)`
- `IAttribute.getAvailableValues()`
- `IAttribute.setAvailableValues(IAgileList)`
- `IAgileList.getSelection()`
- `ICell.getAvailableValues()`
- `IListLibrary.createAdminList(java.util.Map)`
- `IListLibrary.getAdminList(java.lang.Object)`
- `IListLibrary.getAdminLists()`
- `IProperty.getAvailableValues()`

The following methods either return an `IAgileList` or require an `IAgileList` parameter when the related attribute, cell, or property is a list (the datatype is `SingleList` or `MultiList`):

- `ICell.getValue()` – For SingleList and MultiList cells, the returned `Object` is an `IAgileList`.
- `ICell.setValue(java.lang.Object value)` – For SingleList and MultiList cells, `value` is an `IAgileList`.
- `IProperty.getValue()` – For SingleList and MultiList properties, the returned `Object` is an `IAgileList`.
- `IProperty.setValue(java.lang.Object value)` – For SingleList and MultiList properties, `value` is an `IAgileList`.
- `IRow.getValue(java.lang.Object cellId)` – For SingleList and MultiList cells, the returned `Object` is an `IAgileList`.
- `IRow.getValues()` – For each SingleList or MultiList cell in the row, the returned `Map` object contains an `IAgileList`.
- `IRow.setValue(java.lang.Object cellId, java.lang.Object value)` – If `cellID` specifies a SingleList or MultiList cell, value is an `IAgileList`.
- `IRow.setValues(java.util.Map map)` – For each SingleList or MultiList cell in the row, `map` contains an `IAgileList`.

Selecting a List Value

To select a list value, whether it is a SingleList or MultiList list, you must first get the available values for the list. You can then set the selected value. After selecting the list value, save the selection by setting the value for the cell or property.

The following example shows how to change the value of the Visible property of an attribute. The Visible property is a SingleList property with two possible values, `No` and `Yes` (or 0 and 1).

Note You can use `IAgileList getAPIName()` to get the available values of a list. For information, see Accessing PLM Metadata with APIName Field on page 123.

Example: Changing the Visible property of an attribute

```
try {
  // Get the Admin instance
IAdmin admin = m_session.getAdminInstance();
  // Get part  sub-class
IAgileClass partClass = admin.getAgileClass(ItemConstants.CLASS_PART);
  // Get the "Page Two.List03" attribute
IAttribute attr =
partClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST03);
  // Get the Visible property
IProperty propVisible =
attr.getProperty(PropertyConstants.PROP_VISIBLE);
```

```
  // Get all available values for the Visible property IAgileList
values = propVisible.getAvailableValues();
  // Set the selected list value to "Yes"
values.setSelection(new Object[] { "Yes" });
  // Instead of setting the selection to "Yes", you could also //
specify the corresponding list value ID, as in the following line:
// values.setSelection(new Object[] { new Integer(1)});
  // Set the value of the property
propVisible.setValue(values);
} catch (APIException ex) {
  System.out.println(ex);
}
```

When you use the `IAgileList.setSelection()` method, you can specify `String[]`, `Integer[]`, or `IAgileList[]` values for the `childNodes` parameter. When you select a value from the `IAgileList` object, you can use its `String` representation or its `Integer` ID.

To get the currently selected value for a list, use the `IAgileList.getSelection()` method. For a SingleList cell or property, `getSelection()` returns an array containing one `IAgileList` object. For a MultiList cell or property, `getSelection()` returns an array containing one or more `IAgileList` objects.

The following example demonstrates how to use several `IAgileList` methods, including `getSelection()`.

Example: Getting the current list value for the Visible property

```
try {
    // Get the Admin instance
IAdmin admin = m_session.getAdminInstance();
    // Get the Parts class
IAgileClass partClass =
admin.getAgileClass(ItemConstants.CLASS_PARTS_CLASS);
    // Get the "Page Two.List03" attribute
IAttribute attr =
partClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST03);
    // Get the Visible property
IProperty propVisible =
attr.getProperty(PropertyConstants.PROP_VISIBLE);
// Get the current value of the Visible property IAgileList value =
(IAgileList)propVisible.getValue();
    // Print the current value
System.out.println(value); // Prints "Yes"
  // Print the list value ID
  System.out.println(value.getSelection()[0].getId()); // Prints 1
  // Print the list value
  System.out.println(value.getSelection()[0].getValue()); // Prints
"Yes"
} catch (APIException ex) {
  System.out.println(ex);
}
```

Lists can be reused for several attributes, even for attributes of different classes. The following example reuses the list of available values for a Page Two attribute to set the list of available values for a Page Three list attribute.

Example: Reusing list values for different attributes

```
try {
  // Get the Admin instance
  IAdmin admin = m_session.getAdminInstance();

  // Get the Parts class
  IAgileClass partClass =
admin.getAgileClass(ItemConstants.CLASS_PARTS_CLASS);

  // Get the "Page Two.List01" attribute
  IAttribute attr1 =
partClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST01);
  // Get the "Page Three.List01" attribute
  IAttribute attr2 =
partClass.getAttribute(ItemConstants.ATT_PAGE_THREE_LIST01);
  // Set the available values for the list, using values from "Page
Two.List01"
  attr2.setAvailableValues(attr1.getAvailableValues());
} catch (APIException ex) {
  System.out.println(ex);
}
```

Working with Lifecycle Phase Cells

The Lifecycle Phase attribute is a SingleList datatype. Each subclass in the Agile PLM system can be defined with different lifecycle phases. Therefore, you must get a Lifecycle Phase cell for a subclass before you can retrieve the available values for its list. If you use `IAttribute.getAvailableValues()` to retrieve the available values for a Lifecycle Phase attribute instead of a subclass-specific cell, the method returns an empty `IAgileList` object. The following example highlights how to work with Lifecycle Phase cells.

Example: Working with Lifecycle Phase cells

```
private static void setLifecyclePhase(IItem item) throws APIException {
  // Get the Lifecycle Phase cell
ICell cell =
item.getCell(ItemConstants.ATT_TITLE_BLOCK_LIFECYCLE_PHASE);
  // Get available list values for Lifecycle Phase IAgileList values =
cell.getAvailableValues();
  // Set the value to the second phase
values.setSelection(new Object[] { new Integer(1)});
  cell.setValue(values);
}
```

Working with Dynamic Lists

The Agile server has both static lists and dynamic lists. Static lists contain a selection of values that do not change at run time. Dynamic lists contain a selection of values that are updated at run time. Users with administrator privileges can modify static lists and add new values and make current values obsolete. Dynamic lists cannot be modified; consequently, the Editable property of dynamic lists is set to No.

Several dynamic lists are capable of containing thousands of value objects. Items, Changes, and Users lists are examples of such lists. Although you can use these lists for Page Two and Page Three fields, you can not enumerate values for these lists.

Enumerable and Non-Enumerable Lists

As such, Agile SDK object lists are either enumerable, or non-enumerable. If a specific list is enumerable, you can read the contents of that list. If it is non-enumerable, you cannot access the list directly. For non-enumerable lists, query the Agile class that the object list uses to get the objects that are referenced by the list. The enumeration property for an object is hard coded on the server and cannot be changed.

To determine if the values for a dynamic list can be enumerated, use `IAgileList.getChildNodes()` as shown in the following example. If `getChildNodes()` returns null, the list values cannot be enumerated. However, this does not prevent you from selecting a value for the list.

Example: Checking whether values for a dynamic list are enumerable

```
private void setPageTwoListValue(IItem item) throws APIException {
  // Get the "Page Two.List01" cell
ICell cell = item.getCell(CommonConstants.ATT_PAGE_TWO_LIST01);
  // Get available values for the list IAgileList values =
cell.getAvailableValues();
  // If the list cannot be enumerated, set the selection to the current
user
if (values.getChildNodes() == null) {
   values.setSelection(new Object[]{m_session.getCurrentUser()});
   cell.setValue(values);
  }
}
private void setPageTwoMultilistValue(IItem item) throws APIException {
  // Get the "Page Two.Multilist01" cell
ICell cell = item.getCell(CommonConstants.ATT_PAGE_TWO_MULTILIST01);
  // Get available values for the list IAgileList values =
cell.getAvailableValues();
  // If the list cannot be enumerated, set the selection to an array of
users
if (values.getChildNodes() == null) {
  IAgileClass cls = cell.getAttribute().getListAgileClass();
if (cls != null) {
  IUser user1 = (IUser)m_session.getObject(cls, "hhawkes");
```

```
   IUser user2 = (IUser)m_session.getObject(cls, "ahitchcock");
   IUser user3 = (IUser)m_session.getObject(cls, "jhuston");
   Object[] users = new Object[] {user1, user2, user3};
   values.setSelection(users);
   cell.setValue(values);
     }
   }
 }
```

Non-Enumerable PG&C Lists

The following PG&C lists that were enumerable in earlier releases of the SDK, are no longer enumerable in this release.

- Declarations
- Substances
- Specifications
- Part Families
- Part Families Commodities

Selecting a List from the List Library

The `IListLibrary` interface enables working with the library of Agile lists. You can load an existing list, or create a new one. To load an existing list, use `IListLibrary.getAdminList()`. You can specify the string name of a list, such as “Disposition.” You can also specify a list, by its ID, or by an `AdminListConstants`, such as `LIST_DISPOSITION_SELECTION`. Before you attempt to use a list from the list library, make sure the list is enabled.

The following example shows how to configure a Page Two list attribute to use a list called Users.

Example: Configuring an attribute to use an Agile list

```
try {
 IAgileList values = null;
  // Get the Admin instance
IAdmin admin = m_session.getAdminInstance();
  // Get the List Library
IListLibrary listLib = admin.getListLibrary();
  // Get the Parts class
IAgileClass partClass =
admin.getAgileClass(ItemConstants.CLASS_PARTS_CLASS);
  // Get the "Page Two.List01" attribute
IAttribute attr =
partClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST01);
  // Make the list visible
IProperty propVisible =
attr.getProperty(PropertyConstants.PROP_VISIBLE);
values = propVisible.getAvailableValues();
```

```
values.setSelection(new Object[] { "Yes" });
propVisible.setValue(values);

  // Change the name of the attribute to "Project Manager"
IProperty propName = attr.getProperty(PropertyConstants.PROP_NAME);
propName.setValue("Project Manager");
  // Get the list property
  IProperty propList = attr.getProperty(PropertyConstants.PROP_LIST);
  // Use the Users list from the list library.
  IAdminList users =
listLib.getAdminList(AdminListConstants.LIST_USER_OBJECTS);
  if (users != null ) {
    if (users.isEnabled()) {
      propList.setValue(users);
    } else {
      System.out.println("Users list is not enabled.");
    }
  }

  // Specify the Default Value to the current user
  IProperty propDefValue =
attr.getProperty(PropertyConstants.PROP_DEFAULTVALUE);
  values = propDefValue.getAvailableValues();
  values.setSelection(new Object[]{m_session.getCurrentUser()});
  propDefValue.setValue(values);
} catch (APIException ex) {
  System.out.println(ex);
}
```

When you select a user-defined list using `IListLibrary.getAdminList()`, you can specify the list by name or ID. All list names must be unique. The following example shows how to select an Agile list called Colors.

Example: Selecting a list named Colors

```
private void selectColorsList(IAttribute attr, IListLibrary
m_listLibrary) throws APIException {
  // Get the List property
  IProperty propList = attr.getProperty(PropertyConstants.PROP_LIST);
  // Use the Colors list
  IAdminList listColors = m_listLibrary.getAdminList("Colors");
  if (listColors != null ) {
    if (listColors.isEnabled()) {
      propList.setValue(listColors);
    } else {
      System.out.println("Colors list is not enabled.");
    }
  }
```

```
}
```

Creating Custom Lists

The Agile API lets you modify list attributes for different classes and configure custom list attributes for Page Two and Page Three. You can customize these list attributes to create simple lists or multilists. You can also configure a list to be cascading, that is, have multiple levels.

In Agile Java Client, administrators can configure a library of custom lists by choosing Admin > Data Settings > Lists. In the Agile API, the `IListLibrary` interface provides functionality equivalent to Admin > Data Settings > Lists. The `IAdminList` interface provides functionality for configuring and customizing each list.

Creating a Simple List

To create a new list, use the `IListLibrary.createAdminList()` method, which takes a `map` parameter. The `map` that you pass with `createAdminList()` must contain values for the following `IAdminList` fields:

- `ATT_NAME` – the String name of the list. This is a required field. The list name must be unique.
- `ATT_DESCRIPTION` – the String description of the list. This is an optional field; the default value is an empty string.
- `ATT_ENABLED` – a Boolean value specifying whether the list is enabled. This is an optional field; the default value is false.
- `ATT_CASCADED` – a Boolean value specifying whether the list contains multiple levels. This is an optional field; the default value is `false`. The `ATT_CASCADED` value cannot be changed after the list is created.

Once the list is created, you can use the `IAdminList` interface to enable or disable the list and set values for it.

The following example shows how to create a new list called Colors. This list is a simple list with only one level.

Example: Creating a simple list

```
try {
  // Get the Admin instance
IAdmin admin = m_session.getAdminInstance();
  // Get the List Library
IListLibrary listLib = admin.getListLibrary();
  // Create a new Admin list
HashMap map = new HashMap();
 String name = "Colors";
 map.put(IAdminList.ATT_NAME, name);
 map.put(IAdminList.ATT_DESCRIPTION, name);
 map.put(IAdminList.ATT_ENABLED, new Boolean(true));
 map.put(IAdminList.ATT_CASCADED, new Boolean(false));
 IAdminList listColors = listLib.createAdminList(map);
```

```
   // Add values to the list
  IAgileList list = listColors.getValues(); //The list is empty at this
 point.
  list.addChild("Black");
  list.addChild("Blue");
  list.addChild("Green");
  list.addChild("Purple");
  list.addChild("Red");
  list.addChild("White");
  listColors.setValues(list);
 } catch (APIException ex) {
  System.out.println(ex);
 }
```

Lists that contain String values are case-sensitive. This means that a list can contain uppercase, lowercase, and mixed-case variations of the same value, which may not be desirable. For example, the following code snippet adds three variations of each color value to the Colors list.

Example: Adding case-sensitive values to a list

```
  IAgileList list = listColors.getValues(); //The list is empty at this
 point.
  list.addChild("Black");
  list.addChild("BLACK");
  list.addChild("black");
  list.addChild("Blue");
  list.addChild("BLUE");
  list.addChild("blue");
  list.addChild("Green");
  list.addChild("GREEN");
  list.addChild("green");
  list.addChild("Purple");
  list.addChild("PURPLE");
  list.addChild("purple");
  list.addChild("Red");
  list.addChild("RED");
  list.addChild("red");
  list.addChild("White");
  list.addChild("WHITE");
  list.addChild("white");
```

Automatically Creating New Lists by Modifying Existing Lists

Each list attribute must reference an Agile list for its values. If you retrieve an Agile list and modify its values without saving the list and then use those values for a list attribute, the Agile API automatically creates a new list. In the following example, the Colors list is retrieved, but before it is used to populate the values for a list field a new value, “Violet,” is added to the list. When `IAttribute.setAvailableValues()` is called, a new list is created.

Note Lists that are created automatically by the Agile API have a prefix “SDK” followed by a random number. You can rename such lists, if you prefer.

Example: Creating a new list automatically by modifying an existing list

```
try {
  // Get the Colors list
  IAdminList listColors = m_listLibrary.getAdminList("Colors");
  // Get the Parts class
  IAgileClass partsClass =
admin.getAgileClass(ItemConstants.CLASS_PARTS_CLASS);

  // Get the "Page Two.List01" attribute
  IAttribute attr =
partsClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST01);
  // Get the color values
  IAgileList values = listColors.getValues();

  // Add a new color
  values.addChild("Violet");

  // Set the available list values for "Page Two.List01". Because the
list
  // was modified, a new AdminList is created automatically.
  attr.setAvailableValues(values);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Creating a Cascading List

A cascading list is a list with multiple levels. You can configure SingleList attributes and cells using a cascading list instead of a simple list

Note Once you set a list to be cascading, you can’t change it to a flat list. You cannot change the value of `IAdminList.ATT_CASCADED` after the list is created.

The following example shows how to create a new cascading list called “Field Office.” The list has two levels.

Important When setting level names for cascading lists, always start with the index 0 for the first level and increment the index subsequent levels as shown in the following two examples below.

Example: Creating a cascading list

```
try {
  // Get the Admin instance
  IAdmin admin = m_session.getAdminInstance();
```

```
  // Get the List Library
  IListLibrary listLib = admin.getListLibrary();

  // Create a new Admin list
  HashMap map = new HashMap();
  String name = "Field Office";
  map.put(IAdminList.ATT_NAME, name);
  map.put(IAdminList.ATT_DESCRIPTION, name);
  map.put(IAdminList.ATT_ENABLED, new Boolean(true));
  map.put(IAdminList.ATT_CASCADED, new Boolean(true));
  IAdminList listFO = listLib.createAdminList(map);

  // Get the empty list
  IAgileList list = listFO.getValues();

  // Add the list of countries
  IAgileList india = (IAgileList)list.addChild("India");
  IAgileList china = (IAgileList)list.addChild("China");
  IAgileList usa = (IAgileList)list.addChild("USA");
  IAgileList australia = (IAgileList)list.addChild("Australia");

  // Add the list of cities
  india.addChild("Bangalore");
  china.addChild("Hong Kong");
  china.addChild("Shanghai");
  china.addChild("Suzhou");
  usa.addChild("San Jose");
  usa.addChild("Milpitas");
  usa.addChild("Seattle");
  usa.addChild("Jersey City");
  australia.addChild("Sidney");

  // Save the list values
  listFO.setValues(list);

// Set level names starting with index 0 for level 1.
  list.setLevelName(0, "Field Office Country");
  list.setLevelName(1, "Field Office City");

} catch (APIException ex) {
  System.out.println(ex);
}
```

In cascading lists, level names used by the list must be unique and you cannot share them between lists. The level names are stored internally, but Agile Java Client and Web Client currently don't display them. The level names are needed only if you want to show them in a cascading list UI that you created.

After you call the `IAdminList.setValues()` method, a valid ID is assigned to each list value. Only leaf nodes, that is, nodes on the lowest level of a cascading list, have valid IDs. In the previous example, the city nodes are leaf nodes. All other nodes have a null ID. You can use the ID to set the selection of the `IAgileList` object.

You can add a list value and its parent nodes in one statement instead of adding the parent node and then its subnodes. Use the | character to separate nodes, which represent levels, in the string. The following example replaces a portion of the code in the example; it shows how to add the same list values as in the following example, but using fewer lines of code.

Example: Adding parent nodes and subnodes to a cascading list

```
// Get the list values
IAgileList list = listFO.getValues(); // The list is empty at this
point.
  // Add nodes
  list.addChild("India|Bangalore");
  list.addChild("Hong Kong|Hong Kong");
  list.addChild("China|Suzhou");
  list.addChild("USA|San Jose");
  list.addChild("USA|Milpitas");
  list.addChild("USA|Jersey City");
  list.addChild("Australia|Sidney");

// Save the list values
listFO.setValues(list);

  // Set level names
  list.setLevelName(0, "Field Office Country");
  list.setLevelName(1, "Field Office City");
```

Checking the Data Type of a List

A list can contain objects of any Agile datatype. Therefore, before getting or setting a list value, you should determine the data type of objects in the list. If you are working with a cascading list, the data type can vary with each level. There are several ways to determine the data type of a list:

- For predefined lists in the List Library, use `IAdminList.getListDataType()` to get the data type.
- For SingleList and MultiList attributes that have only one list level, use the `IAttribute.getListDataType()` method to get the data type for the entire list.
- For a level within a cascading list, use the `IAgileList.getLevelType()` method to get the data type for a particular level.

Example: Checking the data type of a list

```
public void setDefaultValue() throws APIException {
  // Get the Parts class
  IAgileClass partClass =
m_admin.getAgileClass(ItemConstants.CLASS_PARTS_CLASS);
  // Get the "Page Two.List01" attribute
  IAttribute attr =
partClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST01);
  switch (attr.getListDataType()) {
    case DataTypeConstants.TYPE_OBJECT:
      //Add code here to handle Object values
      break;

    case DataTypeConstants.TYPE_STRING:
      //Add code here to handle String values
      break;
    default:
      //Add code here to handle other datatypes
  }
}
```

Modifying a List

Once a list has been created, you can modify it in the following ways:

- Add values to a list
- Make list values obsolete
- Set the list name and description
- Set level names for a cascading list
- Enable or disable a list
- Delete a list
- Modify or remove values added to a list

Adding a Value to a List

The following example shows how to add several values to a list. Before adding a value to a list, use the `ITreeNode.getChildNode()` method to make sure the value doesn't already exist.

Example: Adding values to a list

```
private static void updateProductLinesList() throws APIException {
  // Get the Admin instance
  IAdmin admin = m_session.getAdminInstance();

  // Get the List Library
  IListLibrary listLib = admin.getListLibrary();
```

```
  // Get the Product Lines list
  IAdminList listProdLine = listLib.getAdminList("Product Line");
  // Add values to the list
  IAgileList listValues = listProdLine.getValues();
  addToList(listValues, "Saturn");
  addToList(listValues, "Titan");
  addToList(listValues, "Neptune");
  listProdLine.setValues(listValues);
}
private static void addToList(IAgileList list, String value) throws
APIException {
  if (list.getChildNode(value) == null) {
    list.addChild(value);
  }
}
```

Making List Values Obsolete

You can prevent the selection of a list value by making the list entry obsolete. However, when you invoke the `IProperty.getAvailableValues()` method, the returned `IAgileList` object can include obsolete list values. This is due to the fact that when the list value is marked obsolete, the server continues to maintain the value in its obsolete list values for existing objects that use these values.

The following example shows how to check whether a list value is obsolete and how to make it obsolete.

Example: Making a list value obsolete

```
public void checkIfObsolete(IAgileList list) throws APIException {
  if (list != null ) {
    if (list.isObsolete() == false) {
      System.out.println(list.getValue());
    }
  }
}
public void setObsolete(IAgileList list, String value) throws
APIException {
  if (list != null ) {
    list.setObsolete(true);
      System.out.println(list.getValue() + " is now obsolete.");
  }
}
```

Setting the List Name and Description

To create a list, you must specify a unique name for it. Therefore, when you use `IListLibrary.createAdminList()`, you must pass a value for the `IAdminList.ATT_NAME`

field. Other `IAdminList` fields, such as `ATT_DESCRIPTION`, are optional. After the list is created, you can modify it's name and description. The following example shows how to set the name and description of a list.

Example: Setting the list name and description

```
try {
  IAdminList list = m_listLibrary.getAdminList("Packaging Styles");
  list.setName("Packaging Color Codes");
  list.setDescription("Color codes for product packaging");
} catch (APIException ex) {
        System.out.println(ex);
}
```

Setting Level Names for a Cascading List

Like list names, the level names for a list must be unique. You can't reuse the level name used by another cascading list. To check if the list with a given name already exists, use `IListLibrary.getAdminList()`. Use one of the following methods to set the level name of a cascading list:

- `IAgileList.setLevelName(int, String)` – Sets the level name for a specified level.
- `IAgileList.setLevelName(String)` – Sets the level name of the current level.

For an example showing how to set the level names of a cascading list, see Creating a Cascading List on page 165.

Note	Level names for cascading lists are not displayed in Agile Java Client or Web Client. However, you can choose to display them in Clients you create with the Agile SDK.

Enabling or Disabling a List

When you create a custom list, you can use the `IAdminList.ATT_ENABLED` field to specify whether it's enabled. If you omit this field, the list is disabled by default. The following example shows how to enable and disable a list after it has been created.

Example: Enabling and disabling a list

```
public void enableList(IAdminList list) throws APIException {
    list.enable(true);
    System.out.println("List " + list.getName() + " enabled.");
}
public void disableList(IAdminList list) throws APIException {
    list.enable(false);
    System.out.println("List " + list.getName() + " disabled.");
}
```

Deleting a List

If a list is not read-only and is not currently being used by an Agile dataobject, you can delete it. Otherwise, the `IAdminList.delete()` method throws an exception. Once you delete a list, it is

removed permanently. You cannot undo the deletion.

The following example shows how to delete a list.

Example: Deleting a list

```
public void deleteList(IAdminList list) throws APIException {
  // Make sure the list is not read-only
  if (!list.isReadOnly()) {
    // Delete the list
    list.delete();
    System.out.println("List " + list.getName() + " deleted.");
  } else {
    System.out.println("List " + list.getName() + " is read-only.");
  }
}
```

Modifying and Removing List Values

The SDK provides the following methods to modify String element entries, or remove an entry in an Agile list:

- The `IAgileList.setValue(Object)` method to modify String list element entries in an Agile Admin list.

> **Note** This method only applies to String values. You can only use this method to modify String entries and not object entries.

- The `IAgileList.clear()` and `ITree.removeChild(Object)` methods to remove any Agile list entry that is not restricted by the applicable business rules.

The following example uses these methods to modify and clear values in an Agile list.

Example: Renaming and removing Admin list entries

```
public void exampleClearList() throws Exception {
  IAdmin admin = m_session.getAdminInstance();
  IListLibrary listLibrary = admin.getListLibrary();

    HashMap map = new HashMap();
    String name = "Color";
    String desc = "Example";
    map.put(IAdminList.ATT_NAME, name);
    map.put(IAdminList.ATT_DESCRIPTION, desc);
    map.put(IAdminList.ATT_ENABLED, new Boolean(true));
    map.put(IAdminList.ATT_CASCADED, new Boolean(false));
    IAdminList newList = listLibrary.createAdminList(map);
IAgileList list = newList.getValues();
list.addChild("RED");
list.addChild("GREEN");
list.addChild("BLUE");
newList.setValues(list);
list = newList.getValues();
```

```
    // Removing the selection
    IAgileList agList = (IAgileList)list.getChild("BLUE");
    Object errorCode = null;
    try {
    list.removeChild(agList);
    }catch(APIException e){
    errorCode = e.getErrorCode();
    }

    //  Clear the list
    list = newList.getValues();
    list.clear();
    newList.setValues(list);

    // Clean up
    newList.delete();
  }
```

Printing Contents of IAgileList Objects

When working with an `IAgileList` object, particularly one with several levels, it's helpful to print the entire hierarchy of the list. The following code prints the list nodes contained within an `IAgileList` object.

Example: Printing list nodes in an IAgileList object

```
private void printList(IAgileList list, int level) throws APIException
{
  if (list != null ) {
    System.out.println(indent(level*4) + list.getLevelName() + ":" +
    list.getValue() + ":" + list.getId());
    Object[] children = list.getChildren();
    if (children != null) {
      for (int i = 0; i < children.length; ++i) {
        printList((IAgileList)children[i], level + 1);
      }
    }
  }
}
private String indent(int level) {
  if (level <= 0) {
    return "";
  }
  char c[] = new char[level*2];
  Arrays.fill(c, ' ');
  return new String(c);
}
```

Chapter 12

Working with Attachments and File Folder Objects

This chapter includes the following:

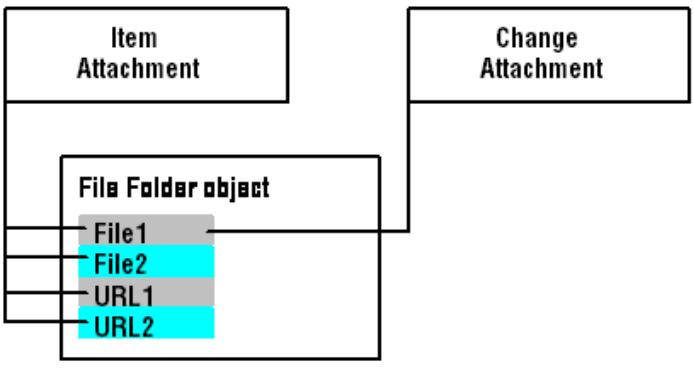
- About Attachments and File Folders 173
- Working with File Folders 174
- Working with Attachments Table of an Object 180
- Checking Out a File Folder 183
- Canceling a File Folder Checkout 183
- Adding Files and URLs to the Attachments Table 184

About Attachments and File Folders

Attachments to objects contain information about the object or a manufacturing process. You can attach files and URLs by referencing them in a *File Folder* object. The File Folder object holds pertinent content, or *Attachments*. Most primary Agile API objects, such as `IItem`, `IChange`, `IManufacturer`, `IManufacturerPart`, `IPackage`, `ITransferOrder`, `IUser`, and `IUserGroup`, have an *Attachments* table (or tab in the Java Client) that lists indirect references to the files or URLs that are in separate file folders. Each row in an Attachments table can refer to one file or to all files from a referenced file folder.

The following figure is an example of the way files or URLs contained in a file folder are referenced indirectly from the Attachments table of multiple business objects, in this case an item and a change.

Figure 13: How Attachments table rows refer indirectly to File Folder files or URLs



The Agile API does not provide support for viewing or printing an attachment. However, after you download a file, you can use another application to view, edit, or print the attachment.

A File Folder is a business object that specifies one or more files or URLs that are stored in the file

PLM server vault. In addition, a file folder has its own set of tables. This means that you can create and load an independent file folder and add one or more files to its Files table. You can also search for a file folder, just as you would search for an Item or Change.

Important Before you try to add Agile PLM attachments and work with file folders, make sure the File Manager Internal Locator property is set in Agile Java Client. Choose Admin > Settings > Server Settings > Locations > File Manager > Advanced > File Manager Internal Locator. The *format* for the value is <*protocol*>://<*machinename*>:<*port*>/<*virtualPath*>/services/FileServer. For example, http://agileserver.agile.agilesoft.com:8080/Filemgr/services/FileServer is a valid value. For more information about Agile PLM server settings, refer to the *Agile PLM Administrator Guide*.

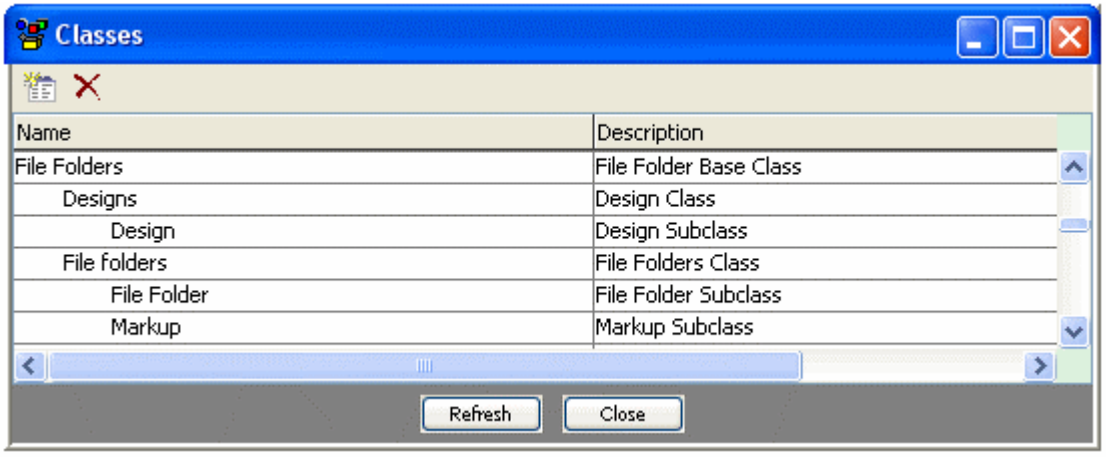
Working with File Folders

Similar to Attachments, the SDK exposes APIs to perform File Folders-related task such as checking-in and checking-out files associated with objects in the rows of an Attachments table, adding files and URLs to an Attachments table, and deleting attachments. This section lists and describes these features, and provides the necessary procedures to use the SDK to perform these tasks.

File Folder Classes and Subclasses

The *File Folder Base Class* has two Classes and each of these classes have their own respective Subclasses. The figure below lists the *File Folders Base Class*, *Classes,* and *Subclasses*. The Agile PLM administrator can define new file folder subclasses.

Figure 14: File Folders Classes and Subclasses



A description of these classes and objects appears in the table below.

Base Class	Class	Subclass	Description
File Folders	Designs	Design	Objects that permit building model structures in CAD

	File folders	File Folder Markup	Objects that include files or URLs; this class includes all file folder objects except historical report file folders.

For information about routing these objects, see Checking the State of Agile PLM Objects on page 29.

File Folder Tables and Constants

The File Folder object supports the following tables and corresponding constants:

Table	Constant	Read/Write Mode
Title Block	TABLE_TITLEBLOCK	Read/Write
Page Two	TABLE_PAGETWO	Read/Write
Page Three	TABLE_PAGETHREE	Read/Write
Files	TABLE_FILES	Read/Write
Structure	TABLE_STRUCTURE	Read/Write
Routing Slip/Workflow	TABLE_WORKFLOW	Read/Write
Relationships	TABLE_RELATIONSHIPS	Read-only
History	TABLE_HISTORY	Read-only
Where Used	TABLE_WHEREUSED	Read/Write
Where Used Design	TABLE_WHEREUSEDDESIGN	Read-only

Creating File Folder Objects

`IFileFolder` is the interface that corresponds to the file folder business object. The following example shows how to create a file folder.

Example: Creating a file folder

```
public void createFileFolder() throws Exception {
IAgileClass attClass =
    m_admin.getAgileClass(FileFolderConstants.CLASS_FILE_FOLDER);
IAutoNumber an = cls.getAutoNumberSources()[0];
String attNumber = an.getNextNumber();
    IFileFolder ff = (
    IFileFolder)m_session.createObject(attClass, attNumber);
    ff.checkOutEx();
}
```

Note When you add a file or a URL to the row of the Attachments table of a business object, you will automatically create automatically a new file folder object that contains the associated file or URL. See Creating File Folder Objects by Adding Rows to Attachments Table on page 178.

The File Folders Design class is similar to the File folder class with the additional Structures table

(Tab in the Java Client UI) for CAD objects. The following examples show how to create a Design object, adding a Design object to a the Structure tree, and loading a structure table.

Example: Creating a Design object

```
    // autoNum is autoNumber as usual
IFileFolder obj = (IFileFolder) m_session.createObject(
FileFolderConstants.CLASS_DESIGN, autoNum);
```

Example: Adding Design objects to a Structure tree

```
IFileFolder obj = // some Design object
IFileFolder childObj1 = // some Design object
IFileFolder childObj2 =  // some Design object


obj.checkOutEx();
ITable table = obj.getTable(FileFolderConstants.TABLE_STRUCTURE);
    // add row
Object[] vers = childObj1.getVersions();
IRow row = table.createRow(childObj1);
row.setValue(FileFolderConstants.ATT_STRUCTURE_LABEL,
    "label modified by creating row 1");
row = table.createRow(childObj2);
row.setValue(FileFolderConstants.ATT_STRUCTURE_LABEL,
    "label modified by creating row 2");
            obj.checkIn();
```

Example: Loading a Structure table

```
      public void testLoadingDesignStructureTable() throws Exception {
            addCaseInfo("Design Object", "load Structure table", "");
    // assuming Design object Design00004 existed with some data in
    Structure
IFileFolder obj = (IFileFolder) m_session.getObject(
    FileFolderConstants.CLASS_DESIGN, "Design00004");
    IAgileClass agileClass = obj.getAgileClass();
    // load Structure table
ITable table = obj.getTable(FileFolderConstants.TABLE_STRUCTURE);
Integer tableId = (Integer) table.getTableDescriptor().getId();
// ITable performs related tasks
      }
```

Example: Loading a Structure table as a tree

```
public void testLoadingDesignStructureTree()
throws Exception
    {addCaseInfo("Design Object", "load Structure tree", "");
    // assuming Design object Design00004 existed with some data in
    Structure
IFileFolder obj = (IFileFolder) m_session.getObject(
    FileFolderConstants.CLASS_DESIGN, "Design00004");
    IAgileClass agileClass = obj.getAgileClass();
    // load Structure table
    ITable table = obj.getTable(FileFolderConstants.TABLE_STRUCTURE);
    Integer tableId = (Integer) table.getTableDescriptor().getId();
    ITreeNode root = (ITreeNode) table;
    Collection topLevelChildren = root.getChildNodes();


      Iterator it;
```

```
      ITreeNode row;
      if (topLevelChildren != null) {
        it = topLevelChildren.iterator();
        int level = 0;
        while (it.hasNext()) {
             row = (ITreeNode) it.next();
             if(row instanceof IRow) {
                IRow aRow = (IRow) row;
                IDataObject referent =
                     aRow.getReferent();
                if(referent != null) {
                   System.out.println(
                    "Row Referent Object ID/row:
                    "+ referent.getObjectId()+ " / "
                    + referent.getName());
                              }
                         }
              iterateTreeNode(agileClass, true,
                     tableId, (ITreeNode) row);
                        count++;
                   }
            }

    System.out.println("The number of rows in top level is " + count);
}

private void iterateTreeNode(IAgileClass agileClass, boolean print,
Integer tableId, ITreeNode node) throws APIException {
    Collection childNodes = node.getChildNodes();
    printRow(agileClass, print, tableId, (IRow) node);
    if (childNodes == null || childNodes.size() <= 0) {
      return;
            }

    Iterator it = childNodes.iterator();
    ITreeNode childNode;
    IRow row;

    while (it.hasNext()) {
      childNode = (ITreeNode) it.next();
      if (childNode instanceof IRow) {
         row = (IRow) childNode;
         if(row instanceof IRow) {
            IDataObject referent =
                  row.getReferent();
            if(referent != null) {
              System.out.println
               ("Row Referent Object ID/row:
               "+ referent.getObjectId()+ " / "
               + referent.getName());
                        }
```

```
                    }
iterateTreeNode(agileClass, print, tableId, (ITreeNode) childNode);
                }
            }
}
```

Creating File Folder Objects by Adding Rows to Attachments Table

When you add a file or a URL to the row of the Attachments table of a business object, you automatically create a new file folder that contains the associated file or URL. You can load the referenced file folder using the `IRow.getReferent()` method, as shown in the following example.

Example: Creating a file folder by adding a row to the Attachments table

```
public IFileFolder addRowToItemAttachments
  (IItem item, File file) throws Exception
  {ITable attTable = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
  IRow row = attTable.createRow(file);
  IFileFolder ff = (IFileFolder)row.getReferent();
  return ff;
}
```

Working with the Files Table of a File Folder

The Files table of a file folder lists the files and URLs associated with the object. To edit the table, you must first check out the file folder. You cannot add files or URLs to the Files table or delete them unless the file folder is checked out.

The following example shows how to check out a file folder and then add files and URLs to the Files table.

Example: Adding files and URLs to the Files table of a file folder

```
public void addFiles(IFileFolder ff, File[] files, URL[] urls) throws
Exception {
  // Check out the file folder
  ff.checkOutEx();

  // Get the Files table
  ITable filesTable = ff.getTable(FileFolderConstants.TABLE_FILES);
  // Add files to the Files table
  for (int i = 0; i < files.length; ++i) {
    filesTable.createRow(files[i]);
  }
  // Add URLs to the Files table
  for (int i = 0; i < urls.length; ++i) {
    filesTable.createRow(urls[i]);
  }
  // Check in the file folder
  ff.checkIn();
}
```

Accessing Files in Agile PLM File Vault with IAttachmentFile

`IAttachmentFile` is the interface that provides generalized access to files stored in the Agile PLM file vault. This interface is supported by the following Agile API objects:

- File folder – you can class cast `IFileFolder` to `IAttachmentFile`.
- A row of the Files table of a file folder – you can class cast `IRow` from the Files table to `IAttachmentFile`.
- A row of the Attachments table of a business object – you can class cast `IRow` from the Attachments table to `IAttachmentFile`.

`IAttachmentFile` provides the following methods for working with attachments:

- `getFile()`
- `isSecure()`

Note `IAttachmentFile` also has a `setFile()` method that lets you change the file(s) for an attachment, but it is supported only for rows of the Attachments table.

The results returned from `IAttachmentFile` methods vary depending on the object you're working with, as shown in the following table.

Calling object	getFile() return value	isSecure() return value
Row from the Attachments table of any business object	Returns either a single file `InputStream` if the row refers to a specific file from the file folder or a zipped `InputStream` with all the files from the file folder.	`true` if the referenced file is not URL, or all the files are not URLs.
`FileFolder` object	Returns a zipped `InputStream` with all files from the file folder.	`true` if all the files contained in the file folder are not URLs.
Row from the Files table of a file folder	Returns a single file InputStream that refers to a specific file from the file folder.	`true` if the referenced file is not a URL.

Note To read files in a zipped `InputStream`, use methods of the `java.util.zip.ZipInputStream` class.

The following example shows how to use `IAttachmentFile.isSecure()` and `IAttachmentFile.getFile()` from the row of an Attachments table for an item.

Example: Using isSecure() and getFile()

```
public InputStream getItemAttachment(IItem item) throws Exception {
  InputStream content = null;
  ITable attachments = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
  IRow row = (IRow)attachments.iterator().next();
  if (((IAttachmentFile)row).isSecure())
    content = ((IAttachmentFile)row).getFile();
  return content;
```

```
}
```

Working with Attachments Table of an Object

To work with the Attachments table of an object, follow this sequence.

1. Get the object that has the attachment you want.

   For example, you can use the `IAgileSession.getObject()` method to get a particular object, or you can create a query to return objects.

2. Get the Attachments table. Use the `IDataObject.getTable()` or `IAttachmentContainer.getAttachments()` methods to get the table.

3. Select a row in the Attachments table.

   Create an iterator for the table, and then select a particular row. You can use the `ITable.getTableIterator()` method to get a bidirectional iterator for the table.

The following example below shows how to retrieve an item, get the Attachments table for the item, and then select the first attachment.

Example: Getting an attachment for an Item

```
try {
// Get Item P1000
  Map params = new HashMap();
  params.put(ItemConstants.ATT_TITLE_BLOCK_NUMBER, "P1000");
  IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, params);
// Get the attachment table for file attachments
  ITable attTable = item.getAttachments();
// Get a table iterator
  ITwoWayIterator it = attTable.getTableIterator();
// Get the first attachment in the table
  if (it.hasNext()) {
    IRow row = (IRow)it.next();
// Read the contents of the stream
    InputSteam stream = ((IAttachmentFile)row).getFile();
  }
  else {
    JOptionPane.showMessageDialog(null, "There are no files listed.",
    "Error", JOptionPane.ERROR_MESSAGE);
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Checking In and Checking Out Files with ICheckoutable

The `ICheckoutable` is an interface that you can use to check in and check out files that are associated with an object. This applies only to rows of the Attachments table. You can class cast `IRow` from the Attachments table to `ICheckoutable`.

`ICheckoutable` provides the following methods for working with attachments:

- `cancelCheckout()`
- `checkIn()`
- `checkOutEx()`
- `isCheckedOut()`

This example shows how to use the `ICheckoutable` interface to check out and check in a file from a row of the Attachments table.

Example: Using ICheckoutable methods to check out and check in an attached file

```
public InputStream checkOutRow(IRow row) throws APIException {

  // Check out the attachment
  ((ICheckoutable)row).checkOutEx();

  // Read the contents of the stream
  InputStream stream = ((IAttachmentFile)row).getFile();
  return stream;
}
public checkInRow(IRow row, String filePath) throws APIException {
  if (row.isCheckedOut()) {
    // Set the new file
    ((IAttachmentFile)row).setFile(new File(filePath));
    // Check in the file
    ((ICheckoutable)row).checkIn();
  }
  else {
    JOptionPane.showMessageDialog(null, "The attachment is not checked
out.",
    "Error", JOptionPane.ERROR_MESSAGE);
  }
}
```

Specifying the Revision of the Item

When you are working with items, each revision can have different attachments. If an item has multiple revisions, your program should allow the user to select a revision. For information about specifying the revision, see Getting and Setting the Revision of an Item on page 109.

Checking whether the Revision Is Incorporated

When the revision for an item is released, the revision is also incorporated. The attachments for an incorporated item are locked and cannot be checked out.

However, you can still view incorporated attachments, but you cannot modify them. To modify an incorporated attachment, you must either un-incorporate the attachment, or submit a new changes order to create new revision as shown in the examples below. For more information about checking whether a revision is incorporated, see Changing the Incorporated Status of a Revision on page 111.

Example: Incorporating Attachments

```
class IncorporateItem implements ICustomAction {
    public ActionResult doAction(IAgileSession session, INode
    actionNode,
                    IDataObject affectedObject)   {
        try {
            System.out.println("Workflow action kicked off....");
            IItem object = (IItem)affectedObject;
            System.out.println("Incorporating...");
            IItem loItem = (IItem) session.getObject(IItem.OBJECT_TYPE,
            object.getName());
    //this will get the latest version. Make sure the latest is against
    a MCO
            loItem.setIncorporated(true);
    //incorporate the attachment
            System.out.println("Attachment added.");
            String message = ("Incorporated "+object);
            return new ActionResult(ActionResult.STRING, message);
        } catch (APIException ae) {
            ae.printStackTrace();
            return new ActionResult(ActionResult.EXCEPTION, ae);
        }
    }
}
```

Example: Un-incorporating Attachments

```
class IncorporateItem implements ICustomAction {
    public ActionResult doAction(IAgileSession session, INode
    actionNode,
                IDataObject affectedObject)   {
        try {
            System.out.println("Workflow action kicked off....");
            IItem object = (IItem)affectedObject;
            System.out.println("Incorporating...");
            IItem loItem = (IItem) session.getObject(IItem.OBJECT_TYPE,
            object.getName());
    //this will get the latest released version
            loItem.setIncorporated(false);
    //Unincorporate the attachment
            System.out.println("Attachment added.");
            String message = ("Incorporated "+object);
            return new ActionResult(ActionResult.STRING, message);
        } catch (APIException ae) {
            ae.printStackTrace();
            return new ActionResult(ActionResult.EXCEPTION, ae);
```

```
        }
    }
}
```

Checking Out a File Folder

Before you can add, delete, or modify the files contained in a file folder, you must check out the file folder. With the appropriate privileges, you can check out a file folder as long as it is not already checked out by another user. Once a file folder is checked out, no one else can check it out or modify it.

The user who checked out a file folder, as well as other users who are change analysts or component engineers, can check it in. If the file folder was checked out to a location on the network, or to a shared drive or directory, anyone who has access to that network location or to that shared directory can check in the file folder.

The following example shows how to check out a file folder.

Example: Checking out a file folder

```
void checkOutFileFolder(IFileFolder ff) throws Exception {
  ff.checkOutEx();
}
```

Note You can also use `ICheckoutable.checkOutEx()` to check out a row of the Attachments table. See Checking In and Checking Out Files on page 180.

Canceling a File Folder Checkout

If you check out a file folder and then decide that you don't want to modify it, or you want to discard your changes and revert to the original file folder, you can cancel the checkout. When you cancel a checkout, you also make the file folder available for other users to check out.

Note Only the user who checked out a file folder can cancel the checkout.

This example shows how to cancel a checkout of a file folder.

Example: Canceling checkout of a file folder

```
void cancelCheckOut(IFileFolder ff) {
  // Show a confirmation dialog box
  int i = JOptionPane.showConfirmDialog(null,
  "Are you sure you want to cancel checkout?",
  "Cancel Checkout", JOptionPane.YES_NO_OPTION);

  // If the user clicks Yes, cancel checkout
  try {
    if (i == 0) {
      ff.cancelCheckout();
    }
  } catch (APIException ex) {
```

```
      System.out.println(ex);
    }
  }
```

Note You can also use `ICheckoutable.cancelCheckout()` to cancel checkout of a row of the Attachments table. See Checking In and Checking Out Files on page 180.

Adding Files and URLs to the Attachments Table

The Agile API lets you add files and URLs to the Attachments table of many types of objects, such as `IItem`, `IChange`, `IManufacturerPart`, and `IManufacturer`. An attachment is one or more physical files or an Internet address (URL). A file is considered a secured attachment because it is physically stored in the Agile PLM file vault. A URL, on the other hand, is an unsecured attachment.

When you add a file or a URL to the Attachments table of a business object, the server automatically creates a new file folder containing the associated file or URL. The new row on the Attachments table references the new file folder.

When you add a URL attachment, the server stores a reference to the Internet location but does not upload a file. Therefore, you cannot download a URL attachment. The Agile API validates URL strings that you attempt to check in as an attachment. If a URL is invalid, the Agile API considers the string a filename instead of a URL.

You cannot add a file or URL to the Attachments table of an item if

- The current revision has a pending or released MCO.
- The current revision is incorporated.

When you use the `ITable.createRow(java.lang.Object)` method to add a row to the Attachments table, the `param` method can be any of the following object types:

- `String` – adds one file attachment specified by a local path.
- `String[]` – adds multiple file attachments specified by an array of local paths.
- `File` – adds one file attachment.
- `File[]` – adds multiple file attachments.
- `InputStream` – adds one file attachment.
- `InputStream[]` – adds multiple file attachments.
- `URL` – adds one URL attachment.
- `URL[]` – adds multiple URL attachments.
- `IRow` (of the Attachments or Files tables) – adds a file or URL attachment.
- `IFileFolder` – adds all files and URLs for the specified file folder.
- `Map` – adds one or more files specified by a hash table containing Attachment parameters.

Note The `File` object type performs best when adding attachments.

When you add a file or a URL to the row of the Attachments table of a business object, you automatically create a new file folder that contains the associated file or URL. You can load the referenced file folder using the `IRow.getReferent()` method, as shown in the following example.

Example: Creating a file folder by adding a row to the Attachments table

```
public IFileFolder addRowToItemAttachments
  (IItem item, File file) throws Exception
  {ITable attTable = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
  IRow row = attTable.createRow(file);
  IFileFolder ff = (IFileFolder)row.getReferent();
  return ff;
}
```

This example uses several instances of the `addAttachment()` methods to illustrate the different ways you can add rows to an Attachments table.

Example: Adding files to the Attachments table

```
// Add a single file to the Attachments table row by specifying a file
path
public static IRow addAttachment(ITable attTable, String path) throws
APIException {
  IRow row = attTable.createRow(path);
  return row;
}
// Add a single file to the Attachments table
public static IRow addAttachment(ITable attTable, File file) throws
APIException {
  IRow row = attTable.createRow(file);
  return row;
}
// Add multiple files to the Attachments table
public static IRow addAttachment(ITable attTable, File[] files) throws
APIException {
  IRow row = attTable.createRow(files);
  return row;
}
// Add a URL attachment to the Attachments table
public static IRow addAttachment(ITable attTable, URL url) throws
APIException {
  IRow row = attTable.createRow(url);
  return row;
}
// Add a file folder to the Attachments table
public static IRow addAttachment(ITable attTable, IFileFolder ff)
throws APIException {
  IRow row = attTable.createRow(ff);
  return row;
}
// Add an FileFolder.Files row object or a [BusinessObject].Attachments
row object
```

```
// to the Attachments table. The Agile API validates the row object at
run time to
// determine if it is from a valid table (Files or Attachments).
public static IRow addAttachment(ITable attTable, IRow filesRow) throws
APIException {
  IRow row = attTable.createRow(filesRow);
  return row;
}
// Add a file folder to the Attachments table and specify the version
for all files
public static IRow addAttachmentWithVersion(ITable attTable,
IFileFolder ff) throws APIException {
  ff.setCurrentVersion(new Integer(1));
  IRow row = attTable.createRow(ff);
  return row;
}
```

Deep Cloning Attachments and Files from One Object to Another

To simplify copying file attachments from one object to another, use the `CommonConstants.MAKE_DEEP_COPY` virtual attribute as a Boolean parameter of `ITable.createRow(Object)`. This parameter allows your program to create a new copy of the file in the Agile File Manager vault instead of referencing the old file.

Example: Deep cloning an Attachments table row

```
// Clone an attachment table row and its file from one item to another
public static cloneAttachment(IItem item1, IItem item2, File file)
throws APIException {
  // Get the attachments tables of item1 and item2
  ITable tblAttach1 = item1.getAttachments();
  ITable tblAttach2 = item2.getAttachments();

  // Prepare params for the first row
  HashMap params = new HashMap();
  params.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, file);

  // Add the file to the attachments table of item1
  IRow row1 = tblAttach1.createRow(params);

  // Prepare params for the second row
  params.clear();
  params.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, row1);
  params.put(CommonConstants.MAKE_DEEP_COPY, Boolean.TRUE);
  // Add the same file to the attachments table of item2 IRow row2 =
tblAttach2.createRow(params);
}
```

Example: Deep cloning the Files table row of a File Folder

```
// Clone a Files table row and its file from one File Folder to another
```

```
public static cloneFilesRow(IFileFolder folder1, IFileFolder folder2,
File file) throws APIException {
  // Check out folder1 and folder2
  folder1.checkOutEx();
  folder2.checkOutEx();

  // Get the Files tables of folder1 and folder2
  ITable tblFiles1 = folder1.getTable(FileFolderConstants.TABLE_FILES);
  ITable tblFiles2 = folder2.getTable(FileFolderConstants.TABLE_FILES);
  // Prepare params for the first row
  HashMap params = new HashMap();
  params.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, file);

  // Add the file to the attachments table of folder1
  IRow row1 = tblFiles1.createRow(params);

  // Prepare params for the second row
  params.clear();
  params.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, row1);
  params.put(CommonConstants.MAKE_DEEP_COPY, Boolean.TRUE);
  // Add the same file to the Files table of folder2
  IRow row2 = tblFiles2.createRow(params);

  // Check in folder1 and folder2
  folder1.checkIn();
  folder2.checkIn();
}
```

Specifying the File Folder Subclass When Adding Attachments

You can set up your Agile PLM system with multiple file folder subclasses. If so, when you add a file folder to the Attachments table of a business object, you may want to specify which file folder subclass to use. If you don't specify a subclass, the Agile API uses the default File Folder subclass. The virtual attribute CommonConstants.ATT_ATTACHMENTS_FOLDERCLASS makes it easier to specify the required file folder subclass. It enables you to set the attribute to any file folder subclass.

The following example shows how to use the ATT_ATTACHMENTS_FOLDERCLASS attribute to specify a subclass when you add a file folder to the Attachments table.

Example: Specifying the file folder subclass when adding attachments

```
IAgileClass ffclass = m_admin.getAgileClass("MyFileFolder");
// init item
IItem item = (IItem)session.createObject(ItemConstants.CLASS_PART,
"P0001");
// get attachments table
ITable tab_attachment = item.getAttachments();

// prepare map
```

```
HashMap map = new HashMap();
map.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, new
File("files/file.txt"));
map.put(CommonConstants.ATT_ATTACHMENTS_FOLDERCLASS, ffclass);

// add file
IRow row = tab_attachment.createRow(map);
```

Retrieving Attachment Files

If a file folder is checked out by another user, you can still retrieve a copy of the file folder file(s) and save it to your local machine. The `IAttachmentFile.getFile()` method returns the file stream associated with a row of the Attachments table. The file stream can be for one file or it can be a zipped file stream for multiple files, depending on how many files the associated file folder has. You can also use `IAttachmentFile.getFile()` to get one or more files directly from a file folder instead of accessing the Attachments table of another business object. If you call `getFile()` from the file folder object, you return the zipped file stream for all files listed on the Files table. If you call `getFile()` from a row of the Files table of a file folder, you return a file stream for the specific file associated with that row.

Note When you use `IAttachmentFile.getFile()`, only file attachments are included in the returned file stream. URL attachments don't have files associated with them.

This example shows how to retrieve a copy of an attached file.

Example: Getting attachment files

```
// Get one or more files associated with the row of an Attachments
table or a Files table
public InputStream getAttachmentFile(IRow row) throws APIException {
  InputStream content = ((IAttachmentFile)row).getFile();
  return content;
}
// Get all files associated with a file folder
public InputStream getAttachmentFiles(IFileFolder ff) throws
APIException {
  InputStream content = ((IAttachmentFile)ff).getFile();
  return content;
}
```

If you use `IFileFolder.getFile()` to return a zipped file stream for all files contained in a file folder, you can extract files from the zipped `InputStream` using methods of the `java.util.zip.ZipInputStream` class, as shown in the following example.

Example: Extracting files from a zipped file stream

```
static void unpack(InputStream zippedStream) throws IOException {
  ZipInputStream izs = new ZipInputStream(zippedStream);
  ZipEntry e = null;
  while ((e = izs.getNextEntry()) != null) {
    if (!e.isDirectory()) {
      FileOutputStream ofs = new FileOutputStream(e.getName());
```

```
        byte[] buf = new byte[1024];
        int amt;
        while ((amt = izs.read(buf)) != -1) {
          ofs.write(buf, 0, amt);
        }
        ofs.close();
      }
    }
    zippedStream.close();
  }
```

The Agile API provides no direct method for opening an attachment file. However, you can retrieve a file and then have your program open it in a separate application or display it in a browser window.

Deleting Attachments and File Folders

To delete a file folder, which may contain multiple files, use the `IDataObject.delete()` method. You must have the Delete privilege for file folders to be able to delete them. For more information about deleting objects, see Deleting and Undeleting Objects on page 31.

Note	Deleting a file folder does not automatically remove its associated files from the file server. The Agile PLM administrator is responsible for purging deleted files.

To delete a row from the Attachments table of a business object, use the `ITable.removeRow()` method. For more information, see Removing Table Rows on page 79. Removing a row from the Attachments table does not delete the associated file folder. You cannot delete a row from the Attachments table in the following situations:

- The parent object is an Item whose revision is incorporated.
- The selected attachment is currently checked out.

Working with Thumbnails

Agile PLM supports adding small static graphical images (thumbnails) to key objects which either represent graphical objects or require images. For example, documents attached as files such as Excel worksheets, text files, PDF files, CAD files and so on. Thumbnails display scaled down versions of these files and in case of Part objects, show how they relate to each other.

The SDK supports the following Thumbnail-related functions:

- Regenerating Thumbnails
- Sequencing Thumbnails
- Setting Master Thumbnails
- Generating Thumbnails while adding Files to Attachments tab

Accessing Thumbnails

Agile SDK provides the `IThumbnailContainer` interface for generalized access to thumbnail-related operations for file folder and business objects. This interface is supported by the following API objects:

- `IFileFolder object`
- `IItem object`
- `IManufacturerPart object`

For `IFileFolder` objects, set the applicable version using `IFileFolder.setCurrentVersion` before calling the above APIs. The default version is `LATEST_VERSION`. For `IItem` or `IManufacturerPart` objects, use the revision that is already set on these objects.

The following example gets thumbnail details from `TitleBlock` of an `IItem` or `IFileFolder` object .

Example: Getting thumbnail details from TitleBlock of IItem or IFileFolder objects

```
 IItem dataObj =
        (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
 ITable titleBlockTable =
        this.itemObj.getTable(TableTypeConstants.TYPE_PAGE_ONE);
 Iterator i =
        titleBlockTable.getTableIterator();
 while (i.hasNext()) {
     IRow row = (IRow)i.next();
     Object thumbnailIDDetails =
            row.getValue(ThumbnailConstants.ATT_THUMBNAIL_ATTACHMENT_TAB)
            ;
     IAgileList[] nodes =
            ((IAgileList)thumbnailIDDetails).getSelection();
     for(int ii=0; ii<nodes.length; ii++) {
         IAgileList childNode = nodes[ii];
         IThumbnailID thumbnailID = (IThumbnailID)childNode.getValue();
     }
}
```

Regenerating Thumbnails

Regenerating a thumbnail means generating a thumbnail for an existing (generated) thumbnail for file folder and item objects. This feature is of particular interest in assembly structures where a change in the child of the assembly structure is reflected in the thumbnail after the thumbnail is regenerated.

Agile SDK provides the `IThumbnailContainer.generateThumbnail(IThumbnailID)` API for this purpose. When invoked, it will generate and return a new thumbnail. In case of `IFileFolder` objects, API will use the current *version* that of the object. For `IItem` or `IManufacturerPart` objects, it will use the current *revision* of the object. An `APiException` is thrown when the API fails to regenerate the thumbnail for the specified `thumbnailID` parameter.

Example: Regenerating a thumbnail for an IFileFolder object

```
 IFileFolder ff =
```

```
    (IFileFolder)session.getObject(IFileFolder.OBJECT_TYPE,
    "FOLDER00037");
ff.setCurrentVersion(new Integer(1));
IThumbnailID oldThumbnailID = "";
//get this id from row of supported tables like Title Block
ff.generateThumbnail(oldThumbnailID);
```

Example: Regenerating a thumbnail for an IItem object

```
IItem itemObj =
    (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
IThumbnailID oldThumbnailID = "";
//get this id from row of supported tables like Title Block
itemObj.generateThumbnail(oldThumbnailID);
```

Setting Master Thumbnails

In Agile PLM, a file folder object is represented by a thumbnail file which can contain several files in its Files tab. Using **SetMasterThumbnai**l, a user can decide which row in Files tab will represent the selected thumbnail's file folder.

SDK provides the `void setMasterThumbnail (IRow masterRow ) throws APIException` API to set master thumbnails on file folder objects. An exception thrown if the function fails to set the master thumbnail represented by the parameter `masterRow`.

Example: Setting a master thumbnail

```
IFileFolder ff =
    (IFileFolder)session.getObject(IFileFolder.OBJECT_TYPE,
    "FOLDER00037");
ff.setCurrentVersion(new Integer(1));
ITable attachmentTable =
    ff.getTable(FileFolderConstants.TABLE_FILES);
Iterator i =
    attachmentTable.getTableIterator();
while (i.hasNext()) {
      IRow row = (IRow)i.next();
      IRow masterRow = null;
     //set one of the rows as the master row
}

ff.setMasterThumbnail(masterRow);
```

Replacing Thumbnails

You can replace an Agile PLM generated thumbnail with user provided image for file folder and item objects. The SDK provides the following API for this purpose.

```
IThumbnailID replaceThumbnail (IThumbnailID oldThumbnailID, byte[] bytes)
throws APIException
```

This API will replace the thumbnail referred to in `oldThumbnailID` with the image file referred to in the input stream. That is, it will return the `thumbnailID` of the replaced thumbnail.

For `IFileFolder` objects, the API will use the *version* that is already set on the object. For `IItem` or `IManufacturerPart` objects, it will use the *revision* that is already set on the object. An `APiException` is thrown if it fails to replace the thumbnail specified in the `oldThumbnailID` parameter.

Example: Replacing a thumbnail for an IFileFolder object

```
IFileFolder ff =
   (IFileFolder)session.getObject(IFileFolder.OBJECT_TYPE,
   "FOLDER00037");
ff.setCurrentVersion(new Integer(1));
IThumbnailID oldThumbnailID = "";
//get this id from row of supported tables like Title Block
String filePath = "C:\\Earth.bmp";
File file1_tmp = new File(filePath);
byte[] b1 = new byte[(int)file1_tmp.length()];
FileInputStream fileInputStream = new FileInputStream(file1_tmp);
fileInputStream.read(b1);
IThumbnailID newThumbnailID = ff.replaceThumbnail(oldThumbnailID, b1);
```

Example: Replacing a thumbnail for an IItem object

```
IItem itemObj =
   (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
IThumbnailID oldThumbnailID = "";
//get this id from row of supported tables like Title Block
String filePath = "C:\\Earth.bmp";
File file1_tmp = new File(filePath);
byte[] b1 = new byte[(int)file1_tmp.length()];
FileInputStream fileInputStream = new FileInputStream(file1_tmp);
fileInputStream.read(b1);
IThumbnailID newThumbnailID = itemObj.replaceThumbnail(oldThumbnailID,
b1);
```

Sequencing Thumbnails

When Web Client users add attachment files to business objects, they can also set the order (sequence) of their appearance in the Thumbnail Navigator. Agile PLM provides the `void setThumbnailSequence (IThumbnailID[] thumbnailIDs) throws APIException` API to enable this feature in the SDK. For `IItem` or `IManufacturerPart` objects, the API will use the revision that is already set on the object. The API will sort (sequence) the order of appearance using the `thumbnailIDs` parameter. An exception thrown if the function fails to set the master thumbnail.

Example: Sequencing thumbnails

```
IItem itemObj =
   (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
IThumbnailID[] thumbnailIDs = null;
//get this id from row of Title Block table
IThumbnailID[] newSeqOfThumbnailIDs = null;
//generate new order using thumbnail IDs
itemObj.setThumbnailSequence(newSeqOfThumbnailIDs);
```

Generating Thumbnails while Adding Files to Attachments Tab

There are no APIs specifically for this purpose. When you add a file in Attachments tab of Item, a thumbnail is generated for that file provided thumbnail support is enabled in the Web Client.

Working with Design Objects

A *Design* object is a business object that specifies one or more URLs or files stored in Agile PLM's File Management Server. It contains information about the binary files attached to it. Similar to other Agile PLM business objects, Design objects appear in Agile PLM's class hierarchy as a separate base class.

Designs class objects are used with Agile PLM's Engineering Collaboration (EC) module which is used to manage CAD data in Agile PLM. Objects created in this class have many of the same properties and behaviors of File folders. Users with administrator privileges enable opening and working with Design objects in Java Client and Agile PLM users can the access and work exclusively with these objects in Web Client.

Agile SDK support the following Design object-related functions:

- Managing (adding, removing, getting, and editing) version specific Relationships between two Design objects
- Using where-used queries for Design object deployments in Agile PLM Class structures. For information about where-used queries, see Creating a Where-Used Query on page 63.

Adding and Loading Design Objects

To create or get an `IDesign` object, you can use `IAgileSession.createObject()` or `IAgileSession.getObject()`. The following examples show the different methods provided by the SDK to create and get Design objects.

Example: Creating a Design by class name

```
IDesign des = (IDesign)
    m_session.createObject("Design", "DESIGN00133");
```

Example: Creating a Design by class ID

```
IDesign des = (IDesign)
    m_session.createObject(FileFolderConstants.CLASS_DESIGN,
    "DESIGN00133");
```

Example: Creating a Design by IAgileClass reference:

```
IDesign des = (IDesign)
    m_session.createObject(desClass, "DESIGN00133");
```

Example: Loading a Design object

```
IDesign des = (IDesign)
    m_session.getObject(IDesign.OBJECT_TYPE, "DESIGN00133");
```

Managing Version Specific Relationships between Design Objects

Agile SDK supports the following version specific Relationships functions between Design objects:

Note	These version specific functions only apply to Design objects.

- Adding version specific relationships between Design objects
- Removing version specific relationships between Design objects
- Getting version specific relationships for specific versions of Design objects
- Editing version specific relationship for Design objects

Adding Relationships for Specific Versions of Design Objects

The SDK provides the following API to add relationships between two specific versions of Design objects:

`IDesign.`**`addVersionSpecificRelationship`**`(Object versionNum, IDesign relatedDesign, Object relatedVersionNum)`

The parameters are:

- **`versionNum`** – This an integer showing the version number of this Design object.
- **`relatedDesign`** – The Design object you are creating the Relationships for.
- **`relatedVersionNum`** – This an integer showing the version number of the Design object you are creating the Relationship for.

An APIException is thrown if the version specific relationship between the two Design objects was not created.

Alternatively, you can load the object's `RelationshipsTable` and call `createRow(Object params)` with the following `params`:

```
HashMap params = new HashMap();
params.put(DesignConstants.ATT_RELATIONSHIPS_REV_VERSION, versionNum);
params.put(DesignConstants.ATT_RELATIONSHIPS_NAME, relatedDesign);
params.put(DesignConstants.ATT_DESIGN_VERSION, relatedVersionNum);
```

Removing Relationships for Specific Versions of Design Objects

To remove Version Specific Relationships for `IDesign`:

```
IDesign des1 = (IDesign)session.getObject(IFileFolder.OBJECT_TYPE,
"DESIGN00001");
des1.setCurrentVersion(new Integer(4));
ITable relationshipTable = des1.getRelationship();
relationshipTable.removeRow(row);
```

Getting Relationships for Specific Versions of Design Objects

To get the Relationships for a specific version of `IDesign`:

```
IDesign des1 = (IDesign)session.getObject(IFileFolder.OBJECT_TYPE,
"DESIGN00001");
des1.setCurrentVersion(new Integer(4)); //set desired version
ITable relationshipTable = des1.getRelationship();
```

Editing Relationships for Specific Versions of Design Objects

To edit the Relationships for a specific version of the `IDesign` object.

```
IDesign des1 = (IDesign)session.getObject(IFileFolder.OBJECT_TYPE,
"DESIGN00001");
des1.setCurrentVersion(new Integer(4));
ITable relationshipTable = des1.getRelationship();
HashMap mapForUpdate=new HashMap();
HashMap rowUpdateMap = new HashMap();
rowUpdateMap.put(DesignConstants.ATT_DESIGN_VERSION, new Integer(1));
mapForUpdate.put(row1, rowUpdateMap);
relationshipTable.updateRows(mapForUpdate);
```

Purging Specific Versions of Design Objects

The SDK provides the `IDesign.purgeVersions(Object[] versions)` API for purging specific versions of Design objects and relevant versions of its child objects. The `versions` parameter, an integer value, specifies the version number you want purged. An exception is thrown if the API fails to purge the object.

Searching Design Object Deployments with Where-Used Queries

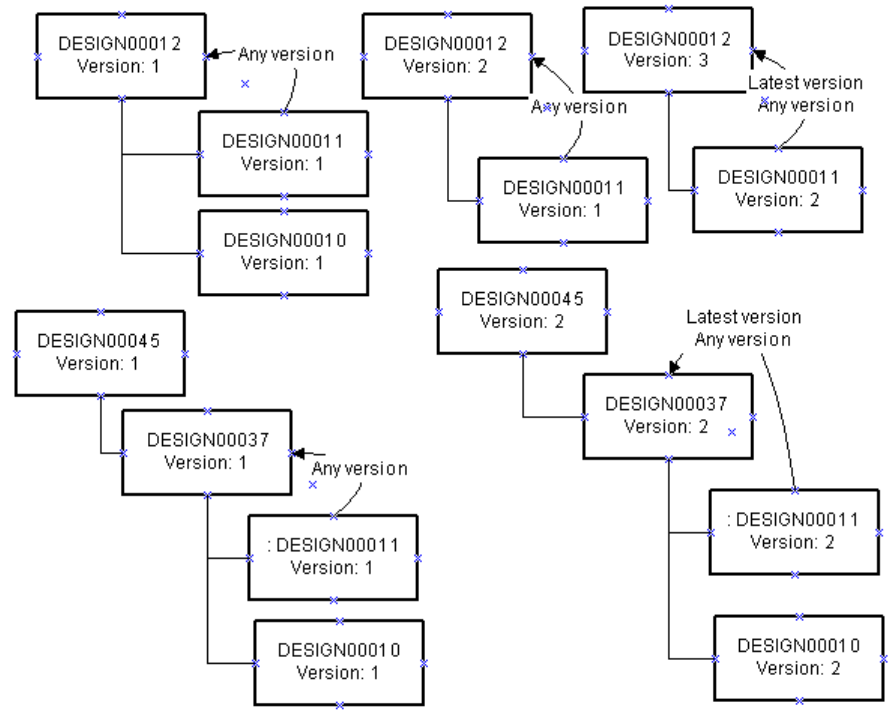
The Structure tab for Design objects enables users to create structures of different Design objects having different versions. The SDK supports searching for Design object usage in Agile PLM Class Structures for the *latest* checked in versions and *all* checked in *versions* with the following queries and query constants:

- `WHERE_USED_IN_STRUCTURE_ONE_LEVEL_LATEST_CHECKEDIN` – This `WHERE_USED` query *returns* the *LATEST* version of the immediate parent of the Design object which uses the input Design object as a child in the design structure. The constant `QueryConstants.WHERE_USED_IN_STRUCTURE_ONE_LEVEL_LATEST_CHECKEDIN` supports this search.
- `WHERE_USED_IN_STRUCTURE_ALL_LEVEL_LATEST_CHECKEDIN` – This `WHERE_USED` query *returns ALL* versions of the immediate parent of the Design object which uses the input Design object as a child in the design structure. The constant `QueryConstants.WHERE_USED_IN_STRUCTURE_ONE_LEVEL_ALL_CHECKEDIN` supports this search

You can find code samples using `QueryConstants` in SDK_samples.zip folder. To access this file, see the Note in Client-Side Components on page 2. These are the Javadoc generated HTML files in the documentation folder.

The two searches and their respective results are explained with the aid of the following illustration. It shows the Design objects and level one structures. The search parameter `Title Block.Number` includes the number 11.

Figure 15: Design objects and search results



Search Objectives and settings

Search for: Designs
Search type: Where-used in structure one level for the **latest** checked in version only
Search type: Where-used in structure one level for the **all** checked in version only
Search parameters: **Title Block.Number** contains **11**

Search results
LATEST checked in versions

Object found	Version
DESIGN00012	3
DESIGN00037	2

Search results
ALL checked in versions

Object found	Version
DESIGN00012	1
DESIGN00012	2
DESIGN00012	3
DESIGN00037	1
DESIGN00037	2

Chapter 13

Importing and Exporting Data with SDK

This chapter includes the following:

- About Importing and Exporting Data 197
- Validating Import Data and Importing Data 197
- Exporting Data from the SDK 200

About Importing and Exporting Data

You can use the SDK to import and export data from external databases into the PLM system. The source can be an Agile database, a third party Product Data Management (PDM) system, or an Enterprise Resource Planning (ERP) system. The following paragraphs provide background information, procedures, and examples to perform these tasks using the agile SDK.

Validating Import Data and Importing Data

When you import data, you have the option to validate the data, or ignore this step. The purpose of import validation is to check the data for compliance with applicable server rules such as length tolerances, allowable values, and other constraints. The validation process informs you the data that will fail to import before initiating the process.

The SDK exposes two methods to programmatically perform the following import-related tasks: .

- The `IImportManager.validateData(byte[], String, byte[], byte[], String[], List)` method to validate the imported data for compliance with server business rules. This action is performed before importing the data to identify the invalid items in the input source data.
- The `IImportManager.importData(byte[], String, byte[], byte[], String[], List)` method supports importing data into the PLM databases. This action is performed after running the `IImportManager.validateData()` method to select the data that meets the server business rules and is importable into the PLM system.

For more information about importing data, refer to *Agile Integration Services Developer Guide* and *Agile Import and Export Guide*.

The following example uses these methods to validate the imported data for compliance and import it into the PLM system upon validation.

Validating Data and Importing Data with SDK

Example: Validating and Importing Data into PLM

```
import com.agile.api.*;
import java.util.*;
```

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class ImportClient {
    public static IAgileSession session = null;
    public static AgileSessionFactory factory;
    public static void main(String[] args) {
        try {
    String _url="http://localhost/Agile";
    String _user="admin";
    String _pwd="agile";
    String srcFilePath="bom.txt";

/* Supported file types: aXML,IPC2571, ExcelFile, DelimitedTextFile
 * The value of "-f" parameter is the same
 * in Import AIS sample command
*/

    String srcFileType="DelimitedTextFile";
// Null implies loading the default mapping
    String mappingPath="NewMapFile.xml";
// Null implies do not transform
    String transformPath=null;
/* The value used by operations is the
    same as the value of the "-t" parameter in the import AIS sample
    command.
    all supported
    operations:"items.bom","items.aml","items.attachments",
    "items.relationships", "manufacturers",
    "manufacturers.relationships",
    "manufacturers.attachments", "manufacturerParts",
    "manufacturerParts.relationships",
    "manufacturerParts.attachments" "partgroups",
    "partgroups.relationships",
    "partgroups.attachments", "productServiceRequests"
    "productServiceRequests.affectedItems",
    "productServiceRequests.relatedPSR"
    "productServiceRequests.relationships"
    "productServiceRequests.attachments" "qualityChangeRequests"
    "qualityChangeRequests.affectedItems"
    "qualityChangeRequests.relationships"
    "qualityChangeRequests.attachments"
*/
    String [] operations=new String[]{"items", "items.bom","items.aml"};
/* The value used by options is the
 * same as the "-n" parameter in  the import AIS sample command
*/
    List options=new ArrayList();
```

```
    options.add("BusinessRuleOptions|ChangeMode=Authoring");
    options.add("BusinessRuleOptions|BehaviorUponNonExistingObjects=Acce
    pt");
    String _output="log.xml";
    FileOutputStream fop=new FileOutputStream(_output);
// Create an instance of IAgileSession
    session = connect(_url,_user,_pwd);
    IImportManager imgr = (IImportManager)
    session.getManager(IImportManager.class);
    byte[] logData=null;
/* Sample code to import data
 * Remove comments to run the importData example.
    * byte[]logData=imgr.importData(stream2byte
    * (new FileInputStream(srcFilePath)),
    * srcFileType, convertFiletoStream(mappingPath),
    * convertFiletoStream(transformPath),
    * operations, options);
 * Sample code to validate data
 * Remove comments to run the validateData example
    * logData=imgr.validateData(stream2byte
    * (new FileInputStream(srcFilePath)),
    * srcFileType, convertFiletoStream(mappingPath),
    * convertFiletoStream(transformPath), * operations, options);
    * byte buf[]=new byte[1024*4];

    int n=0;
    InputStream logStream=byte2stream(logData);
    while((n=logStream.read(buf))!=-1){
    fop.write(buf, 0, n);
      }
    fop.close();
      }
    catch (Exception e) {e.printStackTrace();
      }
    finally {session.close();
      }
}

/*
    * <p> Create an IAgileSession instance </p>
    *
    * @return IAgileSession
    * @throws APIException
*/
    private static IAgileSession connect(String _url,String _user,String
    _pwd) throws APIException {
    factory = AgileSessionFactory.getInstance(_url);
    HashMap params = new HashMap();
    params.put(AgileSessionFactory.USERNAME, _user);
    params.put(AgileSessionFactory.PASSWORD, _pwd);
    session = factory.createSession(params);
    return session;
    }

    private static byte[] stream2byte(InputStream stream) throws
    IOException {
```

```
    ByteArrayOutputStream outStream=new ByteArrayOutputStream();
    byte buf[]=new byte[1024*4];
    int n=0;
    while((n=stream.read(buf))!=-1){
    outStream.write(buf, 0, n);
          }
    byte[] data=outStream.toByteArray();
    outStream.close();
    return data;
    }

    private static InputStream byte2stream(byte[] data) throws
    IOException{
    ByteArrayInputStream stream=new ByteArrayInputStream(data);
    return stream;
    }

    private static byte[] convertFiletoStream(String path) throws
    IOException{
    if(path==null || path.equals(""))
    return null;

    return stream2byte(new FileInputStream(path));
    }
}
```

Exporting Data from the SDK

The SDK exposes the `exportData()` method to programmatically export data from PLM databases. This method is designed to overcome performance and memory issues that are encountered when loading large BOMs into the SDK programs. To overcome this issue, you can invoke the export functionality to load the BOM. The SDK programs can then read and export the data from the extracted XML files.

For more information about exporting data, refer to the *Agile Integration Services Developer Guide* and *Agile Import and Export Guide*.

Invoking SDK's Export Function

Use the following call to invoke the export function of the SDK.

```
public byte[] exportData (Object[], Integer, String[])
```

In this call,

- `exportData` – Is the method that returns the exported data in an array bytes. The byte array represents a ZIP file that contains the export XML file in aXML or PDX formats and any file attachments that are included in the exported package
- `Object[]` – Is the array of objects that are exported from the PLM to the external system. These objects are passed as `IDataObject` objects.
- `Integer` – Is the indicator (constants that are provided in `ExportConstants.java`) to identify whether the output export format should be aXML or PDX. These are the two formats that the SDK supports.

- `String[]` – Is the array of ACS filter names that are used for the export. The filter names are not case sensitive and must match the names of filters defined by the Admin tool for ACS.

The conditions that wil cause the **`exportData`** method to throw an exception and the respective exceptions are:

- Invalid Data Format – The method was called with an unrecognized value for the export data format. Only aXML (provide constant label) and PDX (provide constant label) values are valid.
- No Filter Specified – The method was called but no filters were specified. At least one valid filter must be provided.
- Specified Filter Not Found – The method was called with specified filter which was not found in the system

Example: Exporting data from PLM using the SDK

```
... //

 IItem item = (IItem) session.getObject(IItem.OBJECT_TYPE, "P0001");
 if (item == null) {
    ... // throw an error, the part wasn't found

 }
 IDataObject[] expObjs = {item};
 String[] filters = {"Default Item Filter"};

 ...

 IExportManager eMgr = (IExportManager)
session.getManager(IExportManager.class);
 try {
    byte[] exportData = eMgr.exportData(expObjs,
ExportConstants.EXPORT_FORMAT_PDX, filters);
    if (exportData != null) {
       String fileName = createOutputFileName();
       FileOutputStream outputFile = new FileOutputStream(fileName);
       outputFile.write(exportData);
       outputFile.close();
       System.out.println("Data exported to file: " + fileName);
    }
 } catch (Throwable t) {
    ...  // error handling
 }
 ...
```

Chapter 14

Managing Workflow

This chapter includes the following:

- About Workflow 203
- Selecting a Workflow 205
- Adding and Removing Approvers 206
- Approving or Rejecting Change 215
- Commenting a Change 216
- Auditing a Change 217
- Changing the Workflow Status of an Object 218
- Sending an Agile Object to Selected Users 220
- Sending an Agile Object to User Groups 221

About Workflow

Agile has electronic routing, Notification, and signoff capabilities, thus automating the change control process and providing a simplified but powerful Workflow mechanism. With these Workflow features, you can

- Route changes automatically to the users who need to approve or observe the change.
- Send email alerts automatically to approvers and observers to notify them that a change has been routed to them.
- Approve or reject changes online.
- Attach comments to changes.

The Change Control Process

The change control process can vary for each Workflow defined for a routable object. The table below lists the sequences for the default Workflows for each type of routable object. For changes the first four steps in the sequence are identical and only the final step is different.

Workflow	Default sequence
Default Activities	Not Started > In Process > Complete
Default Attachments	Review
Default Audits	Prepared > Initiated >.Audited >.Issued >.Corrected > Validated >.Closed
Default CAPAs	Identified > Acknowledged > Investigated > Implemented > Validated > Closed
Default Change Orders	Pending > Submitted > CCB > Released > Implemented
Default Change Requests	Pending > Submitted > CCB > Released > Closed

Workflow	Default sequence
Default CTOs	Pending> Review > Released > Complete
Default Declarations	Pending > Open to Supplier > Submit to Manager > Review > Released > Implemented
Default Deviations	Pending > Submitted > CCB > Released > Expired
Default Gates	Closed > In Review > Open
Default Manufacturer Orders	Pending > Submitted > CCB > Released > First Article Complete
Default Non-Conformance Reports	Pending > Submitted > Review > Released > Closed
Default Packages	Pending > Submitted > Review > Accepted > Closed
Default Price Change Orders	Pending > Submitted > Price Review > Released > Implemented
Default Problem Reports	Pending > Submitted > Review > Released > Closed
Default Sites Change Orders	Pending > Submitted > CCB > Released > Implemented
Default Stop Ships	Pending > Submitted > CCB > Released > Resumed

Dynamics of Workflow Functionality

The Workflow functionality available to each user for a particular routable object depends on the status of the routable object and the user's privileges. Your Agile API program should take these Workflow dynamics into account and, where possible, adjust your program accordingly.

How the Status of a Change Affects Workflow Functionality

The Workflow actions available for a pending change are different from those for a released change. To check the status of a change to determine whether it's pending or released, use the `IRoutable.getStatus()` method. The `getStatus()` method returns an `IStatus` object for the Workflow status. `IStatus` extends the `INode` interface and provides helpful methods for working with status nodes. The following example shows how to use `getStatus()` to determine whether a change is released.

Example: Getting the status of a change object

```
private static boolean isReleased(IChange change) throws APIException {
    return
    (change.getStatus().getStatusType().equals(StatusConstants.TYPE_RELE
    ASED);
}
```

How User Privileges Affect Workflow Functionality

Agile privileges determine the types of Workflow actions a user can perform on a change. The Agile system administrator assigns roles and privileges to each user. Table below lists privileges needed to perform Workflow actions.

Privilege	Related API
Change Status	IRoutable.changeStatus()
Comment	IRoutable.comment()
Send	DataObject.send()

To determine at run time whether a user has the appropriate privileges to perform an action, use the IUser.hasPrivilege() method. You can adjust your program's UI based on the user's privileges. The following example shows how to check whether a user has the privilege to change the status of a change before calling the IRoutable.changeStatus() method.

Example: Checking the privileges of a user before changing the status of a change

```
private void goToNextStatus(IChange change, IUser user) throws
APIException {
    // Check if the user can change status
    if(user.hasPrivilege(UserConstants.PRIV_CHANGESTATUS, change)) {
        IUser[] approvers = new IUser[] { user };
        IStatus nextStatus = change.getDefaultNextStatus();
        change.changeStatus(nextStatus, true, "", true, true, null,
        approvers, null, false);
    } else {
       System.out.println("Insufficient privileges to change status.");
        }
}
```

Selecting a Workflow

When you create a new change, package, product service request, or quality change order, you must select a Workflow. Otherwise, the object is in an unassigned state and cannot progress through a Workflow process. Your Agile system can have multiple Workflows defined for each type of routable object. To retrieve the valid Workflows for an object, use the IRoutable.getWorkflows() method. If a routable object has not been assigned a Workflow yet, you can use the IRoutable.getWorkflows() method to select a Workflow.

As long as a change is in the Pending status, you can select a different Workflow. Once a change moves beyond the Pending status, you can't change the Workflow.

Example: Selecting a Workflow

```
private IChange createECO(IAgileSession session) throws APIException {
  // Get an Admin instance
  IAdmin admin = session.getAdminInstance();

  // Create a change
  IAgileClass ecoClass =
admin.getAgileClass(ChangeConstants.CLASS_ECO);
  IAutoNumber[] autoNumbersPart = ecoClass.getAutoNumberSources();
  IChange change = (IChange)m_session.createObject(ecoClass,
autoNumbersPart[0]);
  // Get the current Workflow (a null object,
  // since the Workflow has not been set yet)
  IWorkflow wf = change.getWorkflow();
```

```
   // Get all available Workflows
   IWorkflow[] wfs = change.getWorkflows();

    // Set the change to use the first Workflow
    change.setWorkflow(wfs[0]);

    // Set the change to use the second Workflow
    change.setWorkflow(wfs[1]);

    return change;
 }
```

If a change is still in the Pending status type, you can deselect a Workflow to make the change "unassigned." To make a change unassigned, use the `IRoutable.setWorkflow()` method and specify `null` for the Workflow parameter.

Example: Making a change unassigned

```
 private void unassign(IChange change) throws APIException {
   change.setWorkflow(null);
 }
```

Adding and Removing Approvers

After a change is routed and the online approval process is initiated, it is sometimes necessary to add or remove people from the list of approvers or observers. To add or remove approvers or observers, a user must have the Route privilege.

You don't need to load the Workflow table to modify the list of approvers. Once you have a routable object, such as an ECO, you can modify its list of approvers using the `IRoutable.addApprovers()` and `IRoutable.removeApprovers()` methods. When you use `addApprovers()` or `removeApprovers()`, you specify the lists of approvers and observers, whether the Notification is urgent, and an optional comment. The Agile API provides overloaded `addApprovers()` and `removeApprovers()` methods for adding or removing a user or a user group from the list of approvers. For more information, refer to the API Reference files at Oracle® E-Delivery Web site (http://edelivery.oracle.com/).

If any users you select as approvers or observers do not have appropriate privileges to view a change, your program throws an `APIException`. To avoid the possible exception, check the privileges of each user before adding him to the approvers or observers list.

The following example shows how to add and remove approvers for a change.

Example: Adding and removing approvers and observers

```
 public void modifyApprovers(IChange change) {
  try {

    // Get current approvers for the change
    IDataObject[] currApprovers =
change.getApproversEx(change.getStatus());
```

```
    // Get current observers for the change
    IDataObject[] currObservers =
change.getObserversEx(change.getStatus());

    // Add hhawkes to approvers
    IUser user = (IUser)m_session.getObject(IUser.OBJECT_TYPE,
"hhawkes");
    IUser[] approvers = new IUser[]{user};
    // Add flang to observers user =
(IUser)m_session.getObject(IUser.OBJECT_TYPE, "flang");
    IUser[] observers = new IUser[]{user};
    // Add approvers and observers
    change.addApprovers(change.getStatus(), approvers, observers, true,
    "Adding jsmith to approvers and jdoe to observers");
    // Add skubrick to approvers user =
(IUser)m_session.getObject(IUser.OBJECT_TYPE, "skubrick"); approvers[0]
= user;
    // Add kwong to observers user =
(IUser)m_session.getObject(IUser.OBJECT_TYPE, "kwong"); observers[0] =
user;
    // Remove skubrick from approvers and kwong from observers
    change.removeApprovers(change.getStatus(), approvers, observers,
    "Removing skubrick from approvers and kwong from observers");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

If you want to modify only the list of approvers or the list of observers for a change, you can pass a `null` value for the parameter you don't want to change. The following example shows how to add the current user to the approvers list without changing the list of observers.

Example: Adding approvers without changing observers

```
public void addMeToApprovers(IChange change) {
  try {
    // Get the current user
    IUser user = m_session.getCurrentUser();

    // Add the current user to the approvers list for the change
    IUser[] approvers = new IUser[]{user};
    change.addApprovers(change.getStatus(), approvers, null, true,
    "Adding current user to approvers list");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Setting the "Signoff User Dual Identification" Preference

The "*Signoff User Dual Identification*" feature is a systemwide preference that controls whether approval/rejection signoff requires a dual identification, or a "second signoff." This feature is required by FDA-regulated companies and can benefit companies with a corporate policy requiring double authentication of user identity when approving or rejecting change orders. For more information, refer to the latest release of the *Agile PLM Administrator Guide*.

The following paragraphs list and describe the APIs that support the Signoff User Dual Identification feature and provide code samples that use these methods.

Approving a Routable Object

This method informs users the object is approved by the approver, or when the approver is approving the object on behalf of one or more user groups. You can also use this method to specify the `secondSignature`, `escalations`, `transfers`, or `signoffForSelf` parameters as they are set in server's Preferences settings.

```
// Approving a user
/*
Parameters:
  password: User's approval password
  secondSignature: User's second signature for approval
  comment: A character string for user comments (4000 characters
maximum)
  notifyList: List of users and user groups to notify
  approveForGroupList: List of user groups to approve for
  escalations: Escalated from other users and user groups to approve
for
  transfers: From other users and user groups to approve for
  signoffForSelf: True to signoff for self and False otherwise
APIException:
Thrown if the method fails to approve the routable object
*/
public void approve (String password, String secondSignature,
            String comment, Collection notifyList,
            Collection approveForGroupList, Collection escalations,
            Collection transfers, boolean signoffForSelf)
        throws APIException;
```

The following code example approves a routing object requiring Dual identification. Other conditions are:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification is selected.
- Set the Workflow Settings for *Workflow Status CCB: Default Change Orders Dual Identification Required* to *Yes*.
- Create a *change object an*d add a user, for example **`admin`** as approver to the CCB Status and the Released Status.

Example: Approving a routable object

```
// Admin approves the change by supplying approval password and "User
// ID" as the second signature
IWorkflow [] wfs = chg.getWorkflows();
IWorkflow wf = wfs[0];
chg.setWorkflow(wf);

// Advance ECO to submitted state
IStatus [] sts;
sts = wf.getStates(StatusConstants.TYPE_SUBMIT);
IStatus submit = sts[0];

// Change status to submit
m_session.disableWarning(new Integer(553));
m_session.disableWarning(new Integer(574));

Object [] nullObjectList = null;
chg.changeStatus(submit, false, null, false, false, nullObjectList,
nullObjectList, nullObjectList, false);
// Add approvers to CCB status and to Released status
IUser usr = (IUser) m_session.getObject(IUser.OBJECT_TYPE, "admin");
Object [] apprList = new IUser [] {usr};
sts = wf.getStates(StatusConstants.TYPE_REVIEW);
IStatus ccb = sts[0];
chg.addApprovers(ccb, apprList, nullObjectList, false,
"ADDAPPROVER_OBSERVER");
//   Change it to CCB status
chg.changeStatus(ccb, false, null, false, false, nullObjectList,
nullObjectList, nullObjectList, false);
m_session.enableWarning(new Integer(553));
m_session.enableWarning(new Integer(574));

// Admin approves the change by supplying approval password and "User
// ID" as the second signature
String userName = session.getCurrentUser().getName();
chg.approve ("agile", userName, "OK, Approved", null, null, null, null,
true);
```

Rejecting a Routable Object

This method informs users that the routable object is rejected by the approver, or when the approver is rejecting the object on behalf of one or more user groups. You can also use this method to specify the `secondSignature`, `escalations`, `transfers`, or `SignoffForSelf` parameters as they are set in server's Preferences settings.

```
// Rejecting a user
/*
Parameters
```

```
    password: User's approval password
    secondSignature: User's second signature for approval
    comment: A character string for user comments
       (4000 characters maximum)
    notifyList: List of users and user groups to notify
    approveForGroupList: List of user groups to approve for
    escalations: Escalated from other users and user groups
        to approve for
    transfers: From other users and user groups to approve for
    signoffForSelf: True to signoff for self and False otherwise
APIException
    Thrown if the method fails to approve the routable object
*/
public void reject(String password, String secondSignature,
    String comment, Collection notifyList,
    Collection approveForGroupList, Collection escalations,
    Collection transfers, boolean signoffForSelf)
throws APIException;
```

The following code sample requires Dual identification to reject a routing object. Other conditions are:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification is selected.
- Set the Workflow Settings for Workflow Status CCB: Default Change Orders Dual Identification Required to Yes.
- Create a *change object an*d add a user, for example **admin** as an approver of the CCB Status and the Released Status.

Example: Rejecting a routable object

```
// Admin rejects the change by supplying approval password and "User
// ID" as the second signature
IChange chg;
String chgNo;

IUser curUser = session.getCurrentUser();
String userName = curUser.getName();

chg = (IChange) session.getObject(ChangeConstants.CLASS_ECO, chgNo);
chg.reject("agile", userName, "Rejected", null, null, null, null,
true);
```

Adding User Groups of Approvers and Users to Approve Routable Objects

This method is designed to retrieve an array of user groups that is added as approvers for a particular Workflow status and the current user/approver is also a group member.

```
/*
Parameters
    status: A node corresponding to the desired Workflow status. You can
    retrieve the current status using getStatus(). To retrieve the
    default next status, use getDefaultNextStatus().
APIExceptions and Returns
```

- throws APIException if the method fails

- returns an array of IUserGroup objects

```
*/
public IDataObject[] getPossibleUserGroupsForSignoff(IStatus status)
        throws APIException;
```

The following code sample adds a user group that approves the ECO at the CCB status and contains the current user as its member. In addition to Dual identification, the following conditions are also required:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification is selected.
- Set the *Workflow Settings* for Workflow Status CCB: Default Change Orders Dual Identification Required to Yes.

Example: Adding User Groups of Approvers with Current User as a Member of the "User Group"

```
// IChange change;
//Set Workflow
IWorkflow[] wfs=change.getWorkflows();
IWorkflow workflow = wfs[0];
change.setWorkflow(workflow);

//Add the User Group as approver for CCB
Object[] appr=new Object[] {user_group};
IStatus current=change.getStatus();
StatusConstants type=current.getStatusType();

m_session.disableWarning(new Integer(574));
while(!(type == (StatusConstants.TYPE_REVIEW))){
      IStatus nextstatus=change.getDefaultNextStatus();
      change.changeStatus(nextstatus,true,"",true,true,(Object[])null,(
Object[])null,             (Object[])null,false);
      current=change.getStatus();
      type=change.getStatus().getStatusType();
}
m_session.enableWarning(new Integer(574));
change.addApprovers(current, appr, (Object [])null, false, "");
IDataObject[] u = change.getPossibleUserGroupsForSignoff(status);
/* APPROVE */
Collection gl = new ArrayList();

//Group list
gl.add(u[0]);
change.approve("agile", session.getCurrentUser().getName(),
     "ESIGN-FIRST", null, gl, null, null, false);
```

Approving a Routable Object by Users on behalf of "Transferred from Users"

This method is designed to retrieve an array of users that is added as transfer authorities for the current user and for a particular Workflow status.

```
/*
Parameters
    status: A node corresponding to the desired Workflow status. You can
    retrieve the current status using getStatus(). To retrieve the
    default next status, use getDefaultNextStatus().
APIExceptions and Returns
    - throws APIException if the method fails
    - returns an array of IUserGroup objects
*/
public IDataObject[] getPossibleTransferredFromUsers(IStatus status)
        throws APIException;
```

The following code sample set up a Transfer Authority from user A to another user B, creates an ECO, and adds user A as an approver of the CCB status.

Note The `getPossibleTransferredFromUsers(IStatus status)` return value for user B 's CCB status is the array of Users whose Sign-off authority is transferred to user B.

In addition to Dual identification, the following conditions are also required:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification is selected.
- Set the *Workflow Settings* for Workflow Status CCB: Default Change Orders Dual Identification Required to Yes.

Example: Setting up a Transfer of Authority from one user to another user

```
Log in and execute the following code as User B
IDataObject [] usrs = chg.getPossibleTransferredFromUsers(status);
// Prepare the collection
Collection col = new ArrayList ();

for (int i=0; i < usrs.length; i++){
col.add(usrs[i]);
}

// approve the change
chg.approve("agile", userName, "OK, Approved", null, null, null, col,
false);
```

Adding Active Escalations for the Current User to Approve a Routable Object

This method is designed to retrieve an array of users that serve as active escalations for the current user for a particular Workflow status. This method will override the settings in the *Allow Escalation Designation Approval* attribute on the user's cover page.

```
/*
```

```
Parameters
    status: A node corresponding to the desired Workflowstatus. You can
    retrieve the current status using getStatus(). To retrieve the
    default next status, use getDefaultNextStatus().
APIExceptions and Returns
    - throws APIException if the method fails
    - returns an array of IUser and IUserGroup objects
*/
public IDataObject[] getPossibleEscalatedFromUsers(IStatus status)
        throws APIException;
```

The following code sample sets up an Escalation from user A to user, B creates an ECO, and adds user A as an approver of the CCB status.

Note The `getPossibleEscalatedFromUsers(IStatus status)` for user B for the CCB status will return the array of Users whose escalations are set to user B.

In addition to Dual identification, the following conditions are also required:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification is selected.
- Set the *Workflow Settings* for Workflow Status CCB: Default Change Orders Dual Identification Required to Yes.

Example: Setting up an escalation for a user

```
// Log in and execute the following code as "User B"
IDataObject [] usrs = chg. getPossibleEscalatedFromUsers(IStatus
status);
// Prepare the collection
Collection col = new ArrayList ();

for (int i=0; i < usrs.length; i++){
     col.add(usrs[i]);
}

// approve the change
chg.approve("agile", userName, "OK, Approved", null, null, null, col,
false);
```

Specifying a Second Signature to Approve a Routable Object

This method is designed to verify if a second signature is required to approve a routable object. Use this with methods in combination with the methods documented in Adding User ID as Second Signature to Approve a Routable Object on page 215 which also has the `secondSignature` parameter and Approving a Routable Object on page 208 and Rejecting a Routable Object on page 209.

Examples

```
/*
Parameters
    Status: The status (IStatus) of the object checked for the next
    Workflowstatus.
```

```
Returns
    true if a second signature is required, false otherwise
*/
public boolean isSecondSignatureRequired(IStatus status)
        throws APIException;
```

The following code sample creates an ECO `chg` (change order). The `chg.isSecondSignatureRequired (IStatus status)` for the CCB status will return `true`.

In addition to Dual identification, the following conditions are also required:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification is selected.
- Set the *Workflow Settings* for Workflow Status CCB: Default Change Orders Dual Identification Required to Yes.

Example: Specifying a second signature to approve a routable object

```
// set the "Signoff User Dual Identification Type" preferences Node
IAdmin admin = session.getAdminInstance();
INode node = admin.getNode(NodeConstants.NODE_PREFERENCES);
// Node Properties
IProperty propSecondSignature = node.getProperty("Signoff User Dual
Identification Type");
IAgileList lst = propSecondSignature.getAvailableValues();
lst.setSelection(new Object [] {"User ID"});
propSecondSignature.setValue(lst);

// set the "Dual Identification Required" property for "Workflow Status
CCB: Default Change Orders" to "Yes"
IAdmin admin = session.getAdminInstance();
INode root=admin.getNode(NodeConstants.NODE_AGILE_WORKFLOWS);
INode CCBStatus=(INode)root.getChildNode("Default Change Orders/Status
List/CCB");
IProperty propDualIdentification = CCBStatus.getProperty("Dual
Identification Required");
IAgileList lst = propDualIdentification.getAvailableValues();
lst.setSelection(new Object [] {"Yes"});
propDualIdentification.setValue(lst);

// Get and print the "Second Signature Required Property" for the
// various states of a workflow
IWorkflow [] wfs = chg.getWorkflows();
IWorkflow wf = wfs[0];
chg.setWorkflow(wf);
IStatus [] sts = wf.getStates();
boolean secondSigReqd;

for (int i=0; i< sts.length; i++) {
    IStatus st = sts[i];
    System.out.println("Status Name =" + st.getName());
    System.out.println("IS Second Signature Reqd = "+
          chg.isSecondSignatureRequired(st));
```

```
    System.out.println("IS Second Signature UserId = " +
          chg.isSecondSignatureUserId(st));

    secondSigReqd = chg.isSecondSignatureRequired(st);
}
```

Adding User ID as Second Signature to Approve a Routable Object

This method is designed to set the user's ID as the second signature to approve a routable object. Use this method in combination with methods documented in Specifying a Second Signature to Approve a Routable Object on page 213. These methods have the `secondSignature` parameter. See Approving a Routable Object on page 208 and Rejecting a Routable Object on page 209.

```
/*
Parameters
    Status: The status (IStatus) of the object checked for the next
    Workflow status.
Returns
    true if a second signature required is User ID, false otherwise
*/
public boolean isSecondSignatureUserId(IStatus status)
        throws APIException;
```

The following code sample creates an ECO `chg` (change order). The `chg.isSecondSignatureRequired (IStatus status)` for the CCB status will return `true`.

In addition to Dual identification, the following conditions are also required:

- Display User ID when Sever Settings > Preferences Signoff User Dual Identification Type is selected.
- Set the *Workflow Settings* for Workflow Status CCB: Default Change Orders Dual Identification Required to Yes.

Example: Speechifying User ID as the second signature

```
boolean secondSigUserID;
secondSigUserID = chg.isSecondSignatureUserId (status);
```

Approving or Rejecting Change

After a change is routed to a group of approvers, the online approval process begins. Users listed in the Workflow table for a change can approve or reject the change.

When you approve a change, the Agile system records the approval on the Workflow table. When all approvers have approved the change, the system sends an email Notification to the change analyst or component engineer indicating that the change is ready to be released.

Note	To approve or reject a change, users must have either a Create And Manage or Request And Approve privileges. For more information, refer to *Agile PLM Administrator Guide*.

When you use the `IRoutable.approve()` method, you specify the user's approval password and an optional comment. Overloaded `approve()` methods allow you to specify a Notification list and a collection of user groups for which you're approving; refer to the API Reference files at Oracle® E-Delivery Web site (http://edelivery.oracle.com/) for details.

The following paragraphs documents approving or rejecting a given routable object. The APIs that support approving or rejecting a PC change object when a second signature is required are described in detail in Setting the “Signoff User Dual Identification” Preference on page 208

The following example shows how to approve a change.

Example: Approving a change

```
public void approveChange(IChange change) {
  try {
    change.approve("agile", "Looks good to me");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

If a change has a fundamental flaw, users listed on the Workflow table may reject it. When you reject a change, the system records the rejection on the Workflow tab for the change and sends an email Notification to the change analyst or component engineer. The change analyst or component engineer may decide to return the rejected change to the originator, thus reverting its status to Pending.

When you use the `IRoutable.reject()` method, you must specify the userNotifications approval password and optional comments. An overloaded `reject()` method allows you to specify a Notification list and a collection of user groups for which you’re approving; refer to the API Reference files at Oracle® E-Delivery Web site (http://edelivery.oracle.com/) for more information.

The following example shows how to reject a change.

Example: Rejecting a change

```
public void rejectChange(IChange change) {
  try {
    change.reject("agile", "Incorrect replacement part!");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Commenting a Change

When you comment a change, you send a comment to other CCB reviewers during the online approval process. In addition to the comment, you can specify whether to notify the originator, the change analyst, and the change control board. An overloaded `comment()` method allows you to specify a Notification list. For more information, refer to the API Reference files at Oracle® E-Delivery Web site (http://edelivery.oracle.com/).

The following example shows how to comment a change.

Example: Commenting a change

```
public void commentChange(IChange change) {
  try {
    change.comment(true, true, true, "Change flagged for transfer to
ERP.");
```

```
    } catch (APIException ex) {
      System.out.println(ex);
    }
  }
```

Auditing a Change

At any point in a change's lifecycle you can audit it to determine if any required entry cells are not completed or if the change violates any Agile SmartRules. When you use the `IRoutable.audit()` method, the method returns a `Map` object containing `ICell` objects as keys and a `List` of `APIException` objects as values. The `ICell` key can be null if there are no problems with the change. The `APIException` object describes a problem with the associated entry cell.

The `Map` object returned by the `audit()` method may also contain null objects as keys. The `APIException` object associated with a null object describes a problem unrelated to data cells.

The following example shows how to audit a change.

Example: Auditing a change

```
public void auditChange(IChange change) {
  try {
    // Audit the release
    Map results = change.audit();

    // Get the set view of the map
    Set set = results.entrySet();

    // Get an iterator for the set
    Iterator it = set.iterator();

    // Iterate through the cells and print each cell name and exception
    while (it.hasNext()) {
      Map.Entry entry = (Map.Entry)it.next();
      ICell cell = (ICell)entry.getKey();
      if(cell != null) {
        System.out.println("Cell : " + cell.getName());
      } else {
        System.out.println("Cell : No associated data cell");
      }
      //Iterate through exceptions for each map entry.
      //(There can be multiple exceptions for each data cell.)
      Iterator jt = ((Collection)entry.getValue()).iterator();
      while (jt.hasNext()) {
        APIException e = (APIException)jt.next();
        System.out.println("Exception : " + e.getMessage());
      }
    }
```

```
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Changing the Workflow Status of an Object

The `IRouteable.changeStatus()` method is a general purpose method for changing the status of an Agile object. For example, you can use `changeStatus()` to submit, release, or cancel a change. In instances such as failed audits, it throws the compound exception `ExceptionConstants.API_SEE_MULTIPLE_ROOT_CAUSES`. You can disable this exception by modifying the code that caught the exception. See the example below.

Example: Throwing compound exception s

```
while (true) {
    try {
        change.changeStatus(
    wf.getStates(expectStatus)[0],
    false,
    "comment",
    false,
    false,
    null,
    null,
    null,
    false
        );
    } catch (APIException ae) {
      try {
         if
(ae.getErrorCode().equals(ExceptionConstants.API_SEE_MULTIPLE_ROOT_CAUS
ES)){
           Throwable[] causes = ae.getRootCauses();
            for (int i = 0; i < causes.length; i++) {
             m_session.disableWarning(
               (Integer)((APIException)causes[i]).getErrorCode()
                   );
              }
             } else {
               m_session.disableWarning((Integer)ae.getErrorCode());
             }
         } catch (Exception e) {
         throw ae;
         }
         continue;
    }
    break;
}
```

In general, you release a change after it is signed off by CCB members. In addition to modifying the

status of a change, you can also use `changeStatus()` to specify a Notification list, optional comments, and whether to notify the originator and change control board.

Depending on the overloaded `changeStatus()` method you use, the `notifyList` parameter is an array of `IUser` or `IUserGroup` objects that should be notified about the change in status; refer to the API Reference files at Oracle® E-Delivery Web site (http://edelivery.oracle.com/).for details. To use the default Notification list for the Workflow status, specify a `null` value. To indicate that no users should be notified, specify an empty array.

For both the `approvers` and `observers` parameters of the `changeStatus()` method, you must explicitly pass an array of users or user groups. If you pass `null`, no approvers or observers are used. To get the default approvers and observers for a particular Workflow status, use `getApproversEx()` and `getObserversEx()`, respectively.

The following example shows how to check the Workflow status of a change.

Example: Checking the status of a change

```
void checkStatus(IChange change) {
  try {
    // Get current workflow status (an IStatus object)
    IStatus status = change.getStatus();
    System.out.println("Status name = " + status.getName());
    // Get next available Workflow statuses
    IStatus[] nextStatuses = change.getNextStatuses();
    for (int i = 0; i < nextStatuses.length; i++) {
      System.out.println("nextStatuses[" + i +"] = " +
      nextStatuses[i].getName());
    }
    // Get next default Workflow status
    IStatus nextDefStatus = change.getDefaultNextStatus();
    System.out.println("Next default status = " +
nextDefStatus.getName());
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

The following example shows how to change the status of a change.

Example: Changing the status of a change

```
public void nextStatus(IChange change, IUser[] notifyList,
  IUser[] approvers, IUser[] observers) {
  try {
    // Check if the user has privileges to change to the next status
    IStatus nextStatus = change.getDefaultNextStatus();
    if (nextStatus == null) {
      System.out.println("Insufficient privileges to change status.");
      return;
    }
    // Change to the next status
    else {
```

```
      change.changeStatus(nextStatus, true, "", true, true, notifyList,
      approvers, observers, false);
    }
  } catch (APIException ex) {
   System.out.println(ex);
  }
}
```

The following example shows how to use the default approvers and observers when you change the status of a routable object.

Example: Changing the status and routing to the default approvers and observers

```
public void changeToDefaultNextStatus(IChange change) throws
APIException {
  // Get the next status of the change
  IStatus nextStatus = change.getDefaultNextStatus();

  // Get default approvers for the next status
  IDataObject[] defaultApprovers = change.getApproversEx(nextStatus);
  // Get default observers for the next status
  IDataObject[] defaultObservers = change.getObserversEx(nextStatus);
  // Change to the next status
  change.changeStatus(nextStatus, false, "", false, false, null,
defaultApprovers,
     defaultObservers, false);

}
```

Sending an Agile Object to Selected Users

You can send any Agile object to a selected group of users. When you send an object, such as an ECO, there is no signoff required. The selected recipients receive an email message with an attached link to the object. When you use the `IDataObject.send()` method, you can specify an array of Agile users and an optional comment. Unlike other Workflow commands, the `send()` method is not limited to routable objects. You can use it to send any type of Agile dataobject, including an item.

The following example shows how to send an object to all users.

Example: Sending an Agile object to selected users

```
public void sendToAll(IDataObject object) {
  try {
    // Get all users
    IQuery q = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
"select * from [Users]");
    ArrayList userList = new ArrayList();
    Iterator i = q.execute().getReferentIterator();
    while (i.hasNext()) {
      userList.add(i.next());
```

```
    }
    IUser[] users = new IUser[userList.size()];
    System.arraycopy(userList.toArray(), 0, users, 0, userList.size());
    // Send the object to all users
    object.send(users, "Please read this important document.");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Sending an Agile Object to User Groups

You can send an Agile change object or an item object to a user group. When you send an object, such as an ECO, there is no signoff required. The selected recipients receive an email message with an attached link to the object. When you use the `IDataObject.send(IDataObject[] to, String Comment)` method, you can specify an array of Agile User Groups and an optional comment. The `IDataObject` parent interface represents the `IUserGroup` Agile object. Unlike other Workflow commands, the `send()` method is not limited to routable objects. You can use it to send any type of Agile dataobject, including an item.

The following example shows how to send an object to all User Groups.

Example: Sending an Agile object to selected user groups

```
public void sendToAll(IDataObject[] object) {
  try {
    // Get all user groups
    IQuery q = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
"select * from [UserGroup]");
    ArrayList userList = new ArrayList();
    Iterator i = q.execute().getReferentIterator();
    while (i.hasNext()) {
      usergroupList.add(i.next());
    }
    IUserGroup[] group = (IUserGroup[])(usergroupList.toArray());
    // Send the object to all user groups
    object.send(usergroups, "Please read this important document.");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Chapter 15

Managing and Tracking Quality

This chapter includes the following:

- About Quality Control 223
- Working with Customers 224
- Working with Product Service Requests 226
- Working with Quality Change Requests 229
- Using Workflow Features with PSRs and QCRs 231

About Quality Control

The Agile PLM system provides tools that allow companies to track and manage the following quality-related items:

- customer complaints
- product and manufacturing quality issues
- enhancement and corrective action requests

The corrective action process in the Agile PLM system is flexible and can be implemented in many different ways. For example, one way to customize the Agile PLM system is to use the Agile API to integrate the system with a Customer Relationship Management (CRM) system.

Quality-Related API Objects

The Agile API includes the following new interfaces:

- `ICustomer` – interface for the `Customer` class. A customer is anyone that uses a company's product(s). In some Agile PLM implementations, customers and problem reports will be imported directly from Customer Relationship Management (CRM) systems.
- `IServiceRequest` – interface for the `ServiceRequest` class. `IServiceRequest` is a subinterface of `IRoutable`; it lets you create two types of service requests, problem reports and nonconformance reports (NCRs).
- `IQualityChangeRequest` – interface for the `QualityChangeRequest` class, which is similar to an ECR and other types of change requests. It represents a closed loop Workflow process that addresses quality problems. Audit and CAPA (Corrective Action/Preventive Action) are subclasses of `QualityChangeRequest`.

Quality-Related Roles and Privileges

To create, view, and modify problem reports, issues, NCRs, CAPAs, and QCRs, you must have the appropriate privileges. The Agile PLM system has two default user roles that provide users with privileges to work with these quality-related objects:

- Quality Analyst – role for users who manage problem reports, issues, and NCRs.
- Quality Administrator – role for users who manage audits and CAPAs.

For more information about roles and privileges, refer to the *Agile PLM Administrator Guide*.

Working with Customers

This section describes how to create, load, and save `ICustomer` objects.

About Customers

The `ICustomer` object stores contact information about a customer. What role does a customer have in the Agile PLM system? Customers provide feedback on your company's products, alerting you to quality issues or problems they encounter.

This object can originate in another system, such as a CRM system. You can use the Agile API to import customer data and problem reports from CRM systems into the Agile PLM system.

Creating a Customer

To create a customer, use the `IAgileSession.createObject()` method. At a minimum, you should specify values for the `General Info.Customer Name` and `General Info.Customer Number` attributes.

Example: Creating a customer

```
try {
  //Create a Map object to store parameters
  Map params = new HashMap();

  //Initialize the params object
  params.put(CustomerConstants.ATT_GENERAL_INFO_CUSTOMER_NUMBER,
"CUST00006");
  params.put(CustomerConstants.ATT_GENERAL_INFO_CUSTOMER_NAME, "Western
Widgets");
  //Create a new customer
  ICustomer cust1 =
(ICustomer)m_session.createObject(CustomerConstants.CLASS_CUSTOMER,
params);
  } catch (APIException ex) {
    System.out.println(ex);
  }
```

Loading a Customer

To load a customer, use the `IAgileSession.getObject()` method. To uniquely identify a customer, specify the value for the General Info | Customer Number attribute.

Example: Loading a customer

```
try {
  // Load a customer by specifying a CustomerNumber
  ICustomer cust =
(ICustomer)m_session.getObject(ICustomer.OBJECT_TYPE, "CUST00006");
} catch (APIException ex) {
  System.out.println(ex);
}
```

Saving a Customer as Another Customer

To save a customer as another customer, use the `IDataObject.saveAs()` method, which has the following syntax:

```
public IAgileObject saveAs(java.lang.Object type, java.lang.Object
params)
```

For the *params* parameter, specify the General Info | Customer Name and General Info | Customer Number attributes.

Example: Saving a customer to another customer

```
try {
  // Load an existing customer
  ICustomer cust1 =
(ICustomer)m_session.getObject(ICustomer.OBJECT_TYPE, "CUST00006");
  //Create a Map object to store parameters
  Map params = new HashMap();

  //Initialize the params object
  params.put(CustomerConstants.ATT_GENERAL_INFO_CUSTOMER_NUMBER,
"CUST00007");
  params.put(CustomerConstants.ATT_GENERAL_INFO_CUSTOMER_NAME, "Wang
Widgets");
  // Save the customer
  ICustomer cust2 =
(ICustomer)cust1.saveAs(CustomerConstants.CLASS_CUSTOMER, params);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Working with Product Service Requests

This section describes how to work with the two classes of Product Service Requests, Problem Reports and Nonconformance Reports.

About Problem Reports

A problem report describes a problem or an issue that occurred with a product from the customer's perspective. A problem report can be submitted by a customer, sales representative, or customer service representative.

Because a problem report usually originates with a customer, it may not accurately describe the actual cause of the problem. To understand the root cause of a problem, a Quality Analyst must investigate the problem.

Problem reports can be routed for investigation. The investigating team, consisting of Quality Analysts, determines the root cause of the problem and decides whether to escalate the problem into an issue.

About Nonconformance Reports

A nonconformance report (NCR) is used to report material damages, failure modes, or defects in a product received by a customer or supplier. An NCR is typically identified when a product shipment is inspected after receipt from a supplier. A product is nonconforming if it does not meet customer requirements or specifications. Such products are generally rejected or segregated to await disposition. A nonconformance report may require that a Quality Analyst investigate the problem and determine whether corrective action is required.

NCRs can be routed for review. Typically, the review is used for additional information gathering rather than approval and rejection.

Creating a Product Service Request

To create a problem report or nonconformance report, use the `IAgileSession.createObject()` method. The only required attribute value you must specify is the object's number. Example 12-4 shows how to create problem reports and NCRs.

Example: Creating a problem report or NCR

```
public IServiceRequest createPR(String strNum) throws APIException {
  IServiceRequest pr = (IServiceRequest)m_session.createObject(
    ServiceRequestConstants.CLASS_PROBLEM_REPORT, strNum);
  return pr;
}
public IServiceRequest createNCR(String strNum) throws APIException {
  IServiceRequest ncr = (IServiceRequest)m_session.createObject(
    ServiceRequestConstants.CLASS_NCR, strNum);
  return ncr;
}
```

Assigning a Product Service Request to a Quality Analyst

To assign a problem report or NCR to a Quality Analyst, set the value for the Cover Page | Quality Analyst field, which is a list field. The available values for the list field consists of Agile PLM users. The following example shows how to set the value for the `Cover Page.Quality Analyst` field for a problem report or NCR.

Example: Assigning a problem report or nonconformance report

```
void assignServiceRequest(IServiceRequest sr) throws APIException {
  Integer attrID;
  //Set attrID equal to the Quality Analyst attribute ID attrID =
ServiceRequestConstants.ATT_COVER_PAGE_QUALITY_ANALYST;
  //Get the Cover Page.Quality Analyst cell
  ICell cell = sr.getCell(attrID);

  //Get available list values for the list
  IAgileList values = cell.getAvailableValues();

  //Set the value to the current user
  IUser user = m_session.getCurrentUser();
  values.setSelection(new Object[] { user });
  cell.setValue(values);
}
```

Adding Affected Items to a Product Service Request

To associate a problem report or nonconformance report with one or more items, you add items to the Affected Items table. Each Product Service Request can be associated with many items.

Note If Product Service Requests have been added to the Related PSR table, the Affected Items table cannot be modified.

Example: Adding an affected item to a Product Service Request

```
void addAffectedItem(IServiceRequest sr, String strItemNum) throws
APIException {
  //Get the class
  IAgileClass cls = sr.getAgileClass();

  //Attribute variable
  IAttribute attr = null;

  //Get the Affected Items table
  ITable affItems =
sr.getTable(ServiceRequestConstants.TABLE_AFFECTEDITEMS);
  //Create a HashMap to store parameters
  HashMap params = new HashMap();

  //Set the Item Number value
```

```
  params.put(ServiceRequestConstants.ATT_AFFECTED_ITEMS_ITEM_NUMBER,
strItemNum);
  //Set the Latest Change value
  attr =
cls.getAttribute(ServiceRequestConstants.ATT_AFFECTED_ITEMS_LATEST_CHAN
GE);
  IAgileList listvalues = attr.getAvailableValues();
  listvalues.setSelection(new Object[] { new Integer(0)});
  params.put(ServiceRequestConstants.ATT_AFFECTED_ITEMS_LATEST_CHANGE,
listvalues);
  //Set the Affected Site value
  attr =
cls.getAttribute(ServiceRequestConstants.ATT_AFFECTED_ITEMS_AFFECTED_SI
TE);
  IAgileList listvalues = attr.getAvailableValues();
  listvalues.setSelection((new Object[] { "Hong Kong" });
  params.put(ServiceRequestConstants.ATT_AFFECTED_ITEMS_AFFECTED_SITE,
listvalues);
  //Create a new row in the Affected Items table
  IRow row = affItems.createRow(params);
}
```

Adding Related PSRs to a Product Service Request

A Product Service Request can be used to aggregate multiple problem reports or NCRs into one master. To do this, create a new Product Service Request and don't add items to the Affected Items table. Instead, select the Related PSR table and add a row for each related Product Service Request.

Note If items have been added to the Affected Items table, the Related PSR table cannot be modified.

Example: Adding related PSRs to a Product Service Request

```
void addRelatedPSRs(IServiceRequest sr, String[] psrNum) throws
APIException {
  //Get the Related PSR table
  ITable relPSR =
sr.getTable(ServiceRequestConstants.TABLE_RELATEDPSR);
  //Create a HashMap to store parameters
  HashMap params = new HashMap();

  //Add PSRs to the Related PSR table
  for (int i = 0; i < psrNum.length; i++)
  {
    //Set the PSR Number value
    params.put(ServiceRequestConstants.ATT_RELATED_PSR_PSR_NUMBER,
psrNum[i]);
    //Create a new row in the Related PSR table
    IRow row = relPSR.createRow(params);
```

```
    //Reset parameters
    params = null;
  }
}
```

Working with Quality Change Requests

A Quality Change Request, or QCR, allows a Quality Analyst to manage quality records that contain aggregated problems related to products, documents, suppliers, and customers. You can route the QCR for review and approval, driving the issue(s) to closure using corrective or preventive action. This may result in changes to a product, process, or supplier by initiating an ECO or MCO. QCRs also provide an audit trail between problems, corrective and preventive actions, and engineering changes.

Agile PLM provides two classes of Quality Change Requests:

- CAPA – Stands for Corrective Action/Preventive Action, which addresses defects that (generally) surfaced from problem reports. By the time a problem reaches the CAPA stage, the team has figured out which specific items must be fixed. Consequently, the affected item for a CAPA may be different from the affected item of its related problem report. For example, say a customer reported a problem with a DVD-ROM drive. A CAPA is initiated and the root-cause is identified to be a defect in the IDE controller. Therefore, the CAPA and its related problem report have different affected items.
- Audit – Systematic, independent and documented processes for obtaining evidence and evaluating it objectively to determine the extent to which criteria are fulfilled. Audits can be initiated against items for which no problems have been reported.

Creating a Quality Change Request

To create a QCR, use the `IAgileSession.createObject()` method. The only required attribute value you must specify is the object's number. Example below shows how to create both CAPA and Audit QCRs.

Example: Creating a QCR

```
public IQualityChangeRequest createCAPA(String strNum) throws
APIException {
  IQualityChangeRequest capa =
(IQualityChangeRequest)m_session.createObject(
    QualityChangeRequestConstants.CLASS_CAPA, strNum);
  return capa;

}
public IQualityChangeRequest createAudit(String strNum) throws
APIException {
  IQualityChangeRequest audit =
(IQualityChangeRequest)m_session.createObject(
    QualityChangeRequestConstants.CLASS_AUDIT, strNum);
  return audit;
```

```
}
```

Assigning a Quality Change Request to a Quality Administrator

To assign a QCR to a Quality Administrator, you set the value for the Cover Page | Quality Administrator field. This process is similar to the way you assign a Product Service Request to a Quality Analyst.

Example: Assigning a QCR

```
void assignQCR(IQualityChangeRequest qcr) throws APIException {
  Integer attrID;
  //Set attrID equal to the Quality Administrator attribute ID
  attrID =
QualityChangeRequestConstants.ATT_COVER_PAGE_QUALITY_ADMINISTRATOR;
  //Get the Cover Page.Quality Administrator cell
  ICell cell = qcr.getCell(attrID);

  //Get available list values for the list
  IAgileList values = cell.getAvailableValues();

  //Set the value to the current user
  IUser user = m_session.getCurrentUser();
  values.setSelection(new Object[] { user });
  cell.setValue(values);
}
```

Saving a Quality Change Request as a Change

You can use the `IDataObject.saveAs()` method to save a QCR as another QCR or as an ECO (or another type of change order). When you save a QCR as an ECO, the items affected by the QCR are not automatically transferred to the Affected Items tab of the ECO. If you want to transfer affected items from the QCR to the ECO, you must write the code in your program to provide that functionality. Workflow is a required input parameter for using `saveAs()` on QCRs.

Note If you try to save a QCR to an object that is not a subclass of either the Quality Change Request or Change superclasses, the Agile API throws an exception.

Example: Saving a QCR as an ECO

```
public IChange saveQCRasECO(IAgileSession session,
IQualityChangeRequest qcr) throws APIException {
// Get the ECO class
    IAgileClass cls = m_admin.getAgileClass(ChangeConstants.CLASS_ECO);
// Get autonumber sources for the ECO class
  IAutoNumber[] numbers = cls.getAutoNumberSources();
// Get Workflow for the ECO class
    IWorkflow ecoWf =
    ((IRoutableDesc)session.getAdminInstance().getAgileClass(ChangeConst
    ants.CLASS_ECO)).getWorkflows()[0];
// Save the QCR as an ECO
    HashMap map = new HashMap();
```

```
     map.put(ChangeConstants.ATT_COVER_PAGE_NUMBER, numbers[0]);
     map.put(ChangeConstants.ATT_COVER_PAGE_WORKFLOW, ecoWf);
    IChange eco = (IChange)qcr.saveAs(ChangeConstants.CLASS_ECO, map);
// Add code here to copy affected items from the QCR to the ECO
    return eco;
}
```

Using Workflow Features with PSRs and QCRs

PSRs and QCRs derive all Workflow functionality from the `IRoutable` interface. The following table lists the Workflow commands you can use to manage product quality objects.

Feature	Equivalent API(s)
Audit a PSR or QCR	`IRoutable.audit()`
Change the status of a PSR or QCR	`IRoutable.changeStatus()`
Send a PSR or QCR to another user	`IDataObject.send()`
Approve a PSR or QCR	`IRoutable.approve()`
Reject a PSR or QCR	`IRoutable.reject()`
Comment on a PSR or QCR	`IRoutable.comment()`
Add or remove approvers for a PSR or QCR	`IRoutable.addApprovers()` `IRoutable.removeApprovers()`

Selecting Workflows for PSRs and QCRs

When you create a new Product Service Request or a Quality Change Request, you must select a workflow. Your Agile PLM system can have multiple workflows defined for each type of Product Service Request and Quality Change Request. To retrieve the valid workflows for an object, use `IRoutable.getWorkflows()`. If a Workflow has not been assigned yet, you can use `IRoutable.getWorkflows()` to select a workflow, as shown in the following example.

Example: Selecting a workflow

```
public static IServiceRequest createPSR() throws APIException {
    // Create a problem report
    IAgileClass prClass =
admin.getAgileClass(ServiceRequestConstants.CLASS_PROBLEM_REPORT);
    IAutoNumber[] numbers = prClass.getAutoNumberSources();
    IServiceRequest pr =
(IServiceRequest)m_session.createObject(prClass, numbers[0]);
    // Get the current Workflow (a null object, since the Workflow has
not been set yet)
    IWorkflow wf = pr.getWorkflow();
```

```
    // Get all available workflows
    IWorkflow[] wfs = pr.getWorkflows();

    // Set the problem report to use the first workflow
    pr.setWorkflow(wfs[0]);
    return pr;
}
```

You can also set the Workflow for a Product Service Request or a Quality Change Request by selecting a value for the `Cover Page.Workflow` field, as shown in the following example.

Example: Selecting a Workflow by setting the value of the "Cover Page.Workflow" attribute

```
void selectWorkflow(IServiceRequest psr) throws APIException {
  int nAttrID;

  //Set nAttrID equal to the Workflow attribute ID
  nAttrID = ServiceRequestConstants.ATT_COVER_PAGE_WORKFLOW;
  //Get the Workflow cell
  ICell cell = psr.getCell(nAttrID);

  //Get available list values for the list
  IAgileList values = cell.getAvailableValues();

  //Select the first workflow
  values.setSelection(new Object[] {new Integer(0));
  cell.setValue(values);
}
```

Chapter 16

Creating and Managing Projects

This chapter includes the following:

- About Projects and Projects Objects 233
- Differences in the Behavior of Projects Objects 234
- Creating Projects 234
- Adding Rules for PPM Objects 236
- Loading Projects 237
- Using Projects Templates 237
- Scheduling Projects 240
- Working with Projects Baselines 243
- Delegating Ownership of a Projects to Another User 244
- Adding Resources to a Projects Team 244
- Substituting Projects Resources 247
- Locking or Unlocking Projects 247
- Working with Discussions 248

Note In Release 9.3, the name of the Program Base Class was changed to Projects and Projects to Sourcing Projects. However the interface for the Projects Base Class is still called `IProgram` in the SDK Guide and in Javadoc references.

About Projects and Projects Objects

You can use the project management features of Agile Product Portfolio Management (PPM) to define a project and its associated elements such as activity schedules, deliverables, and discussions. These capabilities enable you to determine the availability of the required resources, assigning resources to tasks, identifying bottlenecks, and responding to over- and under-allocated resource conditions. You can also create and reuse project templates.

The Projects object is used to schedule and execute projects. Each project, in addition to schedule information, contains attachments, discussions and actions items, resources and roles, and history and content of related activities. For management visibility, data is rolled up to higher levels by rules and parent-child relationships.

The Agile API provides support for creating, loading, and working with Projects. The `IProgram` interface represents all Projects objects, including programs, phases, tasks, and gates.

Similar to other Agile PLM business objects, the `IProgram` interface implements `IRoutable`, which means it uses the same `IRouteable.changeStatus()` method to change a Projects' Workflow status and to route it to other users. For more information, see Changing the Workflow Status of an Object on page 218.

Differences in the Behavior of Projects Objects

The `IProgram` interface implements several interfaces commonly used by other Agile PLM objects. However, it also provides the following distinct functionality that separates Projects objects from other objects.

- The Projects object is a container of other underlying Projects objects, such as Phases, Tasks, and Gates. The underlying Projects objects are related to the parent object, usually the Projects, through the Schedule table.
- Projects have baselines that allow you to track changes in the schedule. Therefore, the `IProgram` interface provides methods that let you create, get, or remove a baseline.
- Projects can be archived. If you archive the root Projects, the entire Projects tree is soft-deleted from the system.
- Projects can be locked or unlocked.

Creating Projects

Use the `IAgileSession.createObject()` method to create Projects. When you specify the Projects parameters, you must specify the Projects subclass (for example, Program, phase, task, or gate). For Programs, phases, and tasks, you must also specify following required Projects attributes:

- `General Info.Name`
- `General Info.Schedule Start Date`
- `General Info.Schedule End Date`
- `General Info.Duration Type`

For gates, only two attributes are required, `General Info.Name` and `General Info.Schedule End Date`.

The following example shows how to create new Projects and specify the required attributes.

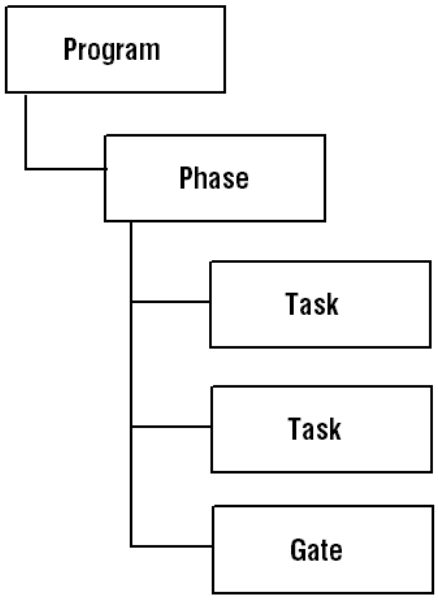
Example: Creating Projects

```
try {
  // Create a Map object to store parameters
   Map params = new HashMap();
  // Set Projects name
   String name = "APOLLO PROJECTS";
  // Set Projects start date
  Date start = new Date();
  start.setTime(1);

  // Set Projects end date
  Date end = new Date();
  end.setTime(1 + 2*24*60*60*1000);
```

```
  // Set Projects duration type
IAttribute attr =
m_admin.getAgileClass(ProgramConstants.CLASS_PROGRAM).
   getAttribute(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE);
IAgileList avail = attr.getAvailableValues();
avail.setSelection(new Object[] {"Fixed"});

  // Initialize the params object
params.put(ProgramConstants.ATT_GENERAL_INFO_NAME, name);
params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE,
start);
params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE, end);
params.put(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE, avail);
  // Create Projects
IProgram program =
(IProgram)m_session.createObject(ProgramConstants.CLASS_PROGRAM,
params);
} catch (APIException ex) {
    System.out.println(ex);
}
```

Projects contain other types of activities, such as phases, tasks, and gates. A gate is a special milestone—a task with an end date but no duration—that denotes the completion of a set of related phases, tasks, or Projects. The following figure shows the hierarchy of Projects objects.

Figure 16: Program hierarchy



You can use the `IAgileSession.createObject()` method to create phases, tasks, and gates in the same way that you create other Projects objects. Once you create these different types of activities, you can add them to the Schedule table of a Projects object. For more information, see Scheduling Projects on page 240.

Adding Rules for PPM Objects

In PLM, any object that is assigned a lifecycle phase or a Workflow, can be added as a deliverable. The only exceptions are Discussions, Users, and User groups.

Rules in PPM ensure an activity will not complete before the completion of the preceding activity as set in the workflow, or lifecycle phase. For example, if you want to ensure the completion of an activity before a Gate is opened, you can add that activity as a deliverable for the Gate to open. You can even restrict one Gate from opening before another adding the prior Gate as a deliverable for the subsequent Gate to open. For more information, refer to the Agile PLM *Product Portfolio Management User Guide*.

The SDK supports this function with `IProgram` interface as shown in the following example.

Example: Setting rules for PPM objects

```
try{
    //Get Program
    IProgram pgm =
    (IProgram)session.getObject
(ProgramConstants.CLASS_PROGRAM,"PGM00239");
    //Get Object and add as relationship under Content tab
    IChange eco = (IChange)session.getObject
(ChangeConstants.CLASS_ECO,"C00060");
    ITable table = pgm.getTable
(ProgramConstants.TABLE_RELATIONSHIPS);
    IRow row = table.createRow(eco);
    //Get the Control object status
    IStateful state = (IStateful)pgm;
    IStatus[] statuses = state.getStates();
    IStatus ctl_status = null;
    for(int i=0; i<statuses.length; i++){
      if(statuses[i].getName().equals("In Process")){
            ctl status = statuses[i];
            break;
      }
    }
    //Get the Affected object status
    state = (IStateful)eco;
    statuses = state.getStates();
    IStatus aff_status = null;
    for(int i=0; i<statuses.length; i++){
      if(statuses[i].getName().equals("Submitted")){
            aff status = statuses[i];
            break;
      }
    }

    //Add Rule
    HashMap map = new HashMap();
    map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_CONTROLOBJECT, pgm);
    map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_AFFECTEDOBJECT, eco);
    map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_CONTROLOBJECTSTATUS,
    ctl_status);
    map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_AFFECTEDOBJECTSTATUS,
    aff_status);
```

```
    row.setValue(CommonConstants.ATT_RELATIONSHIPS_RULE, map);
    System.out.println(row.getCell
(CommonConstants.ATT_RELATIONSHIPS_RULE));
  }catch (APIException ex) {
    System.out.println(ex);
  }
```

Loading Projects

To load Projects, use the `IAgileSession.getObject()` method. To uniquely identify the Projects object, specify the value for the `General Info.Numbe`r attribute. You can also load a Projects object by searching for it by name, and then selecting it from the search results.

Note The `IProgram.getName()` method actually returns the value of the `General Info.Number` attribute, not `General Info.Name`.

Example: Loading Projects

```
public IProgram loadProgram(String number) throws APIException {
  IProgram program =
(IProgram)m_session.getObject(IProgram.OBJECT_TYPE, number);
  return program;
}
```

Note The News table for Projects is disabled by default. To enable it, log in to the Java Client as an Administrator and make the News tab visible.

Using Projects Templates

Projects templates make it easy to define a new Projects object, activity, or task. A template is a Projects with the `General Info.Template` attribute set to “Template”. You can use a template to create a new Projects by loading it and then using the `IProgram.saveAs()` method.

This special version of the `saveAs()` method enables to use the SDK to:

- Create a new Projects from a template and specify the tables that you want copied over
- Change the owner of the Projects and the owner of the children
- Create a new Projects template by saving a Projects as a template

Creating New Projects Using Templates

You can use this special version of the `saveAs()` method to specify the Projects tables that you want to copy from the original Projects to the new Projects. You don’t need to specify all tables. The General Info, Schedule, Dependencies - Dependent Upon, Dependencies - Required For, and Workflow tables are copied automatically. The Discussion, News, and History tables cannot be copied. Generally, you should copy Page Two, Page Three (if it’s used), and the Team table, as shown in the example below.

Example: Creating a new Projects from a template

```
try {
  // Get the Projects template whose number is PGM00004
  IProgram template =
(IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00004");
  if (template != null) {
    // Create a hash map of the program attributes to use for the new
program HashMap map = new HashMap(); String name = "Scorpio Program";
    IAttribute att =
m_admin.getAgileClass(ProgramConstants.CLASS_PROGRAM).getAttribute(
                      ProgramConstants.ATT_GENERAL_INFO_TEMPLATE);
    IAgileList templateList = att.getAvailableValues();
    // Note: Available values for the Template attribute are Active,
Proposed, and Template
    templateList.setSelection(new Object[] {"Active"});
    map.put(ProgramConstants.ATT_GENERAL_INFO_NAME, name);
    map.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE, new
Date());
    map.put(ProgramConstants.ATT_GENERAL_INFO_TEMPLATE, templateList);
    // Define the tables to copy to the new program from the template
Integer pagetwo = ProgramConstants.TABLE_PAGETWO;
    Integer pagethree = ProgramConstants.TABLE_PAGETHREE;
    Integer team = ProgramConstants.TABLE_TEAM;
    Object[] tables = new Object[]{pagetwo, pagethree, team};
    // Save the template as a new program
    IProgram program =
(IProgram)template.saveAs(ProgramConstants.CLASS_PROGRAM,
                    tables, map);
  }
} catch (APIException ex) {
    System.out.println(ex);
}
```

Creating Projects and Changing Ownerships

When you create a Projects from a template using the `saveAs()` API call, you can change the ownership of the Projects and propagate the change to the children of the Projects. In SDK, the exposed API is:

```
public IAgileObject saveAs(Object type, Object[] tablesToCopy,Object
params, boolean applyToChildren)
throws APIException;
```

This is done by specifying a value for both the `ProgramConstants` and the `OWNER` attributes. The value for the `OWNER` attribute is required in order to change the Projects ownership. Set the Boolean `applyToChildren` to true if you want to apply the `OWNER` value to all children

In the UI, when you create a Projects from a template, you have the option to change ownership of the Projects and applying the change to the children. In this situation, the SDK mirrors the UI. However, the original Projects must be a *Template* to create a Projects from a template via SDK's `saveAs() API`.

Note In the SDK, a Projects is a template when the value of the `General Info.Template` attribute in the original `program` is set to Template.

Example: Creating a Projects from a template, change owner, and propagate change

```
public  IProgram saveTemplateAndSetOwner (IProgram  template, String
userID, boolean applyToChildren) throws APIException {
/*  "template" is a program template
   userID --  The "userID" of the user that
    is specified as the owner of the Saved program object
    applyToChildren --  true or false.
    If "true" the "specified owner" will be the owner of the entire
    program tree
 If "false", the specified owner will be the owner of the Root Parent
object only
*/

HashMap map = new HashMap () ;
String newPgmName =
    "PROG" + System.currentTimeMillis() ;  // Generate a random name for
    the Saved Program object
IUser user =
    session.getObject(UserConstants.CLASS_USER, userID) ;
    map.put(ProgramConstants.ATT_GENERAL_INFO_NAME, newPgmName);
    map.put(ProgramConstants.ATT_GENERAL_INFO_OWNER, User);
    map.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE, new
    Date());
// Define the tables to copy  from the template
// If you do not want any tables to be copied,
// specify "null" for the "tables" param

    Integer pageTwo = ProgramConstants.TABLE_PAGETWO ;
    Integer pageThree = ProgramConstants.TABLE_PAGETHREE ;
    Integer team = ProgramConstants.TABLE_TEAM ;
Object[] tables =
    { pageTwo, pageThree, team } ;
 IProgram pgm =
    (IProgram) root.saveAs(ProgramConstants.CLASS_PROGRAM, tables, map,
    applyToChildren);
 System.out.println
    ("New Program Number = " + pgm.getName()) ;
 System.out.println
    ("Owner Value = " +
    pgm.getValue(ProgramConstants.ATT_GENERAL_INFO_OWNER).toString())
 return pgm ;
}
```

Saving Projects as Templates

When you create a Projects, you can specify that it's a template by setting the value of the Template attribute (`ProgramConstants.ATT_GENERAL_INFO_TEMPLATE`) to "Template". You can only do this when you create a Projects or when you save it as a new Projects. Existing Projects cannot be changed from the "Active" or "Proposed" state to "Template".

The following example shows how to open a Projects object and save it as a template.

Example: Saving a Projects object as a template

```
try {
// Get the program whose number is PGM00005
 IProgram program =
    (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00005");
 if (program != null) {
// Create a hash map of the program attributes to use for the new
program
 HashMap map = new HashMap();
 String name = "Rapid Development");
 IAttribute att =
    m_admin.getAgileClass(ProgramConstants.CLASS_PROGRAM).getAttribute(P
    rogramConstants.ATT_GENERAL_INFO_TEMPLATE);
 IAgileList templateList =
    att.getAvailableValues();
// Note: Available values for the Template attribute
    are Active, Proposed, and Template
 templateList.setSelection(new Object[] {"Template"});
    map.put(ProgramConstants.ATT_GENERAL_INFO_NAME, name);
    map.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE, new
    Date());
    map.put(ProgramConstants.ATT_GENERAL_INFO_TEMPLATE, templateList);
//Define the tables to copy to the template
 Integer pagetwo =
    ProgramConstants.TABLE_PAGETWO;
 Integer pagethree =
    ProgramConstants.TABLE_PAGETHREE;
 Integer team =
    ProgramConstants.TABLE_TEAM;
 Object[] tables =
    new Object[]{pagetwo, pagethree, team};
// Save the program as a template
 IProgram program =
    (IProgram)template.saveAs(ProgramConstants.CLASS_PROGRAM,
                    tables, map);
  }
} catch (APIException ex) {
   System.out.println(ex);
}
```

Scheduling Projects

To schedule Projects, edit the Schedule table, which lets you add, edit, and remove schedule items. To add a new row to the Schedule table, use the `ITable.createRow()` method and specify an `IProgram` object for the parameter.

Example: Modifying the Schedule table

```
try {
```

```
  // Define a row variable IRow row = null;
  // Set the date format
DateFormat df = new SimpleDateFormat("MM/dd/yy");
  // Get a Projects
   IProgram program =
     (IProgram)m_session.getObject(ProgramConstants.CLASS_PROGRAM,
   "PGM00012");
  if (program != null) {
    // Get the Schedule table
   ITable schedule =
      program.getTable(ProgramConstants.TABLE_SCHEDULE);
    Iterator i = schedule.iterator();
    // Find task T000452 and remove it
    while (i.hasNext()) {
      row = (IRow)i.next();
   String num =
      (String)row.getValue(ProgramConstants.ATT_GENERAL_INFO_NUMBER);
      if (num.equals("T000452")) {
        schedule.removeRow(row);
        break;
      }
    }
    // Add a phase
   HashMap info = new HashMap();
   info.put(ProgramConstants.ATT_GENERAL_INFO_NAME, "Specifications
   phase");
   info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE,
   df.parse("06/01/05"));
   info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
   df.parse("06/10/05"));
   IAttribute attr =
   m_admin.getAgileClass(ProgramConstants.CLASS_PHASE).
       getAttribute(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE);
   IAgileList list = attr.getAvailableValues();
   list.setSelection(new Object[] {"Fixed"});
   info.put(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE, list);
   IProgram phase =
   (IProgram)m_session.createObject(ProgramConstants.CLASS_PHASE,
   info);
   row = schedule.createRow(phase);
       // Add a task info = null; list = null;
   info.put(ProgramConstants.ATT_GENERAL_INFO_NAME, "Write
   specifications");
   info.put(ProgramConstants.ATT_GENERAL_INFO_NUMBER, "T000533");
   info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE,
   df.parse("06/01/05"));
   info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
   df.parse("06/05/05"));
   attr = m_admin.getAgileClass(ProgramConstants.CLASS_TASK).
     getAttribute(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE);
   list = attr.getAvailableValues();
   list.setSelection(new Object[] {"Fixed"});
   info.put(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE, list);
   IProgram task =
   (IProgram)m_session.createObject(ProgramConstants.CLASS_TASK, info);
```

```
     row = schedule.createRow(task);
        // Add a gate info = null;
    info.put(ProgramConstants.ATT_GENERAL_INFO_NAME, "Specifications
    complete");
    info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
    df.parse("06/10/05"));
    IProgram gate =
    (IProgram)m_session.createObject(ProgramConstants.CLASS_GATE, info);
    row = schedule.createRow(gate);
   }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Once a Projects' schedule is defined, you can reschedule it using the `IProgram.reschedule()` method. The `reschedule()` method takes a couple of parameters, the `IProgram.RESCHEDULE` constant and the new value for that schedule option. Here are the list of `IProgram.RESCHEDULE` constants you can use:

- `STARTDATE` – This moves the scheduled start date to the specified date.
- `ENDDATE` – This moves the scheduled end date to the specified date.
- `BACKWARDDAYS` – This moves the schedule backward by the specified number of days.
- `FORWARDDAYS` – This moves the schedule forward by the specified number of days.

Example: Rescheduling Projects

```
try {
  // Get a Projects
IProgram program = (IProgram)m_session.getObject(IProgram.OBJECT_TYPE,
"PGM00012");
if (program != null) {
    // Define new start and end dates String startDate = "02/01/2005
GMT"; String endDate = "06/01/2005 GMT";
SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy z");
Date start = df.parse(startDate);
Date end = df.parse(endDate);

    // Change the schedule start date
    program.reschedule(IProgram.RESCHEDULE.STARTDATE, start);
    // Change the schedule end date
program.reschedule(IProgram.RESCHEDULE.ENDDATE, end);
    // Move the schedule backward three days
program.reschedule(IProgram.RESCHEDULE.BACKWARDDAYS, new Integer(3));
    // Move the schedule forward two days
    program.reschedule(IProgram.RESCHEDULE.FORWARDDAYS, new
Integer(2));
  }
} catch (Exception ex) {
  System.out.println(ex);
}
```

Working with Projects Baselines

Projects baselines allow you to compare actual progress with your original plans. When you create a baseline, a snapshot of your Projects' schedule is preserved. The original estimates contained in the baseline are permanent reference points against which you can compare the updated task structure, schedule, and actual dates.

Baselines can be created only for the root Projects object. You can save multiple baselines, and retrieve them later for comparison. The `IProgram` interface provides the following methods for creating, retrieving, and removing baselines:

- `createBaseline(java.lang.Object)`
- `getBaseline()`
- `getBaselines()`
- `removeBaseline(java.lang.Object)`
- `selectBaseline(java.lang.Object)`

Example: Creating and retrieving baselines

```
try {
  // Get a Projects
   IProgram program =
   (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
  if (program != null) {
    // Create a baseline
    Object baseline = program.createBaseline("august 8 baseline");
    // Get all baselines
    Map map = program.getBaselines();

    // Get the first baseline
    Set keys = map.keySet();
    Object[] objs = keys.toArray();
    baseline = map.get(objs[0]);

    // Remove the first baseline
    program.removeBaseline(baseline);

    // Get all baselines again
    map = program.getBaselines();

    // Select the first baseline
    If (map.size() > 0) {
      keys = map.keySet();
      objs = keys.toArray();
      baseline = map.get(objs[0]);
      program.selectBaseline(baseline);
```

```
    }
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Delegating Ownership of a Projects to Another User

The owner or manager of a Projects object can assign the ownership of the Projects to other users by delegating it. The delegated user receives a request that he can accept or decline. If he accepts, the delegated user becomes owner of the task. A delegated owner is automatically given the Projects Manager role for the delegated Projects object.

To delegate ownership of a Projects, use the `IProgram.delegateOwnership()` method. When you delegate ownership of a Projects, you automatically update the Delegated Owner field, which is read-only. The `delegateOwnership()` method lets you specify whether delegated ownership also applies to the Projects' children.

Example: Delegating ownership of a Projects object

```
try {
  // Get the task whose number is T00012
  IProgram task = (IProgram)m_session.getObject(IProgram.OBJECT_TYPE,
"T00012");
  if (task != null) {
    // Get a user
    IUser user1 = (IUser)m_session.getObject(UserConstants.CLASS_USER,
"kkieslowski");
    if (user1 != null) {
      // Delegate the task to the user
      task.delegateOwnership(user1, false);
    }
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Adding Resources to a Projects Team

The Team table lets you manage the team member list for a Projects object. You can add or remove team members, change team members' roles, and change their allocation. You must have the appropriate privileges to modify a Projects' Team table.

When you add a resource to the Team table, you specify what roles the user or user group has for that Projects object. The roles available are not the complete set of Agile PLM roles; they are roles specifically related to Projects Execution functionality. Here is the list of roles you can assign to team members:

- Executive

- Change Analyst
- Program Team Member
- Program Manager
- Resource Pool Owner
- Program Administrator

For a description of each of these roles, refer to the *Agile PLM Administrator Guide*.

The Team table has two attributes that require special mention:

- `ProgramConstants.ATT_TEAM_NAME`
- `ProgramConstants.ATT_TEAM_ROLES`.

These are SingleList and MultiList attributes, respectively. To get the available values for these attributes, use `ITable.getAvailableValues()` instead of `IAttribute.getAvailableValues()`. Otherwise, the `IAgileList` object returned from the method may contain invalid list values.

Example: Adding resources to a Projects' team

```
try {
      // Get users
      IUser user1 = (IUser)session.getObject(UserConstants.CLASS_USER,
"daveo");
      IUser user2 = (IUser)session.getObject(UserConstants.CLASS_USER,
"yvonnec");
      IUser user3 = (IUser)session.getObject(UserConstants.CLASS_USER,
"albertl");
      IUser user4 = (IUser)session.getObject(UserConstants.CLASS_USER,
"brians");
     //  Get a resource pool (user group)
      IUserGroup pool =
(IUserGroup)session.getObject(IUserGroup.OBJECT_TYPE, "Development");
     // Add all four users to the resource pool
      ITable usersTable =
pool.getTable(UserGroupConstants.TABLE_USERS);
      usersTable.createRow(user1);
      usersTable.createRow(user2);
      usersTable.createRow(user3);
      usersTable.createRow(user4);

      // Get a Projects
      IProgram program =
(IProgram)session.getObject(IProgram.OBJECT_TYPE, "PGM02423");
      if (program != null) {
      // Get the Team table of the program
      ITable teamTable = program.getTable(ProgramConstants.TABLE_TEAM);
     // Get Roles attribute values (use ITable.getAvailableValues)
      IAgileList attrRolesValues =
teamTable.getAvailableValues(ProgramConstants.ATT_TEAM_ROLES);
```

```
     // Create a hash map to hold values for row attributes
      Map map = new HashMap();
      // Add the first user to the team
      attrRolesValues.setSelection(new Object[]{"Change
Analyst","Projects Manager"});
      map.put(ProgramConstants.ATT_TEAM_NAME, user1);
      map.put(ProgramConstants.ATT_TEAM_ROLES, attrRolesValues);
      IRow row1 = teamTable.createRow(map);
      // Add the second user to the team
      attrRolesValues.setSelection(new Object[]{"Projects
Administrator"});
      map.put(ProgramConstants.ATT_TEAM_NAME, user2);
      IRow row2 = teamTable.createRow(map);

      // Add the resource pool to the team
      attrRolesValues.setSelection(new Object[]{"Projects Team
Member"});
      map.put(ProgramConstants.ATT_TEAM_NAME, pool);
      IRow row3 = teamTable.createRow(map);

      }
```

In Agile Web Client, when you add a resource pool to the Team table, you can replace the pool with one or more resources contained within it. In other words, instead of assigning the entire resource pool, you can assign select users from the pool. The `IProgram.assignUsersFromPool()` method reproduces this functionality. To use `assignUsersFromPool()`, you must specify a user group that has already been added to the Projects' Team table.

Example: Assigning users from a resource pool

```
public void replaceUserGroupWithUser(IProgram program) throws Exception
{
  // Get the Team table
  ITable teamTable = program.getTable(ProgramConstants.TABLE_TEAM);
  // Get a table iterator
  Iterator it = teamTable.iterator();

  // Find a user group and replace it with one of its members, kwong
  while(it.hasNext()){
    IRow row = (IRow)it.next();
    IDataObject object = row.getReferent();
    if(object instanceof IUserGroup){
      IUserGroup ug = (IUserGroup)object;
      ITable users = ug.getTable(UserGroupConstants.TABLE_USERS);
      Iterator ref_it = users.getReferentIterator();
      while(ref_it.hasNext()){
        IUser user = (IUser)ref_it.next();
        if(user.getName().equals("kwong")) {
          program.assignUsersFromPool(new IUser[]{user}, ug, true);
          break;
```

```
        }
      }
    }
  }
}
```

Substituting Projects Resources

A resource's availability can frequently change due to overloading, reassignments, vacation, and illness. You can substitute an existing resource for another resource. The current resource's role is assigned to the substituted resource, but only for that Projects. To substitute Projects resources, use the `IProgram.substituteResource()` method.

When you substitute resources, you can specify users as well as user groups. You can also specify whether the resource assignment applies to the Projects' children.

Example: Substituting Projects resources

```
try {
  // Get a Projects
IProgram program =
    (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
  if (program != null) {
    // Get users
IUser u1 =
    (IUser)m_session.getObject(UserConstants.CLASS_USER, "akurosawa");
IUser u2 =
    (IUser)m_session.getObject(UserConstants.CLASS_USER, "creed");
IUser u3 =
    (IUser)m_session.getObject(UserConstants.CLASS_USER, "dlean");
IUser u4 =
    (IUser)m_session.getObject(UserConstants.CLASS_USER, "jford");
    // Get a user group
IUserGroup ug =
    (IUserGroup)m_session.getObject(IUserGroup.OBJECT_TYPE,
                    "Directors");
        // Substitute u1 with u3 and do not apply to children
    program.substituteResource(u1, u3, false);
        // Substitute u2 with u4 and apply to children
    program.substituteResource(u2, u4, true);
        // Substituete u4 with a user group, and apply to children
    program.substituteResource(u4, ug, true);
  }
} catch (APIException ex) {
    System.out.println(ex);
}
```

Locking or Unlocking Projects

The owner of Projects can lock or unlock the Projects object. When a Projects is locked, its schedule cannot be modified. To lock or unlock a Projects, use the `IProgram.setLock()` method.

Note Projects are automatically locked when you use the Gantt Chart or the Microsoft Project integration functionality in Agile Web Client.

Example: Locking Projects

```
try {
  // Get a Program
IProgram program =
    (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
  if (program != null) {
    // Lock it
    program.setLock(true);
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Working with Discussions

During the course of a project, issues arise that require users to collaborate and exchange information. Agile PLM provides threaded discussion functionality that allows team members to reply with their feedback, providing a record of their thoughts and ideas. Discussions are asynchronous; that is, they do not require a simultaneous connection from all discussion participants. People can reply to any thread of the discussion independently. To close issues, action items can be assigned to team resources. The Discussion object is used to manage both threaded discussions and the action items related to them.

Discussion objects, unlike Projects, are not routable objects. Therefore, discussions do not have workflows associated with them.

Note The Action Items, Cover Page, and Replies tables appear on the Discussion tab in Agile PLM clients. The Page Two table appears on the Details tab in Agile PLM clients. The Where Used table is not supported, its functionality is replaced by `General Info.Related To` field.

Creating a Discussion

To create a discussion, use the `IAgileSession.createObject()` method. When you specify discussion parameters, you must specify the discussion subclass and the following required discussion attributes:

- Cover Page.Number
- Cover Page.Subject

In addition, you must also specify data for the Cover Page.Notify List and Cover Page.Message attributes. Otherwise, the discussion does not have a Notification list, or a message that users can respond to.

The following example shows how to create a new discussion and add it to the Discussion table of a Projects.

Example: Creating a discussion

```
try {
  // Create a hash map variable
  Map map = new HashMap();

  // Set the Number field
  IAgileClass discussionClass =
m_session.getAdminInstance().getAgileClass(
                                DiscussionConstants.CLASS_DISCUSSION);
  String number =
discussionClass.getAutoNumberSources()[0].getNextNumber();
  // Set the Subject field
  String subject = "Packaging issues";

  // Make the Message field visible
  IAttribute attr =
discussionClass.getAttribute(DiscussionConstants.ATT_COVER_PAGE_MESSAGE
);
  IProperty propVisible =
attr.getProperty(PropertyConstants.PROP_VISIBLE);
  IAgileList list = propVisible.getAvailableValues();
  list.setSelection(new Object[] { "Yes" });

  // Set the Message field
  String message = "We still have problems with the sleeves and
inserts." +
                   "Let's resolve these things at the team meeting on
Friday.";
  // Set the Notify List field
  IUser user1 = m_session.getCurrentUser();
  IUser user2 = (IUser)m_session.getObject(UserConstants.CLASS_USER,
"jdassin");
  attr =
discussionClass.getAttribute(DiscussionConstants.ATT_COVER_PAGE_NOTIFY_
LIST);
```

```
  list = attr.getAvailableValues();
  list.setSelection(new Object[] {user1, user2});

  // Put the values into the hash map
  map.put(DiscussionConstants.ATT_COVER_PAGE_NUMBER, number);
  map.put(DiscussionConstants.ATT_COVER_PAGE_SUBJECT, subject);
  map.put(DiscussionConstants.ATT_COVER_PAGE_MESSAGE, message);
  map.put(DiscussionConstants.ATT_COVER_PAGE_NOTIFY_LIST, list);
  // Create a Discussion object
  IDiscussion discussion = (IDiscussion)m_session.createObject(
                      DiscussionConstants.CLASS_DISCUSSION, map);
  // Get a Projects
  IProgram program =
(IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
  if (program != null) {
    // Get the Discussion table
    ITable discTable =
program.getTable(ProgramConstants.TABLE_DISCUSSION);

    // Add the new discussion to the table
    discTable.createRow(discussion);
  }

} catch (APIException ex) {
  System.out.println(ex);
}
```

Replying to a Discussion

Team members or notified users—that is, users listed in the Cover Page.Notified List field of a discussion—can reply to discussions. When you reply to a discussion, you create another nested table in the Replies table.

Example: Replying to a discussion

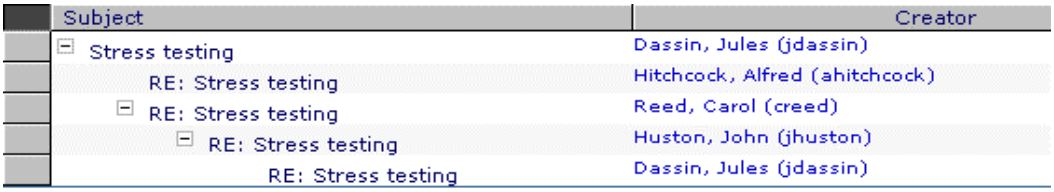
```
private void replyToDiscussion() throws Exception {
  Iterator it;
  IDiscussion discussion;

  // Get a Projects
IProgram program =
   (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
  // Get the Discussion table
ITable discTable =
   program.getTable(ProgramConstants.TABLE_DISCUSSION);
  // Get the first Discussion listed
  if (discTable.size()!=0) {
    it = discTable.iterator();
    if (it.hasNext()) {
```

```
      IRow row = (IRow)it.next();
      discussion = (IDiscussion)row.getReferent();
    }
    // Get the Replies table
ITable repliesTable =
   discussion.getTable(DiscussionConstants.TABLE_REPLIES);
    // Iterate to the only row of the Replies table and send a reply it
= repliesTable.iterator();
   if (it.hasNext()) {
      IRow row = (IRow)it.next();
      IMessage message = (IMessage)row;
      HashMap response = new HashMap();

      // Set the Subject field (use the same Subject as the parent)
response.put(MessageConstants.ATT_COVERPAGE_SUBJECT,
   row.getValue(DiscussionConstants.ATT_REPLIES_SUBJECT));
      // Make the Message field visible
IAgileClass discussionClass =
   m_session.getAdminInstance().getAgileClass(DiscussionConstants.CLASS
   _DISCUSSION);
IAttribute attr =
   discussionClass.getAttribute(DiscussionConstants.ATT_COVER_PAGE_MESS
   AGE);
IProperty propVisible =
   attr.getProperty(PropertyConstants.PROP_VISIBLE);
IAgileList list =
   propVisible.getAvailableValues();
     list.setSelection(new Object[] { "Yes" });
      // Set the Message field
response.put(MessageConstants.ATT_COVERPAGE_MESSAGE,
   "The spec needs to be updated to reflect the latest decisions.");
      // Send a reply
 message.reply(response);
    }
  }
}
```

The previous example showed how to reply to the root discussion. But what if a discussion has several replies and you want to reply to the latest one? That is a little more complicated, and requires further understanding of the Replies table.

The Replies table of a discussion is different from other Agile PLM tables. It contains only one row, even if there are multiple replies. If the discussion has multiple replies, they are contained within a series of nested tables. To select the latest reply, expand the Replies table to its last nested table. The following figure shows an expanded Replies table in Agile Web Client.

Figure 17: Expanded Replies table



You can use a recursive method (one that calls itself) to expand all levels of the Replies table, as shown in the following example. Subsequent levels of the Replies table are obtained by getting the value of the Child Table attribute (`DiscussionConstants.ATT_REPLIES_CHILD_TABLE`).

Example: How to expand the Replies table

```
// Read the Replies table
public void readRepliesTable(IDiscussion discussion) throws Exception {
    ITable replies =
    discussion.getTable(DiscussionConstants.TABLE_REPLIES);
    browseReplies(0, replies);
}
// Recursively browse through all levels of the Replies table
void browseReplies(int indent, ITable replies) throws Exception {
Iterator i = replies.iterator();
    while (i.hasNext()) {
    IRow row = (IRow) i.next();
    System.out.print(indent(indent*4));
    readRow(row);
    System.out.println();
ITable followup =
    (ITable)row.getValue(DiscussionConstants.ATT_REPLIES_CHILD_TABLE);
    browseReplies(indent + 1, followup);
  }
}
// Read each cell in the row and print the attribute name and value
static protected void readRow(IRow row) throws Exception {
    ICell[] cells = row.getCells();
    for (int j = 0; j < cells.length; ++j) {
    Object value = cells[j].getValue();
    System.out.print( "\t" + cells[j].getAttribute().getName() + "="+
    value);
  }
}
// Indent text
private String indent(int level) {
    if (level <= 0) {
return "";
  }
    char c[] = new char[level*2];
    Arrays.fill(c, ' ');
```

```
    return new String(c);
}
```

Joining a Discussion

Agile Web Client allows users to join a discussion by clicking the Discussion tab of a Projects, and then clicking the Join button. When you join a discussion, your username is added to the Notify List field of the Discussion object. To join a discussion using the Agile API, simply add yourself to the Notify List field. You can join a discussion only if you are a team member of the Projects.

Note If you are not on the Notify List of a Discussion object, you cannot read the replies. However, anyone listed on the Team table of a Projects can join a discussion associated with that Projects.

Example: Joining a discussion

```
try {
  // Get a Projects
IProgram program =
    (IProgram)m_session.getObject(ProgramConstants.CLASS_PROGRAM,
    "PGM00012");
    if (program != null) {
     // Get the Discussion table
ITable discTable =
    program.getTable(ProgramConstants.TABLE_DISCUSSION);
 // Get the first discussion
IRow row =
    (IRow)discTable.iterator().next();
IDiscussion discussion =
    (IDiscussion)row.getReferent();
  // Add yourself and another user to the Notify List field
IUser user1 =
    m_session.getCurrentUser();
IUser user2 =
    (IUser)m_session.getObject(UserConstants.CLASS_USER, "owelles");
ICell cell =
    discussion.getCell(DiscussionConstants.ATT_COVER_PAGE_NOTIFY_LIST);
IAgileList list =
    (IAgileList)cell.getAvailableValues();
    list.setSelection(new Object[] {user1, user2});
  }
} catch (APIException ex) {
  System.out.println(ex);
}
```

Creating an Action Item

Action items can be created as part of a Discussion object. If a discussion raises an issue that requires someone to perform an action, you can assign that action to another user. Action items have a subject, status, due date, and an assigned user. When you create an action item, it appears in the Notifications & Requests Inbox of the assigned user.

To create an action item, use the `ITable.createRow()` method to add a row to the Action Items table of a Projects object. Make sure the map object used to initialize the row contains parameters for the Subject, Assigned To, and Due Date fields.

Example: Creating an action item

```
private void replyToDiscussion() throws Exception {
  // Get a Projects
  IProgram program =
(IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
  if (program != null) {
    // Create a hash map for Action Item parameters
    HashMap map = new HashMap();

    // Set the Subject field
    String subj = "Update packaging requirements";
    map.put(ProgramConstants.ATT_ACTION_ITEMS_SUBJECT, subj);

   // Set the Assigned To field
   IUser user1 = (IUser)m_session.getObject(UserConstants.CLASS_USER,
   "akurosawa");
   IAttribute attr = m_session.getAdminInstance().getAgileClass(
      ProgramConstants.CLASS_PROGRAM).getAttribute(
      ProgramConstants.ATT_ACTION_ITEMS_ASSIGNED_TO);
   IAgileList list = attr.getAvailableValues();
   list.setSelection(new Object[] {user1});
   map.put(ProgramConstants.ATT_ACTION_ITEMS_ASSIGNED_TO, list);

   // Set the Due Date field
   DateFormat df = new SimpleDateFormat("MM/dd/yy");
   map.put(ProgramConstants.ATT_ACTION_ITEMS_DUE_DATE,
   df.parse("03/30/05"));
   // Get the Action Items table
   Table table = program.getTable(ProgramConstants.TABLE_ACTIONITEMS);
   // Add the new Action Item to table
   table.createRow(map);
   }
} catch (APIException ex) {
   System.out.println(ex);
}
```

Chapter 17

Working with Product Cost Management

This chapter includes the following:

- Overview 255
- Working with Price Objects 256
- Managing Pricing 256
- Working with Suppliers 261
- Working with Sourcing Projects 262

Overview

The Product Sourcing module of the Agile PLM supports, enhances, and simplifies the handling of all product cost-related data throughout the product lifecycle. This enables you to effectively manage and manipulate Sourcing content, collaborate with suppliers to establish new Sourcing content, and analyze the data. Product Sourcing supports the following functions:

- Create Sourcing Projects
- Gather and prepare product content
- Leverage pricing contracts and history
- Create RFQs
- Manage supplier RFQ responses and negotiate pricing (Not supported by PCM SDK)
- Conduct Sourcing Projects analysis

The Agile API supports the following Product Sourcing objects:

- `IChange` – This is interface for the `Change` class, which includes Price Change Orders (PCOs).
- `IPrice` – This is interface for the `Price` class, which handles both published prices and historical prices.
- `IProject` – This is interface for the `Sourcing Projects` class, which is the container used for product Sourcing data.
- `IRequestForQuote` – This is interface for the `RequestForQuote` class, which represents an RFQ for a Sourcing Projects.
- `ISupplier` – This is interface for the `Supplier` class.

Except for the `ISupplierResponse` object, the Agile API allows you to read and modify all Product Sourcing objects. The following table lists the create, read, and modify rights for Product Sourcing objects.

Object	Create	Read	Modify
IChange (including PCO)	Yes	Yes	Yes
IPrice	Yes	Yes	Yes
IProject	Yes	Yes	Yes
IRequestForQuote	Yes	Yes	Yes
ISupplier	Yes	Yes	Yes

Working with Price Objects

Agile PLM's price management solution replaces inefficient manual systems, where prices are often stored in files, spreadsheets, or databases in disparate locations. The Agile PLM system allows you to create and centrally manage prices and terms for items and manufacturer parts.

There are two out-of-the-box Price classes provided with the system:

- Historical Quotes – This is a historical Quote object that contains price quotes from previous Sourcing projects or legacy data.
- Published Prices – This is a historical Published Price that contains published prices or contract prices on current items and manufacturer parts.

These are the basic steps used to define pricing for an item or manufacturer part:

1. Users with the appropriate role can create a new Price object, specifying the Number, Description, Item or Manufacturer Part, Supplier, Site, and Customer.
2. After creating a Price object, users can build out a price/terms matrix for each associated item or manufacturer part. The price and terms matrix includes Effectivity Dates, Quantity, Price, and Cancellation Windows.
3. The Price object is submitted and goes through a Workflow approval process. Other users can approve or reject the object.
4. Users with the appropriate role can create a Price Change Order (PCO) to modify a Price object that has been released. The updated Price object is again submitted for approval.

Managing Pricing

Agile PLM's price management solution replaces inefficient manual systems, where prices are often stored in files, spreadsheets, or databases in disparate locations. The Agile PLM system allows you to create and centrally manage prices and terms for items and manufacturer parts.

There are two out-of-the-box Price classes provided with the system:

- Historical Quotes – This is a historical Quote object that contains price quotes from previous Sourcing projects or legacy data.

- Published Prices – This is a historical Published Price that contains published prices or contract prices on current items and manufacturer parts.

These are the basic steps used to define pricing for an item or manufacturer part:

1. Users with the appropriate role can create a new Price object, specifying the Number, Description, Item or Manufacturer Part, Supplier, Site, and Customer.
2. After creating a Price object, users can build out a price/terms matrix for each associated item or manufacturer part. The price and terms matrix includes Effectivity Dates, Quantity, Price, and Cancellation Windows.
3. The Price object is submitted and goes through a Workflow approval process. Other users can approve or reject the object.
4. Users with the appropriate role can create a Price Change Order (PCO) to modify a Price object that has been released. The updated Price object is again submitted for approval.

Creating a Price Object

There are several steps to create a Price object. First, specify the object class and the unique identifying attributes, and then use `IAgileSession.createObject()` to return the new Price object.

Price objects are more complex than other Agile API objects because they have several key attributes that must be specified. Most other Agile API objects have only one key object, such as the object's number. With a Price object, you must specify a number, customer, item or manufacturer part, revision (for items), Program, site, and supplier. If any one of these attributes is missing, an exception will be thrown and the Price object won't be created.

Note If you are not dealing with site-specific information, specify the Global site for the Manufacturing Site attribute.

After you create a Price object, you can further define it by setting values for Cover Page, Page Two, and Page Three fields. To define prices and terms for items and manufacturer parts, add rows to the Price Lines table. If there are files or documents to attach, add them to the Attachments table.

Defaults

To create a price with `Program==All and Customer==All`, you do not need to pass values for `PriceConstants.ATT_GENERAL_INFORMATION_CUSTOMER` and `PriceConstants.ATT_GENERAL_INFORMATION_Program` during price creation. By default, the price will be created with `Progam==All and Customer==All`.

Specifying Item Revision

When you specify the item revision during price creation, you need to pass the change number, instead of the revision number.

Example: Specifying Item Revision by Passing the Change Number

```
//Pass the change number
params.put(Priceconstants.ATT_GENERAL_INFORMATION_ITEM_REV, "CO-35884");
```

```
//Instead of the revision number
params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_REV, "B")
```

Creating a Published Price

The following example shows how to create a published price.

Example: Creating a published price

```
public void createPublishedPrice(ICustomer customer, ISupplier
supplier) throws Exception {
    HashMap params = new HashMap();
IAgileClass cls =
    m_admin.getAgileClass(PriceConstants.CLASS_PUBLISHED_PRICE);
IAutoNumber an =
    cls.getAutoNumberSources()[0];
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_NUMBER, an);
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_CUSTOMER,
    customer);
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_NUMBER,
    "1000-02");
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_REV, "CO-
    35884");
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_PROGRAM,
    "PROGRAM0023");
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_MANUFACTURING_SITE
    , "San Jose");
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_SUPPLIER,
    supplier);
IPrice price = (IPrice)m_session.createObject(cls, params);
}
```

Loading a Price Object

To load a Price object, use the `IAgileSession.getObject()` method. To uniquely identify a Price object, specify the value for the Title Block | Number attribute.

Example: Loading a Price object

```
public IPrice getPrice() throws Exception {
  IPrice price = (IPrice)m_session.getObject(IPrice.OBJECT_TYPE,
"PRICE10008");
return price;
}
```

For a list of Price object tables, refer to the Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in SDK_samples (ZIP file). To access this file, see the Note in Client-Side Components on page 2.

Adding Price Lines

The Price Lines table of a Price object is where you define the prices and terms for the related item or manufacturer part. When you add a row to the Price Lines table, you must initialize the row with values. At a minimum, you must specify values for the following attributes:

- `ATT_PRICE_LINES_SHIP_FROM`

- ATT_PRICE_LINES_SHIP_TO
- ATT_PRICE_LINES_PRICE_EFFECTIVE_FROM_DATE
- ATT_PRICE_LINES_PRICE_EFFECTIVE_TO_DATE
- ATT_PRICE_LINES_QTY

If you fail to specify a value for one of these attributes, the Price Lines row won't be created.

Example: Adding price lines

```
public void addPriceLines(IPrice price) throws Exception {
DateFormat df = new SimpleDateFormat("MM/dd/yy");
  IAgileClass cls = price.getAgileClass();
  ITable table = price.getTable(PriceConstants.TABLE_PRICELINES);
  IAttribute attr = null;
  IAgileList listvalues = null;
  HashMap params = new HashMap();

  //Set Ship-To Location (List field)
  attr = cls.getAttribute(PriceConstants.ATT_PRICE_LINES_SHIP_TO);
  listvalues = attr.getAvailableValues();
  listvalues.setSelection(new Object[] { "San Jose" });
  params.put(PriceConstants.ATT_PRICE_LINES_SHIP_TO, listvalues);
  //Set Ship-From Location (List field) attr =
cls.getAttribute(PriceConstants.ATT_PRICE_LINES_SHIP_FROM); listvalues
= attr.getAvailableValues(); listvalues.setSelection(new Object[] {
"Hong Kong" }); params.put(PriceConstants.ATT_PRICE_LINES_SHIP_FROM,
listvalues);
  //Set Effective From (Date field)
  params.put(PriceConstants.ATT_PRICE_LINES_PRICE_EFFECTIVE_FROM_DATE,
df.parse("10/01/03"));
  //Set Effective To (Date field)
  params.put(PriceConstants.ATT_PRICE_LINES_PRICE_EFFECTIVE_TO_DATE,
df.parse("10/31/03"));
  //Set Quantity (Number field)
params.put(PriceConstants.ATT_PRICE_LINES_QTY, new Integer(1000));
  //Set Currency Code (List field) attr =
cls.getAttribute(PriceConstants.ATT_PRICE_LINES_CURRENCY_CODE);
  listvalues = attr.getAvailableValues();
  listvalues.setSelection(new Object[] { "USD" });
  params.put(PriceConstants.ATT_PRICE_LINES_CURRENCY_CODE, listvalues);

  //Set Total Price (Money field)
  params.put(PriceConstants.ATT_PRICE_LINES_TOTAL_PRICE, new Money(new
Double(52.95), "USD"));
  //Set Total Material Price (Money field)
  params.put(PriceConstants.ATT_PRICE_LINES_TOTAL_MATERIAL_PRICE, new
Money(new Double(45.90), "USD"));
  //Set Total Non-Materials Price (Money field)
  params.put(PriceConstants.ATT_PRICE_LINES_TOTAL_NON_MATERIAL_PRICE,
new Money(new Double(7.05),
```

```
    "USD"));
   //Set Lead Time (Number field)
params.put(PriceConstants.ATT_PRICE_LINES_LEAD_TIME, new Integer(5));
   //Set Transportation Time (List field)
   attr =
cls.getAttribute(PriceConstants.ATT_PRICE_LINES_TRANSPORTATION_TIME);
   listvalues = attr.getAvailableValues();
   listvalues.setSelection(new Object[] { "FOB" });
   params.put(PriceConstants.ATT_PRICE_LINES_TRANSPORTATION_TIME,
listvalues);
   //Set Country of Origin (List field)
   attr =
cls.getAttribute(PriceConstants.ATT_PRICE_LINES_COUNTRY_OF_ORIGIN);
   listvalues = attr.getAvailableValues();
   listvalues.setSelection(new Object[] { "United States" });
   params.put(PriceConstants.ATT_PRICE_LINES_COUNTRY_OF_ORIGIN,
listvalues);
   //Create a new Price Lines row and initialize it with data IRow row =
table.createRow(params);
}
```

Creating a Price Change Order

Price objects such as published prices and contracts have a revision history. If a Price object is released, it can't modified it without first creating a Price Change Order (PCO) and adding the Price object to the Affected Prices table. The PCO is then submitted for approval. Any changes made to the Price object take effect when the PCO completes its Workflow approval process.

A PCO is similar to other Change objects, such as ECOs and ECRs. You can create a PCO using the `IAgileSession.createObject()` method.

Example: Creating a PCO

```
public void createPCO(IPrice price) throws Exception {
   //Get the PCO class
   IAgileClass cls = m_admin.getAgileClass(ChangeConstants.CLASS_PCO);
   //Get autonumber sources for the PCO class
   IAutoNumber[] numbers = cls.getAutoNumberSources();

  //Create the PCO
  IChange pco =
(IChange)m_session.createObject(ChangeConstants.CLASS_PCO, numbers[0]);
   //Get the Affected Prices table
   ITable affectedPrices =
pco.getTable(ChangeConstants.TABLE_AFFECTEDPRICES);

   //Add the Price object to the Affected Prices table
   IRow row = affectedPrices.createRow(price);
}
```

Working with Suppliers

The Agile PLM system comes with five out-of-the-box supplier classes:

- Broker
- Component Manufacturer
- Contract Manufacturer
- Distributor
- Manufacturer Rep

There are two primary key attributes that uniquely identify each supplier: `GENERAL_INFO_NUMBER` and `GENERAL_INFO_NAME`.

Loading a Supplier

To load a supplier, use the `IAgileSession.getObject()` method. To uniquely identify the supplier, specify the General Info | Number attribute.

Example: Loading a supplier

```
public ISupplier getSupplier() throws APIException {
  ISupplier supplier =
(ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE, "SUP20013");
  return supplier;
}
```

Note The Agile API does not support adding new rows to Supplier tables.

Modifying Supplier Data

The Agile API lets you read and update all read/write Supplier fields. For General Info, Page One, and Page Three fields, you can access the cells directly. To access cells on multirow tables like the Contact Users table, you must first load the table and select a particular row.

Example: Modifying supplier data

```
public void updateSupplierGenInfo(ISupplier supplier) throws Exception
{
  ICell cell = null;
  IAgileList listvalues = null;

  //Update Name (Text field)
  cell = supplier.getCell(SupplierConstants.ATT_GENERAL_INFO_NAME);
  cell.setValue("Global Parts");
  //Update URL (Text field)
  cell = supplier.getCell(SupplierConstants.ATT_GENERAL_INFO_URL);
  cell.setValue("http://wwww.globalpartscorp.com");
  //Update Corporate Currency (List field)
```

```
  cell =
supplier.getCell(SupplierConstants.ATT_GENERAL_INFO_CORPORATE_CURRENCY)
;
  listvalues = cell.getAvailableValues();
  listvalues.setSelection(new Object[] { "EUR" });
  cell.setValue(listvalues);
}

public void updateSupplierContactUsers(ISupplier supplier) throws
Exception {
  ICell cell = null;
  IAgileList listvalues = null;

  //Load the Contact Users table
  ITable contactusers =
supplier.getTable(SupplierConstants.TABLE_CONTACTUSERS);
  //Get the first row
  ITwoWayIterator i = contactusers.getTableIterator();
  IRow row = (IRow)i.next();
  //Update Email (Text field)
  cell = row.getCell(SupplierConstants.ATT_CONTACT_USERS_EMAIL);
  cell.setValue("wangsh@globalpartscorp.com");
}
```

Working with Sourcing Projects

A Sourcing Projects is where you prepare content for Sourcing tasks, such as Requests for Quotes (RFQs) and Sourcing analysis. Sourcing Projects is a centralized, collaborative solution. Multiple users can add data to a Sourcing Projects and perform analysis of Sourcing results. Because Sourcing projects serve as the home for all Sourcing activities, they are linked to many classes of objects, including `Supplier`, `RequestForQuote` (RFQ), and `SupplierResponse`.

You can use the Agile API to:

- Load an existing Sourcing Projects
- Create Sourcing Projects by quantity breaks
- Create Sourcing Projects by price periods
- Open and close a Sourcing Projects
- Add items, including AMLs to Sourcing Projects items
- Access and modify objects, tables, and attributes in Sourcing Projects
- Access and modify Sourcing Projects status
- Update Sourcing Projects AMLs
- Update Page 1, Page 2, and Page 3 in Sourcing projects

- Read and update a nested Pricing table in Sourcing projects
- Sets quantity for an item in Sourcing Projects
- Updates the target price for items in Sourcing Projects
- Sets partners for items in Sourcing Projects
- Performs quantity Rollups in Sourcing Projects
- Sets a response designated as *best* in Sourcing Projects

Unlike the Web Client which provides additional functionality for Sourcing projects, the Agile API exposes Sourcing projects for simple data extraction and updating. Consequently, the Agile API does not support the following functions:

- Validation for items, commodities, or manufacturer parts.
- Filter Sourcing Projects tables
- Modify the price scenario for Sourcing Projects (change quantity breaks and effectivity periods)

Supported API Methods

The SDK supports the following API methods for Sourcing projects. For information on these interfaces, rrefer to the Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in SDK_samples (ZIP file). To access this file, see the Note in Client-Side Components on page 2.

- `IAgileSession.createObject(Object, Object)`
- `IAgileSession.createObject(int, Object)`
- `IAgileSession.getObject(Object, Object)`
- `IAgileSession.getObject(int, Object)`
- `IProject.assignSupplier (Object partnerParams)`
- `IProject.Costrollup()`
- `IProject.lookupPrices()`
- `IProject.rollupQuantity()`
- `IProject.getName()`
- `IProject.changeStatusToOpen()`
- `IProject.changeStatusToClose()`
- `IProject.getTable(Object)`
- `IRow.getValue(Object)`
- `IRow.setValue(Object, Object)`
- `ITable.iterator()`
- `ITable.getName()`

- ITable.getTableDescriptor()
- ITable.size()
- ITable.createRow(Object)

Note The PCM SDK does not support the IRow.getReferent() method.

Loading an Existing Sourcing Projects

To load existing Sourcing Projects, use the IAgileSession.getObject() method. To uniquely identify the Sourcing Projects, specify the value for the Cover Page | Number attribute.

Example: Loading Sourcing Projects

```
public IProject getProject() throws APIException {
  String prjnum = "PRJACME_110";
  IProject prj = (IProject)m_session.getObject(IProject.OBJECT_TYPE,
prjnum);
  return prj;
}
```

Creating Sourcing Projects by Quantity Breaks

Defining Sourcing projects uses the generic IAgileSession method.

Example: Creating Sourcing Projects

```
IAgileObject createObject (Object objectType, Object params)
    throws APIException;
```

Creating Sourcing Projects requires specifying one of the following set of parameters:

- Sourcing Projects number and quantity breaks

Or,

- Sourcing Projects number, quantity breaks, and price period information

Note Quantity breaks is a required parameter and is always specified. Example below creates a Sourcing Projects using the quantity break parameter.

Example: Creating Sourcing Projects by quantity breaks

```
IAgileClass agClass =
m_admin.getAgileClass(ProjectConstants.CLASS_SOURCING_PROJECT);
IAutoNumber number = agClass.getAutoNumberSources()[0];
HashMap map = new HashMap();
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER, number);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER_OF_QTY_BREAKS,
new Integer(4));
    IProject prj = (IProject) m_session.createObject(agClass, map);
```

Important Do not pass numbers that are greater than two digits to the QUANTITY_BREAK attribute.

Creating Sourcing Projects by Quantity Breaks and Price Periods

Alternatively, you can create Sourcing Projects by specifying quantity breaks and price period information such as the number of periods, period type, and start date. Example below creates a Sourcing Projects using these parameters.

Note When you create a Sourcing Projects with price period information set to period type, you must specify the `Period Type` attribute. The supported values are Monthly, Quarterly, Semi-Annually, and Yearly. However, `Period Type` is not correctly returned afterwards when you check the value of period type, for example, by invoking `getValue(ProjectConstants.`**`ATT_GENERAL_INFORMATION_PERIOD_TYPE`**`)`. That is, instead of returning the value that you set when creating the Sourcing Projects, the future returned value is always "`Weekly`". This is not an error. It is normal SDK behavior and the specified period type value is not altered, because it is for internal use only.

Example: Creating a Sourcing Projects by quantity breaks and price periods

```
/*
Descriptions
    ATT_GENERAL_INFORMATION_PERIOD_TYPE is described in ProjectConstants
    Name: Period Type
    Description: Period Type indicates the recurrence of price periods
    in a Sourcing Projects.
    Type: List
    List: Period Type List
    List Id: 4565
    List Valid Values: {Monthly, Quarterly, Semi-Annually, Yearly}
    Restrictions: Required, Read Only. Used only when creating Sourcing
    Projects. Internal use only. Not available through Agile UI clients.
    ATT_GENERAL_INFORMATION_PERIOD_START_DATE is described in
    ProjectConstants
    Name: Period Start Date
    Description: Period Start Date indicates the start date for price
    periods in a Sourcing Projects
    Type: Date
    Valid Values: any Date object.
    Restrictions: Required, Read only, Used only when creating Sourcing
    Projects, Internal use only /Not available through Agile UI clients
    */


IAgileClass agClass =
    m_admin.getAgileClass(ProjectConstants.CLASS_SOURCING_PROJECT);
IAutoNumber number =
     agClass.getAutoNumberSources()[0];
HashMap map =
     new HashMap();
    map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER, number);
    map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER_OF_QTY_BREAK
    S, new Integer(4));
    map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER_OF_PERIODS,
    new Integer(4));
IAgileList list =
    agClass.getAttribute(PERIODTYPE).getAvailableValues();
```

```
String TYPE = "Monthly";
list.setSelection(new Object[]{TYPE});
    map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PERIOD_TYPE, list);
    map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PERIOD_START_DATE,
    (new GregorianCalendar()).getTime());
    IProject prj =
      (IProject) m_session.createObject(agClass, map);
```

Accessing and Modifying Objects, Tables, and Attributes

You can use the generic `IDataObject` method with standard calls such as `getObject, getTable, getValue, setValue` to access and subsequently modify objects, tables, and attributes as follows:

- Read Page 1 or Cover Page, Page 2, Page 3, Items, AML, Analysis, and nested pricing tables
- Update Page 1 or Cover Page, Page 2, Page 3, and AML tables
- Add items including AML to Items table
- Read RFQ table
- Load RFQ table

The `com.agile.api.ProjectConstants.java` file contains information about classes, tables, and attributes.

The PCM does not support the following table operations:

- Sourcing Projects class
  - Sorting PCM specific tables that have a default sorting order. These tables are Sourcing Project Item, Sourcing Project AML, Sourcing Project Changes, Sourcing Project Analysis, and Sourcing Project RFQ, RFQ Response, RFQ Changes, Supplier Response, and Supplier Changes.
- Request for quote and RFQ responses classes
  - Responses and Changes tables
- Removing items from RFQ Response and Sourcing Projects tables because the PCM SDK does not support `ITable.clear()` or `ITable.removeRow()`

Setting Cover Page Values for Sourcing Projects

You can read and update all read/write Sourcing Projects cells. The following example updates the cells on a Sourcing project's Cover Page (Page 1).

Example: Setting values for a Sourcing project's Cover Page

```
public void updateProjectGenInfo (IProject project) throws Exception {

    ICell cell = null;
    IAgileList listvalues = null;

   //Update Customer (List field)
cell =
    project.getCell(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER);
```

```
listvalues =
    cell.getAvailableValues();
    listvalues.setSelection(new Object[] { "CUST00010" });
    cell.setValue(listvalues);
  //Update Description (Text field)
cell =
    project.getCell(ProjectConstants.ATT_GENERAL_INFORMATION_DESCRIPTION
    );
    cell.setValue("Sourcing Projects for Odyssey III");
  //Update Manufacturing Site (List field)
cell =
    project.getCell(ProjectConstants.ATT_GENERAL_INFORMATION_MANUFACTURI
    NG_SITE);
listvalues =
    cell.getAvailableValues();
    listvalues.setSelection(new Object[] { "Global" });
    cell.setValue(listvalues);
  //Update Ship To Location (List field)
cell =
    project.getCell(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOC
    ATION);
listvalues =
    cell.getAvailableValues();
    listvalues.setSelection(new Object[] { "San Jose" });
  cell.setValue(listvalues);
}
```

Understanding Nested Tables in PCM

A nested table is a table within a table. They are used to accesses and modify data in multi-level objects such as BOMs and Items with AMLs. The way the SDK fulfills this function is to treat the cell values in a nested table as a table. For example, when the SDK finds the next level in a cell in a BOM table, it treats and processes the cell as a table. Nested tables are unique to the PCM SDK.

Sourcing Projects' Parent Table and Nested Child Table Constants

The list of parent Sourcing Projects table and the corresponding nested child table constants appear in the Parent Sourcing Projects Table Constants and the Corresponding Nested Sourcing Projects Tables.

Parent Table Constant	Nested Child Table Constant	Read/Write Mode
TABLE_ITEMS	ATT_ITEMS_AML	Read/Write
TABLE_ITEMS	ATT_ITEMS_PRICING	Read/Write
TABLE_AML	ATT_AML_PRICETABLE	Read/Write
TABLE_ITEM	ATT_ITEM_PRICE_TABLE	Read/Write
TABLE_ITEM	ATT_ITEM_BOM_TABLE	Read
TABLE_ANALYSIS	ATT_ANALYSIS_AML	Read

Parent Table Constant	Nested Child Table Constant	Read/Write Mode
TABLE_ANALYSIS	ATT_ANALYSIS_PRICING	Read

Accessing and Modifying Nested Tables in Sourcing Projects or RFQ

Example below is a Read example that accesses a nested table. To modify/update a nested table, see the Example entitled "Nested RFQ table update" in RFQ Parent Table and Nested Child Table Constants on page 282.

Note The Money type attribute in nested PCM Pricing tables always use the “USD” as the default currency unit. This applies even if the buyer specifies a different currency unit. In this case, the “United State Dollar” is the default and only supported currency.

Example: Accessing a nested table

```
Row row = (IRow) table.iterator.next();
ITable nested_table =
    (ITable)row.getValue(ProjectConstants.ATT_ITEMS_AML);
```

Viewing Updates after Modifying a Nested Table

After modifying a nested table, it is necessary to reload the table as in the following example for changes to take effect. If you only reiterate the table, as in example 14-14, the old data will reappear and the new values are not displayed.

Example: Reiterating a nested table

```
/*
 * In nested AML table, make modifications.
 * For example, insert a row, assign suppliers
 *
 */
row.getValue(attribute);
```

Accessing and Modifying the Status of Sourcing Projects

Because Sourcing projects do not have a Workflow connected to them, their status change is controlled internally. They control their status with a set of methods. This is a special case for some PCM objects such as Sourcing Projects and Request for Quote. This release supports changing the status of a Sourcing Projects from Draft to Open and Open to Close.

You can access the status of Sourcing Projects using the standard `IDataObject` method for the lifecycle phase field on the Cover Page (Page 1). You can modify the status of Sourcing Projects with `IProject` methods which enables you to open, modify, and close Sourcing Projects. You must set the ship to location parameter to open Sourcing Projects as shown in the following example.

Example: Setting values for a Sourcing project’s Cover Page

```
// add Ship To //
String sj = “San Jose”;
IAgileList ship2List =
    (IAgileList)prj.getValue(ProjectConstants.ATT_GENERAL_INFORMATION_SH
    IP_TO_LOCATION);
```

```
ship2List.setSelection(new Object[]{sj});
prj.setValue(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,
ship2List);
// open Sourcing Project //
prj.changeStatusToOpen();

// close Sourcing Project //
prj.changeStatusToClose();
```

Managing Data in Sourcing Projects

The following paragraphs provide descriptions and examples to prepare a Souring project to issue an RFQ. You can then use the SDK to complete the RFQ-related tasks.

Note The *Sourcing Projects Start* date that you specify is converted to the GMT format for storage in the PLM database. Due to this conversion, the date value returned by `IProject.getValue(ProjectConstants.ATT_GENERAL_INFORMATION_PERIOD_START_DATE)` is not guaranteed to be the same that the user may expect.

Setting Quantity for Items in Sourcing Projects

You can use the SDK to set the required quantity for the Item object in Sourcing projects. The code sample below sets this value in the Item table, under the Items tab for a single price target. The end user can specify the target price using the displayed name which is `QuantityBreak2` in the following example.

Example: Setting quantity for Items

```
// Setting Quantity for an Item //
ITable tab_item = dObj.getTable(ProjectConstants.TABLE_ITEM);
IRow row = (IRow) tab_item.iterator().next()
ITable priceTable =
row.getValue(ProjectConstants.ATT_ITEM_PRICE_TABLE);
for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
    IRow row = (IRow) iterator.next();
    String name = row.getName();
    if(name.equals("QuantityBreak2")){
          row.setValue(ProjectConstants.ATT_PRICEDETAILS_QUANTITY, new
Double(123));
    }
}
```

Note For items, quantity is only set at the root level. Thus, if an item is not a root, the exception: `ExceptionConstants.PCM_PROJECT_ITEM_IS_NOT_ROOT` is thrown.

In addition, because `priceTable` is a nested table, you must reload the table to get the updated value of Quantity. This is shown in the following example.

Example: Reloading a nested table to get an updated value

```
// Getting the updated value //
priceTable = row.getValue(ProjectConstants.ATT_ITEM_PRICE_TABLE);
```

```
for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
     IRow iRow = (IRow) iterator.next();
     String name = iRow.getName();
     if(name.equals("QuantityBreak2")){
        Object qty =
row.getValue(ProjectConstants.ATT_PRICEDETAILS_QUANTITY));
     }
}
```

Performing Quantity Rollup in Sourcing Projects

Quantity rollups generate data related the quantity values for the selected Item in a Sourcing Projects. In the SDK, you can use the following API to invoke a Quantity rollup in Sourcing Projects.

```
public void rollupQuantity() throws APIException, RemoteException,
Exception;
```

This code sample uses `rollupQuantity()` to do a Quantity rollup.

Example: Quantity Rollup

```
IProject prj =
       (IProject)m_session.getObject(ProjectConstants.CLASS_SOURCING_PRO
       JECT,"PRJ00001");
prj.rollupQuantity();
```

Note To get the updated value of Quantity, it is necessary to invoke `rollupQuantity()` on the Sourcing Project similar to the example in Modifying Spplier Data on page 261. This is necessary because `getValue()` does not return the updated value of the affected item after setting Quantity.

Performing Cost Rollup in Sourcing Projects

Cost Rollup (Rollup cost) generates an Assembly Cost Report (ACR) based on available prices. In this process, it picks up the lowest costs from filtered data, performs Set as Best (on user defined or default parameters) and costed BOM rollup (aggregation) to generate the ACR. In the UI, Rollup cost provides an intuitive mechanism for non PCM users to cost a BOM without going through PCM steps.

Note Cost Rollup runs on existing Sourcing Projects prices. If Cost Rollup needs to run on looked up prices, `lookupPrices()` must be invoked prior to running `costRollup()`. If there are no assemblies in Sourcing Projects, the `ExceptionConstants.PCM_NO_ASSEMBLY_IN_PROJECT` is thrown.

The PCM SDK supports the Cost Rollup function with the following API.

```
public void costRollup()
throws APIException, RemoteException, Exception;
```

Example: Using the costRollup API

```
IProject prj = (IProject)
m_session.getObject(ProjectConstants.CLASS_SOURCING_PROJECT,
"PRJ0001");
prj.costRollup();
```

Note If you need to run quantity rollup immediately after cost rollup, be sure to provide some delay (For example as in `Thread.currentThread().sleep(10000);`) to allow the results of the cost rollup to be refreshed in the database.

Performing Price Lookup in Sourcing Projects

You can use the SDK to verify the existence of a price scenario for a specified period and quantity in the Item Master. You can either use the price information of the Item, or modify the price information and send the RFQ to suppliers for requote.

In Agile PCM, there are three types of price objects:

- Contracts – Predefined agreements with suppliers for Item prices over a specified time period
- Published Prices – The Item price information that has been published from other Sourcing projects
- Quote Histories – Quoted prices that were previously received for an Item

For information about price objects and price lookups in Sourcing projects,, refer to the Agile PLM *Product Cost Management User Guide*.

Price Lookup API and Price Lookup Options

- Supported API

  The SDK supports price lookups with the following API in `IProject`.

  ```
  public void lookupPrices(Object lookupParams)
  throws APIException, RemoteException, Exception;
  ```

- Price lookup options

  This API performs price lookups from Price history and Price lookups from another Sourcing Projects.

  Note `lookupPrices()` looks for an Item or an MPN one object at a time. To run lookup for multiple Items/MPNs, you must run the API one Item or one MPN at a time.

The following examples show price lookups from the Sourcing Projects History and from another Sourcing Projects. In addition, applicable parameters are grouped and listed as those that are specific to the price lookup type and necessary in the price lookup type.

Parameters for Price Lookup from History or Another Sourcing Projects

This example shows a price lookup from Sourcing Projects History and from another Sourcing Projects. It provides a list of specific and required parameters for the two price lookups.

Example: Price lookup from History and another Sourcing Projects

```
ArrayList priceTypes = new ArayList();
priceTypes.add(PriceConstants.CLASS_PUBLISHED_PRICE);
priceTypes.add(PriceConstants.CLASS_QUOTE_HISTORY);
priceTypes.add(PriceConstants.CLASS_CONTRACT);

ArrayList suppliers = new ArrayList();
```

```
suppliers.add(supplier1);
suppliers.add(supplier2);
//supplier1, supplier2 are objects of ISupplier or String

ArrayList customers = new ArrayList();
customers.add(customer1);
customers.add(customer2);
//customer1, customer2 are objects of ICustomer or String

ArrayList programs = new ArrayList();
programs.add(program1);
programs.add(program2);
//program1, program2 are objects of String

String shipTo = "berlin";
HashMap itemMap = new HashMap();
itemMap.put("IPN1","REV1");//itemMap.put("IPN1", null) if no revision
or
itemMap.put(item); //item is an object of IItem

HashMap mpnMap = new HashMap();
mpnMap.put("MPN1","MFR1");
or
mpnMap.put(mfrPart); //mfrPart is and object of IManufacturerPart

IProject srcPrj = (IProject)
    m_session.getObject(ProjectConstants.CLASS_SOURCING_PROJECT,
    "PRJ_SRC");
Boolean isLookupFromPrice = new Boolean(false);
String priceScenario = null;
Map priceScenarios = new HashMap();
//if lookup from price history
priceScenario = "QuantityBreak1";
//if lookup from Sourcing project
String destPricePoint1 = "QuantityBreak1";
String destPricePoint2 = "QuantityBreak2";
String srcPricePoint1 = "QuantityBreak1";
String srcPricePoint2 = "QuantityBreak2";
priceScenarios.put(destPricePoint2,srcPricePoint1);
priceScenarios.put(destPricePoint1,srcPricePoint2);

Boolean ignoreQtyRange = new Boolean(true);
Double qtyPercentRange = new Double(15);

Boolean ignoreDateRange = new Boolean(true);
Integer dateRange = new Integer(20);
```

```
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipT
o);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER,srcPrj);
map.put(LookupConstants.FLAG_IGNORE_ITEM_REVISION,ignoreItemRev);
map.put(LookupConstants.FLAG_CONSIDER_BEST_PRICES,considerBestPrices);
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FIELD_PRICE_SCENARIOS,priceScenarios);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.FIELD_SELECT_RESPONSE_BY,
   LookupConstants.OPTION_LOWEST_PRICE);
map.put(LookupConstants.FIELD_LOOKUP_TYPE,
   LookupConstants.OPTION_LOOKUP_FROM_PRICE);
prj.lookupPrices(map);
```

Parameters specific to price lookups from price history

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER
ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM
ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION
LookupConstants.FLAG_ALL_PRICE_SCENARIOS
LookupConstants.FIELD_PRICE_SCENARIO
LookupConstants.FLAG_IGNORE_QUANTITY
LookupConstants.FIELD_QUANTITY_RANGE
LookupConstants.FLAG_IGNORE_DATE_RANGE
LookupConstants.FIELD_DATE_RANGE
LookupConstants.FIELD_SELECT_RESPONSE_BY
```

Parameters specific to price lookups from Sourcing Projects

```
ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER
LookupConstants.FLAG_ALL_PRICE_SCENARIOS
LookupConstants.FLAG_IGNORE_ITEM_REVISION
LookupConstants.FLAG_CONSIDER_BEST_PRICES
```

Note The remaining parameters are common to both cases.

Parameters required for price lookups from price history

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
LookupConstants.FIELD_QUANTITY_RANGE if
LookupConstants.FLAG_IGNORE_QUANTITY is 'false'
LookupConstants.FIELD_DATE_RANGE if
LookupConstants.FLAG_IGNORE_DATE_RANGE is 'false'
LookupConstants.FIELD_PRICE_SCENARIO if
LookupConstants.FLAG_ALL_PRICE_SCENARIOS is 'false'
```

Parameters required for price lookups from Sourcing Projects

```
ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER
LookupConstants.FLAG_ALL_PRICE_SCENARIOS
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
```

Note Parameters that not required in one of the price lookups, for example, History, may be optional in price lookup from another Sourcing Projects. Comparing the list of required parameters above, the `LookupConstants.FLAG_ALL_PRICE_SCENARIOS` parameter is optional when performing a price lookup from History. You can either omit the optional parameters, or set them to `null`.

Setting the price lookup from History or Sourcing Projects

Set `LookupConstants.FIELD_LOOKUP_TYPE` for lookup from history or Sourcing Projects as follows:

- For lookup from price history – `LookupConstants.OPTION_LOOKUP_FROM_PRICE`
- For lookup from an existing Sourcing Project – `LookupConstants.OPTION_LOOKUP_FROM_PROJECT`

Settings for Quantity Breaks in price lookups

You can set quantity breaks in a price lookup by cost, date, or leadtime by setting `LookupConstants.FIELD_SELECT_RESPONSE_BY` as follows:

- For break tie by cost – `LookupConstants.OPTION_LOWEST_PRICE`
- For break tie by date – `LookupConstants.OPTION_MOST_RECENT_RESPONSE`
- For break tie by leadtime– `LookupConstants.OPTION_SHORTEST_LEAD_TIME`

Impact of improper parameter settings

If the following parameters are not set, or are improperly set, the API will take the following actions:

- `LookupConstants.FIELD_LOOKUP_TYPE` will default to `LookupConstants.OPTION_LOOKUP_FROM_PRICE` which corresponds to the lookup from price history
- `LookupConstants.FLAG_IGNORE_QUANTITY` or `LookupConstants.FLAG_IGNORE_DATE_RANGE` will default to `true`.

- `LookupConstants.FIELD_SELECT_RESPONSE_BY` will default to `LookupConstants.OPTION_LOWEST_PRICE` which corresponds to the break tie by cost
- `LookupConstants.FLAG_IGNORE_ITEM_REVISION` or `LookupConstants.FLAG_CONSIDER_BEST_PRICES` will default to `false`
- `LookupConstants.LookupConstants.FLAG_ALL_PRICE_SCENARIOS` is not set it will be defaulted to 'true'.
- `ExceptionConstants.APDM_ADMIN_MISSINGREQUIREDFIELD` exception is thrown when a required parameter is missing
- `ExceptionConstants.API_INVALID_PARAM` exception is thrown when the datatype, or the value of a parameter is incorrectly set

For RFQ lookup:

The settings are similar to Sourcing Projects lookup from price history. Following is a code sample.

```
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipT
o);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.FIELD_SELECT_RESPONSE_BY,LookupConstants.OPTION
_LOWEST_PRICE);
map.put(LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER,excludeAuthSupplier)
;
rfq.lookupPrices(map);
```

Following is the list of the required parameters for the RFQ lookup:

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
LookupConstants.FIELD_QUANTITY_RANGE if
LookupConstants.FLAG_IGNORE_QUANTITY is 'false'
LookupConstants.FIELD_DATE_RANGE if
LookupConstants.FLAG_IGNORE_DATE_RANGE is 'false'
LookupConstants.FIELD_PRICE_SCENARIO if
LookupConstants.FLAG_ALL_PRICE_SCENARIOS is 'false'
```

Note If you do not set the value of `LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER`, it will default to false.

Setting Partners in a Sourcing Projects

Partners can view complete Sourcing Projects BOMs in RFQs. You can assign partners to an item in the Sourcing Projects when you add the item to the RFQ that will be sent to the partners. If multiple partners are selected, you can split the quantity among the partners by specifying what percentage of an item you want to receive from each supplier. For example, if two partners supply the same item, you can add both partners to the list and then assign a certain percentage to each, for example, 50%-50%, or 60%-40%, and so on.

In the SDK, the following API is used to set partners for an item in Sourcing Projects and split the percentages among the partners.

```
public void assignSupplier(Object partnerParams) throws APIException,
RemoteException, Exception;
```

The behavior of this API and its use cases are similar to `IRequestForQuote.assignSupplier()`. However, when you add new partners for an item with this API, you will override the existing ones. Thus, to avoid removing existing partners, it is necessary to once again add the existing partners and set the split (Percentage for each) level for each one. This only occurs in the SDK and the GUI does not require adding the existing partners when you add new partners. You can't remove a partner for an item, but assigning a `split = 0`, (Percentage of ownership/participation) will remove the partner. For more information on the GUI behavior, refer to the latest release of *Agile Product Lifecycle Management - Product Cost Management Supplier Guide*.

The following code sample sets partners and splits the percentages among the assigned partners.

Example: Setting partners and splitting percentages among partners

```
HashMap map = new HashMap();
HashMap supplierSplit = new HashMap();
HashMap partnerMap = new HashMap();

map.put(ProjectConstants.ATT_ITEM_NUMBER, item);
Double split1 =
   new Double(55);
Double spl1it2 =
   new Double(75);
supplierSplit.put(supplier1, split1);
supplierSplit.put(supplier2, split2);
partnerMap.put(ProjectConstants.ATT_PARTNERS_PARTNER, supplierSplit);
map.put(ProjectConstants.ATT_ITEM_PARTNER_TABLE, partnerMap);
prj.assignSupplier(map);
```

An `item` or `supplier` can be an `IItem` object, a `ISupplier` object, or a `String` object. Partners can be assigned to any Item Part Number (IPN), but not a Manufacturer Part Number (MPN). If the item is not in Sourcing Projects, the `ExceptionConstants.PCM_ERROR_INVALID_PROJECT_ITEM` is thrown.

Split percentages can be any object representing a number. If it is not a number, the `ExceptionConstants.API_INVALID_PARAM` exception is thrown.

To get data on a given partner, you can use the Items or AML tabs as shown in .

Example: Getting partner data using Item or AML

```
ITable tab_item =
   prj.getTable(ProjectConstants.TABLE_ITEMS);
IRow row = (
   IRow) tab_item.iterator().next();
ITable partnerTable =
   (ITable) row.getValue(ProjectConstants.ATT_ITEMS_PARTNERS);
```

Or,

```
ITable tab_item =
   prj.getTable(ProjectConstants.TABLE_ITEM);
IRow row =
   (IRow) tab_item.iterator().next();
ITable partnerTable =
   (ITable) row.getValue(ProjectConstants.ATT_ITEM_PARTNER_TABLE);
for (Iterator iterator =
   partnerTable.iterator(); iterator.hasNext();) {
IRow iRow =
   (IRow) iterator.next();
String partner =
   iRow.getValue(ProjectConstants.ATT_PARTNERS_PARTNER).toString();
String split =
   iRow.getValue(ProjectConstants.ATT_PARTNERS_PARTNER_SPLIT).toString(
   );
}
```

Modifying the Target Price for Items in Sourcing Projects

Target Price is the market cost per unit of the item or the manufacturer part. It is specified when items are ordered. For each Item, For each Pricepoint, Target Price is set in the Items table, under the AML tab. A Pricepoint is the Target price quoted for a given quantity for an Item. For example, price quoted for X number of tires, which can be different for Y number of the same tires.

Note The Target price is always a positive number. Setting a negative value for Target price will throw the `ExceptionConstants.PCM_NEGATIVE_TARGET_PRICE` exception.

Target Price are set at Item level only. You can't set a Target Price the AML level. The end user specifies a Pricepoint using the name displayed for the Pricepoint. For example, `QuantityBreak2` in the following example.

Example: Setting the Target price in Sourcing Projects

```
ITable tab_item = dObj.getTable(ProjectConstants.TABLE_ITEMS);
IRow row = (IRow) tab_item.iterator().next();
ITable priceTable = row.getValue(ProjectConstants.ATT_ITEMS_PRICING);
for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
    IRow row = (IRow) iterator.next();
    String name = row.getName();
```

```
    if(name.equals("QuantityBreak2")){
            row.setValue(ProjectConstants.ATT_PRICEDETAILS_TARGET_COST,
                 new Money(new Double(1.23), "USD") );
    }
}
```

Because `priceTable` is a nested table, you must reload this table to get the updated value of the Target price as shown in the following example. This is similar to the example in Setting Quantity for Items in Sourcing Projects on page 269.

Example: Reloading the priceTable to get the updated value of the target price

```
priceTable = row.getValue(ProjectConstants. ATT_ITEMS_PRICING);
for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
     IRow iRow = (IRow) iterator.next();
     String name = iRow.getName();
     if(name.equals("QuantityBreak2")){
        Object qty =
row.getValue(ProjectConstants.ATT_PRICEDETAILS_TARGET_COST));
     }
}
```

Setting the Best Response for Items in Sourcing Projects

The Best Response is set in the Analysis table under the Analysis tab for both the Item and Manufacturer Part number objects. The end user specifies three of these parameters: Lowest Cost, Lowest Cost Within Lead Time Constraint, Shortest Lead Time, Supplier Rating, and AML Preferred status. For more information, refer to *Agile Product Lifecycle Management - Product Cost Management Supplier Guide*.

You can use the SDK to find the best response for an Item Part Number (IPN), a Manufacturer Part Number (MPN), and for an IPN and an MPN as shown in the following code samples.

Example: Setting the Best Response for an IPN

```
//set best response for ipn //
ITable table_analysis = prj.getTable(ProjectConstants.TABLE_ANALYSIS);
Iterator it = table_analysis.iterator();
while(it.hasNext()) {
      IRow row = (IRow) it.next();
      String itemName =
row.getValue(ProjectConstants.ATT_ANALYSIS_NUMBER);
      String suppName =
row.getValue(ProjectConstants.ATT_ANALYSIS_SUPPLIER);
      if (itemName.equals("IPN1") && suppName.equals("suppName1
(suppNumber1)")) {
               row.setValue(ProjectConstants.ATT_ANALYSIS_BEST_RESPONSE
, "Yes");
      }
}
```

Example: Setting the Best Response for an MPN

```
ITable table_aml =
   (ITable) row.getValue(ProjectConstants.ATT_ANALYSIS_AML);
Iterator _it =
```

```
    table_aml.iterator();
   while(it.hasNext()){
    String itemName =
    row.getValue(ProjectConstants.ATT_ANALYSIS_NUMBER);
    String suppName =
    row.getValue(ProjectConstants.ATT_ANALYSIS_SUPPLIER);
    String mfrName = row.getValue(ProjectConstants.
    ATT_ANALYSIS_MANUFACTURER);
 if (itemName.equals("MPN1") && suppName.equals("suppName1
(suppNumber1)"
      && mfrName.equals("MFR1"))) {
      row.setValue(ProjectConstants.ATT_ANALYSIS_BEST_RESPONSE, "Yes");
      }
}
```

Note Because you can only set the Best Response to `Yes`, if you pass any value other than `Yes`, the `ExceptionConstants.API_INVALID_PARAM` exception is thrown.

Example: Getting the Best Response for an IPN and an MPN

```
String bResp =
row.getValue(ProjectConstants.ATT_ANALYSIS_BEST_RESPONSE).toString();
```

Working with RFQs

Requests for Quotes (RFQs) allow users to request pricing information from suppliers. RFQs serve as the instrument to negotiate pricing and terms for items or manufacturer parts. RFQs are defined for Sourcing projects. Thus, to define an RFQ, you must first create the Sourcing Projects and then create the required RFQs for that Sourcing Projects.

A single Sourcing Projects can generate several RFQs. RFQs support a one-to-many relationship with suppliers. That is, one RFQ may generate several responses from suppliers.

The Agile API supports the following RFQ-related tasks.

- Create an RFQ for Sourcing Projects
- Load and modify RFQ objects, tables, and attributes
- Access and modify Page 1, Page 2, and RFQ Response table
- Add items to RFQ Response tables from the RFQ's Sourcing Projects
- Read and update nested tables in Page 1, Page 2, and RFQ response table
- Assign supplier to items or manufacturer parts in RFQ response table

For a list of API methods that support these RFQ functions, see Supported API Methods on page 280.

Note The PCM SDK RFQ objects do not have a Page three and no Page three RFQ constant is supported. Do not invoke these constants because the RFQ will not produce the expected result.

Supported API Methods

The SDK supports the following APIs for RFQs. For information on these interfaces, For information on these interfaces, rrefer to the Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in SDK_samples (ZIP file). To access this file, see the Note in Client-Side Components on page 2.

- `IAgileSession.createObject(Object, Object)`
- `IAgileSession.createObject(int, Object)`
- `IAgileSession.getObject(Object, Object)`
- `IAgileSession.getObject(int, Object)`
- `IRequestForQuote.getName()`
- `IRequestForQuote.assignSupplier(Object)`
- `IRequestForQuote.getTable(Object)`
- `IRequestForQuote.lookupPrices(Object)`
- `ITable.iterator()`
- `ITable.getTableDescriptor()`
- `ITable.size()`
- `ITable.createRow(Object)`
- `IRow.getValue(Object)`
- `IRow.setValue(Object, Object)`

Creating RFQs for Sourcing Projects

RFQs are defined for a specific Sourcing Projects. Creating an RFQ uses the generic `IAgileSession` method.

Similar to Sourcing projects (see Creating RFQs for Sourcing Projects on page 280), you can use `IDataObject` with standard calls such as `getObject`, `getTable`, `getValue`, `setValue` to access and modify objects, tables, and attributes as follows:

- Read Page 1 or Cover Page and Page 2 tables
- Update Page 1 or Cover Page and Page 2 tables

Example: Creating an object

```
IAgileObject createObject(Object objectType, Object params)
throws APIException;
```

To create an RFQ, you must open the Sourcing Projects. However, to open the Sourcing Projects, it is necessary to first set the ship to location. See the code example in Accessing and Modifying the Status of Sourcing Projects on page 268.

You cannot create an RFQ by specifying the Sourcing Projects number only. You must also specify the related Sourcing Projects as this is a required parameter. This is shown in the example below.

Example: Creating an RFQ for Sourcing Projects

```
IAgileClass rfqClass =
    m_admin.getAgileClass(RequestForQuoteConstants.CLASS_RFQ);
    IAutoNumber rfqNumber = rfqClass.getAutoNumberSources()[0];
    HashMap map = new HashMap();
map.put(RequestForQuoteConstants.ATT_COVERPAGE_RFQ_NUMBER, rfqNumber);
map.put(RequestForQuoteConstants.ATT_COVERPAGE_PROJECT_NUMBER,
pnumber);
    IRequestForQuote rfq =
        (IRequestForQuote) m_session.createObject(rfqClass, map);
```

Loading Existing RFQs

You can load an existing RFQ using the `IAgileSession.getObject()` method, or select it from the RFQ table of Sourcing Projects object.

To load an RFQ, use the `IAgileSession.getObject()` method. To uniquely identify an RFQ, specify the value for the Cover Page | RFQ Number attribute.

Example: Loading an RFQ

```
public IRequestForQuote getRFQ() throws APIException {
IRequestForQuote rfq =
    (IRequestForQuote)m_session.getObject(IRequestForQuote.OBJECT_TYPE,
    "RFQ01004");
 return rfq;
}
```

Loading RFQs from Sourcing Projects' RFQ Table

In addition to loading an RFQ using `IAgileSession.getObject()`, you can also select an RFQ from the RFQ table of Sourcing Projects object.

Example: Loading an RFQ from the Sourcing Projects RFQs table

```
ITable table = prj.getTable(ProjectConstants.TABLE_RFQS);
Iterator it = table.iterator();
IRow row1 = (IRow) it.next();
IDataObject obj1 = (IDataObject)
m_session.getObject(IRequestForQuote.OBJECT_TYPE,
         row1.getValue(ProjectConstants.ATT_RFQS_RFQ_NUMBER));
```

Note The `getReferent()` method does not support the PCM SDK, including the RFQ tables. A list of supported RFQ tables appears in the table below.

Supported RFQ Tables

The supported RFQ tables and their respective constants are listed in the table below.

Table	Constant	Read/Write Mode
Cover Page	TABLE_COVERPAGE	Read/Write
Page Two	TABLE_PAGETWO	Read/Write
Responses	TABLE_RESPONSES	Read/Write

Note The Agile API does not support adding new rows to RFQ tables. However, you can add new rows to the RFQ response table.

Accessing and Modifying RFQ Objects, Tables, Nested Tables, and Attributes

You can access RFQ objects, tables, and attributes using the generic `IAgileSession` and `IDataObject` methods and standard calls such as `getObject`, `getValue`, `setValue`. Information about these classes, tables, and attributes is provided in the `com.agile.api.RequestForQuoteConstants.java` file.

RFQ Parent Table and Nested Child Table Constants

The list of parent RFQ table and the corresponding nested child table constants appears in the table below.

Parent Table Constant	Nested Child Table Constant	Read/Write Mode
TABLE_RESPONSES	ATT_RESPONSES_AML	Read/Write
TABLE_RESPONSES	ATT_RESPONSES_PRICING	Read/Write

Similar to Sourcing Projects, nested RFQ tables are accessed by treating their cell value as a table. See Accessing and Modifying Nested Tables in Sourcing Projects or RFQ on page 268. The following examples updates a nested table.

Note Do not use `Project.ATT_RFQ_RFQ_STATE` to get the status of an RFQ, because it is not exposed to the SDK and will not render the correct value of the RFQ row. To get the status of an RFQ, you must first load the RFQ, and then get the status from the RFQ itself.

Example: Nested RFQ table update

```
ITable subtab1 =
    (ITable)row.getValue(RequestForQuoteConstants.ATT_RESPONSES_PRICING)
    ;
IRow pricing1 =
    (IRow)subtab1.iterator().next();
Integer nest =
    ProjectConstants.ATT_PRICEDETAILS_MATERIAL_PRICE;
Object nre =
    pricing1.getValue(nest);
Money tc =
    new Money(new Integer(100), "USD");
pricing1.setValue(nest, (Object)tc);
```

Note You must assign the supplier before updating an RFQ response table entry.

Performing Price Lookup in RFQs

Similar to Performing Price Lookup in a Sourcing Project on page 271, you can verify the existence of a price scenario for a specified period and quantity for RFQs. If they exist, you do not have to create an RFQ for the specified item. You can either use the price information of the item, or you

can modify the price information and send the RFQ to suppliers for requote. This is shown in the following code sample.

Example: Price lookup from history and from another Sourcing Projects

```
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.
   ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipTo);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.
   FIELD_SELECT_RESPONSE_BY,LookupConstants.OPTION_LOWEST_PRICE);
map.put(LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER,excludeAuthSupplier)
;
rfq.lookupPrices(map);
```

Required parameters for an RFQ price lookup

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
LookupConstants.FIELD_QUANTITY_RANGE if
LookupConstants.FLAG_IGNORE_QUANTITY is ‘false’
LookupConstants.FIELD_DATE_RANGE if
LookupConstants.FLAG_IGNORE_DATE_RANGE is ‘false’
LookupConstants.FIELD_PRICE_SCENARIO if
LookupConstants.FLAG_ALL_PRICE_SCENARIOS is ‘false’
```

Impact of Improper parameter settings

`LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER` will default to `false` if not set

Working with RFQ Responses

The PCM SDK supports the following operations for RFQ responses, nested table of items responses, and Child AML responses:

- Read RFQ tables with different views (price scenarios, currency modes)

  This is supported through generic SDK API.

- Add items to RFQs
- Add Response lines (Assign Suppliers)

PCM RFQ provides the following method for assigning suppliers to items or manufacturer parts.

```
public void assignSupplier(Object supplierParams)
    throws APIException, RemoteException, Exception;
```

You can assign supplier data such as Manufacturer Part Number (mpn) or supplier name as a String or an Object to the RFQ response.

Example: Adding supplier data as String constants

```
IRequestForQuote dObj =
    (IRequestForQuote)m_session.getObject(RequestForQuoteConstants.CLASS
    _RFQ, number);
ITable tab =
    dObj.getTable(RequestForQuoteConstants.TABLE_RESPONSES);
Map mp = new HashMap();
    mp.put(ProjectConstants.ATT_RESPONSES_NUMBER, "P00007");
    mp.put(ProjectConstants.ATT_RESPONSES_SUPPLIER, "SDKSUP");
    dObj.assignSupplier(mp);
```

Example: Adding supplier data as Objects

You can also add an item as an IItem object as shown below.

```
mp.put(ProjectConstants.ATT_RESPONSES_NUMBER, itemObject);
```

If you are assigning supplier to an mpn, you must specify the mpn as an IManufacturerPart Object, or as a pair of Objects. One for the mpn name and one for mfr name.

```
mp.put(RequestForQuoteConstants.ATT_RESPONSES_NUMBER, mpnObject);
```

Or,

```
mp.put(RequestForQuoteConstants.ATT_RESPONSES_NUMBER, mpnName);
mp.put(RequestForQuoteConstants.ATT_RESPONSES_MANUFACTURER, mfrName);
```

Caution When you invoke RequestForQuote.TABLE_RESPONSE to assign suppliers to item components, the table size may change if there is more than one supplier for that item component. That is, if an item has a single supplier, each item and the corresponding supplier will occupy their own distinct separate rows in the TABLE_RESPONSE table. However, if the item has more than one supplier, then the row for this item component is split into the number of suppliers, thus changing TABLE_RESPONSE by increasing the number of rows in the table. It is therefore necessary to immediately reload the ITERATOR to reflect the change in TABLE_RESPONSE table. This is not an SDK defect and is due to SUN J2SE ITERATOR behavior.

- Update Response Lines

  The PCM SDK supports the RFQ response table through generic SDK API. It does not support the RFQ Response Class or Supplier Response.

Note The response currency in the RFQ response line is determined by the response currency attribute. This causes the server to ignore the currency parameter in the material price. Buyers can modify the response currency in the response line and it will be applied to all pricing attributes in the response line. The supplier RFQ response currency is set to your RFQ currency preference and cannot be modified in the supplier responses. Once the response line is opened to suppliers, the response line must be locked before buyers can modify them.

Chapter 18

Managing Product Governance & Compliance

This chapter includes the following:

- About Agile Product Governance and Compliance 287
- Agile PG&C Interfaces and Classes 288
- Agile PG&C Roles 288
- Creating Declarations, Specifications, and Substances 289
- Adding Items, Manufacturer Parts, and Part Groups to Declarations 293
- Adding Substances to Declarations 294
- Adding Substances to a Specification 300
- Adding Specifications to a Declaration 301
- Routing Declarations 302
- Completing a Declaration 303
- Submitting Declarations to Compliance Managers 304
- Publishing a Declaration 304
- Getting and Setting Weight Values 305
- Adding Substance Compositions for Manufacturer Parts 306
- Rolling Up Compliance Data 308

About Agile Product Governance and Compliance

Agile Product Governance & Compliance (PG&C) addresses the growing number of environmental regulations and corporate environmental policies that impact product definition and the import, export, and disposal of restricted substances. Agile PG&C is designed to help OEM manufacturers audit the amount of regulated substances used in their products, and show that they responsibly dispose of, recycle or reuse electronics containing those substances.

Agile PG&C allows companies to cost-effectively comply with environmental regulations. Companies can use Agile PG&C to obtain compliance data for parts from their suppliers. This allows companies to

- Meet substance restrictions
- Satisfy reporting requirements for regulations
- Design recyclable products
- Minimize compliance costs
- Eliminate noncompliance on future products

Agile PG&C is a communication vehicle between the Compliance Manager and suppliers. The Compliance Manager ensures that a company's products adhere to government regulations and company policy. At the supplier, the Material Provider completes and signs off on material declarations, thereby disclosing which hazardous substances are contained within the components and subassemblies it provides.

For a more detailed overview of Agile PG&C features, refer to the *Product Governance & Compliance User Guide*.

Agile PG&C Interfaces and Classes

The following table lists Agile PG&C-related interfaces and classes:

Object	Interface	Constants Class
Declaration	`IDeclaration`	`DeclarationConstants`
Specification	`ISpecification`	`SpecficationConstants`
Substance	`ISubstance`	`SubstanceConstants`
Part Groups	`ICommodity`	`PartGroupConstants`

Items, Manufacturer Parts, and Part Groups are objects that are also related to Agile PG&C. They have Specifications, Compositions, and Substances tables that are populated with data when declarations are released. For Manufacturer Parts, you can edit the Compositions and Substances tables directly without submitting a declaration.

Note The terms “part group” and “commodity” are used interchangeably in this guide to refer to any `ICommodity`object. `ICommodity` represents the Part Group base class, which includes Commodity and Part Family subclasses.

Of course, other common Agile API interfaces, such as `ITable`, `IDataObject`, and `ICell`, are also used to work with Agile PG&C objects.

Agile PG&C Roles

Agile PLM provides two out-of-the-box roles designed for Agile PG&C users:

- Compliance Manager – This provides privileges needed to create and manage Agile PG&C objects, such as Declarations, Substances, and Specifications, and run Agile PG&C reports. Compliance Managers are responsible for routing material declarations to suppliers.
- (Restricted) Material Provider – This provides privileges needed to create and modify declarations, as well as read all other types of Agile PG&C objects. This role is typically assigned to supplier users, who have restricted access to the Agile PLM system. Material Providers are responsible for completing and signing off on material declarations.

To use Agile PG&C APIs mentioned in this chapter, make sure you log in as a user assigned either the Compliance Manager role, or the (Restricted) Material Provider role. For more information about Agile PLM roles, refer to the *Agile PLM Administrator Guide*.

Note The *Discover Change* privilege mask is not included in the *Compliance Manager* role. If you only have the Compliance Manager role, then you do not have sufficient privileges to use the API to set the calculated compliance of a part, in a Declaration, and pass the *Change Number* to the SDK call. To pass the Change Number in the SDK call, you must have the *Discover Change* privilege mask for that object in the Change Orders class. For more information, see Setting Values in the Calculated Compliance Field for Declaration Objects on page 312.

Creating Declarations, Specifications, and Substances

The following paragraphs provide definitions and procedures to define and mange these PG&C classes.

Creating Declarations

A Declaration object is the main record of Agile PG&C. It tracks the substances and materials that are used for items, manufacturer parts, and part groups. When you release a declaration, the information gathered from it is published to the product record, thereby updating the Composition data contained within the items, manufacturer parts, and part groups listed by the declaration.

There are seven declaration subclasses provided with Agile PLM:

- Homogeneous Material Declaration – A homogeneous material composition declaration that uses material-level specifications.
- IPC 1752-1 Declaration – A material composition declaration for electronic products that conforms to IPC standards and uses only one part-level specification.
- IPC 1752-2 Declaration – A homogeneous material composition declaration for electronic products that conforms to IPC standards and uses only one material-level specification.
- JGPSSI Declaration – A material composition declaration that follows the Japanese Green Procurement (JGP) standard and uses part-level specifications.
- Part Declaration – A material composition declaration that uses part-level or material-level specifications.
- Substance Declaration – A material composition declaration for each substance within part-level specifications.
- Supplier Declaration of Conformance – A questionnaire to assess supplier compliance with specifications from customers and government agencies. The survey addresses compliance at a general company level. Can be used for CSR type declarations.

The procedure for creating a declaration is the same for all declaration subclasses. You must specify the declaration subclass as well as values for the Cover Page.Name and Cover Page.Supplier attributes. Other declaration attributes are optional.

By default, the Cover Page.Name field uses an Autonumber format with the prefix “MD” (for “Material Declaration”). Although the Autonumber format isn’t required, it makes sense to use the same prefix for all declarations to make it easier to search for them.

Note The case required for the Cover Page.Name field depends on the selected character set for the field.

Supplier users with the (Restricted) Material Provider role can also create declarations. However, only the Cover Page.Name attribute is required to create the object. The Cover Page.Supplier attribute is filled in automatically with the user's supplier organization.

The following example shows how to create a JGPSSI declaration.

Example: Creating a JGPSSI Declaration

```
public void CreateJGPSSIDeclaration(String num, ISupplier supplier)
throws Exception {
   // Create a Map object to store parameters
   Map params = new HashMap();

   // Initialize the params object
   params.put(DeclarationConstants.ATT_COVER_PAGE_NAME, num);
   params.put(DeclarationConstants.ATT_COVER_PAGE_SUPPLIER, supplier);
   // Get the JGPSSI Declaration subclass
   IAgileClass declClass = m_session.getAdminInstance().getAgileClass(

DeclarationConstants.CLASS_JGPSSI_DECLARATION);
   // Create a new JGPSSI declaration
   IDeclaration object =
(IDeclaration)m_session.createObject(declClass, params);
}
```

Creating Specifications

Specifications are used to state the criteria that a product is expected to meet or exceed. They are generally used to limit the amount of restricted substances contained in a product. Specifications can be internal documents issued by a company or industry, or, more commonly, they are regulations issued by a governing body. Here are some examples of government regulations:

- Restrictions on the Use of Certain Hazardous Substances in Electrical and Electronic Equipment (RoHS) Directive, issued by the European Union
- Waste Electrical and Electronic Equipment (WEEE) Directive, issued by the European Union.
- Food Allergen Labeling and Consumer Protection Act (FALCPA), issued by the U.S.A. Food and Drug Administration (FDA)

A specification defines a list of substances, the parts-per-million (PPM) threshold for each substance, and whether a particular substance is restricted. Compliance Managers can use specifications to pre-populate material declarations with appropriate substances to ensure compliance.

The only required attribute you must specify when you create a specification is General Info.Name. The name must be unique. The name is case-insensitive, which means "ROHS" is treated the same as "Rohs".

The General Info.Validation Type attribute is important because it determines whether the specification

is Part Level (the default) or Homogeneous Material Level, which affects the types of declarations that can be used with the specification. Another optional attribute is General Info.Lifecycle Phase. When you create a specification, the default lifecycle phase is Active. To make the specification obsolete, change the value of its lifecycle phase attribute to Obsolete.

Example: Creating a specification

```
public void createSpecification(String name) throws Exception {
    ISpecification spec =
    (ISpecification)
    m_session.createObject(SpecificationConstants.CLASS_SPECIFICATION,
    name);


}
```

Creating Substances

There are four substance subclasses provided with Agile PLM:

- Subpart – a subunit of a component manufacturer part. The Composition table of a subpart can have other subparts, materials, substance groups, and substances.
- Material – a compound consisting of several substances. The Composition table of a material can have substance groups or substances.
- Substance Group – a group of substances. The Composition table of a substance group can have only substances.
- Substance – a single element, such as lead, chromium, or cadmium. Substances do not have a Composition table.

These substance subclasses comprise the hierarchy of objects that can appear on a Composition table, also known as the Bill of Substances.

Creating a Subpart

A subpart object is a subunit of a component that is tracked in Agile PLM. Subparts are parts without a part number that are used to create a bill of material of manufacturer parts or parts within a composition.

Example: Creating a subpart

```
public void createSubpart(String num) throws Exception {
    // Create a Map object to store parameters
    Map params = new HashMap();
    // Initialize the map object
    params.put(SubstanceConstants.ATT_GENERAL_INFO_NAME, num);
    // Get the Subpart subclass
  IAgileClass subsClass =
    m_session.getAdminInstance().getAgileClass(SubstanceConstants.CLASS_
    SUBPART);
    // Create a new Subpart
  ISubstance sub =
    (ISubstance)m_session.createObject(class, params);
}
```

Creating a Substance Group

A substance group object is a group of multiple substances tracked in Agile PLM that have a common base substance. Every substance within the group has a conversion factor used to convert the weight of the base substance of the group.

Example: Creating a substance group

```
public void createSubstanceGroup(String num, ISubstance sub) throws
Exception {
    // Create a Map object to store parameters
    Map params = new HashMap();
    // Initialize the map object
    params.put(SubstanceConstants.ATT_GENERAL_INFO_NAME, num);
    params.put(SubstanceConstants.ATT_GENERAL_INFO_BASE_SUBSTANCE, sub);
    // Get the Substance Group subclass
   IAgileClass subsClass =
    m_session.getAdminInstance().getAgileClass(SubstanceConstants.CLASS_
    SUBSTANCE_GROUP);
    // Create a new Substance Group
    ISubstance sub =
      (ISubstance)m_session.createObject(class, params);
}
```

Creating a Material

When you create a material object, the only attribute you need to specify is the General Info.Name attribute, which is equivalent to the substance number. After you create a material object, you can add substances to its Composition table.

Example: Creating a material object

```
public void createMaterial(String num, ISubstance[] substances) throws
Exception {
  // Create a Map object to store parameters
    Map params = new HashMap();
  // Initialize the params object
    params.put(SubstanceConstants.ATT_GENERAL_INFO_NAME, num);
  // Create a new material
 ISubstance material =
    (ISubstance)m_session.createObject(SubstanceConstants.CLASS_MATERIAL
    , params);
  // Get the Composition table
 ITable composition =
    material.getTable(SubstanceConstants.TABLE_COMPOSITION);
  // Add substances to the Composition table
 for (int i = 0; i < substances.length; ++i) {
    IRow row = composition.createRow(substances[i]);
  }
}
```

Creating a Substance

Like material objects, the only attribute you need to specify to create a substance is the General Info.Name attribute, which is equivalent to the substance number. You can also specify other optional attributes, such as General Info.CAS Number.

Example: Creating a substance

```
public void createSubstance(String num, String casNumber) throws
Exception {
    // Create a Map object to store parameters
    Map params = new HashMap();
    // Initialize the params object
    params.put(SubstanceConstants.ATT_GENERAL_INFO_NAME, num);
    params.put(SubstanceConstants.ATT_GENERAL_INFO_CAS_NUMBER,
    casNumber);
    // Get the Substance subclass
  IAgileClass subsClass =
    m_session.getAdminInstance().getAgileClass(SubstanceConstants.CLASS_
    SUBSTANCE);
    // Create a new substance
  ISubstance substance =
    (ISubstance)m_session.createObject(subsClass, params);
}
```

Adding Items, Manufacturer Parts, and Part Groups to Declarations

Each declaration has separate tables for items, manufacturer parts, and part groups. Each of these also has an associated composition table: Item Composition, Manufacturer Part Composition, and Part Group Composition.

When you add an item to the Items table of a declaration, the latest released revision of the item is used. If the item does not have a released revision, the Introductory revision is used.

The following example shows how to add items, manufacturer parts, and part groups to a declaration.

Example: Adding items, manufacturer parts, and part groups to a declaration

```
public void addDecObjects(IDeclaration dec) throws APIException {
  try {

    HashMap params = new HashMap();
      //Add an Item to the Items table
  ITable tblItems =
    dec.getTable(DeclarationConstants.TABLE_ITEMS);
    params.clear();
    params.put(DeclarationConstants.ATT_ITEMS_ITEM_NUMBER, "1000-02");
      IRow rowItems =
      tblItems.createRow(params);


      //Add a Manufacturer Part to the Manufacturer Parts table
```

```
ITable tblMfrParts =
    dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTS);
      params.clear();
    params.put(DeclarationConstants.ATT_MANUFACTURER_PARTS_MFR_PART_NUMB
    ER, "Widget103");
    params.put(DeclarationConstants.ATT_MANUFACTURER_PARTS_MFR_NAME,
    "ACME");
    IRow rowMfrParts = tblMfrParts.createRow(params);
      //Add a Commodity to the Part Groups table
 ITable tblPartGroups =
    dec.getTable(DeclarationConstants.TABLE_PARTGROUPS);
    params.clear();
    params.put(DeclarationConstants.ATT_PART_GROUPS_NAME, "RES");
      IRow rowPartGroups =
        tblPartGroups.createRow(params);
  } catch (APIException ex) {
      System.out.println(ex);
  }
}
```

Adding Substances to Declarations

You can add substances to the Item Composition, Manufacturer Part Composition, and Part Group Composition tables contained within a declaration. To publish substances into items, manufacturer parts, and part groups, you release the declaration. When the declaration is released, the substances get added automatically to the Substances tables of the corresponding items, manufacturer parts, and part groups.

The composition tables for a declaration are mapping tables; they map parts to their substances. If there are no substances for the parent object, the composition table has no rows.

To add a row to the composition tables of a declaration, use the `ITable.createRow()` method. Because the composition tables are mapping tables, you cannot pass an `ISubstance` object to create the row. Instead, specify a `Map` object containing attribute-value pairs.

Important	The Substances and Composition tables for items and part groups are read-only. They get populated with data only when declarations are released.

To add a substance to one of the Composition tables of a declaration:

1. Add an item, manufacturer part, or part group to the Items, Manufacturer Parts, or Part Groups tables of a declaration, respectively.
2. Add a substance row to the Composition table that references the parent row on the Items, Manufacturer Parts, or Part Groups table. Use the virtual attribute `DeclarationConstants.ATT_PARENT_ROW` to specify the parent row. When you add a substance, specify the substance name and substance type.

   Important — For the Agile SDK, Composition tables for declarations list all parent objects contained in the Items, Manufacturer Parts, and Part Groups tables. Agile Web Client represents Composition tables differently. It shows a separate Composition table for each parent object.

When you create a row in the Composition tables, you pass a `Map` object containing attribute-value pairs. The following table lists the attributes the `Map` object must contain:

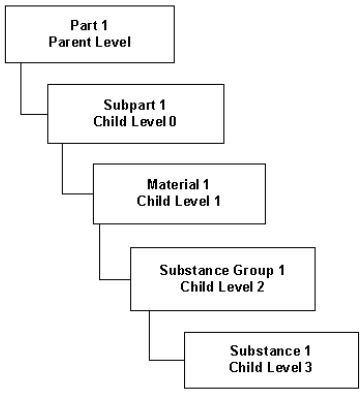
Composition Table	Required Attributes	DeclarationConstants
Item Composition	Item Row Substance Name	ATT_PARENT_ROW ATT_ITEM_COMPOSITION_SUBSTANCE_NAME
Manufacturer Part Composition	Manufacturer Part Row Substance Name	ATT_PARENT_ROW ATT_MANUFACTURER_PART_COMPOSITION_SUBSTANCE_NAME
Part Group Composition	Part Group Row Substance Name	ATT_PARENT_ROW ATT_PART_GROUP_COMPOSITION_SUBSTANCE_NAME

Structure of Bill of Substances

When you add substances to the Composition tables of a declaration, you can structure them in multiple levels. The number of levels you can use depends on the type of declaration.

- Homogeneous Materials Declaration – You can create a multilevel Bill of Substances with subparts, materials, substance groups, and substances. The composition must contain either a subpart or a material as a direct child. It can also include substances and substance groups, but they must be attached to a subpart or material.
- Substance Declaration/JGPSSI Declaration – Users can add substances or substance groups to the Composition tables.
- Part Declaration/Supplier Declaration of Conformance – These declarations do not have Composition tables.

The following figure shows the hierarchy for a Bill of Substances (Composition) with four child levels.



Rules for Adding Substances

Follow these rules when adding substances to a Composition table:

- Parent objects must be added before their children.
- Subparts can have the following children: other Subparts, Materials, Substance Groups, or Substances.
  - A Subpart cannot contain Subparts, Materials, Substance Groups, and Substances all at the same level.
  - A Subpart can contain other Subparts and Material at the same level.
  - A Subpart can contain Substance Groups and Substances at the same level.
- Material can have the following children: Substance Groups or Substances.
- Substance Groups can have the following children: Substances only.

Adding Subparts and Materials that Do Not Exist

When you add substances to a Composition table of a declaration, you can specify "dummy" subparts and materials that do not exist in the Agile PLM system. Such subparts and materials will be visible only within the Composition table. When you add "dummy" subparts and materials to the Composition table, you must specify the Substance Type attribute:

- `ATT_ITEM_COMPOSITION_SUBSTANCE_TYPE`
- `ATT_MANUFACTURER_PART_COMPOSITION_SUBSTANCE_TYPE`
- `ATT_PART_GROUP_COMPOSITION_SUBSTANCE_TYPE`

The following example shows how to add a dummy subpart or material to the Manufacturer Part Composition table. Because the `Substance Type` field is a list field, the value passed for it is an `IAgileList`.

Example: Adding a dummy subpart or material to the Manufacturer Part Composition table

```
public IRow addDummy(IDeclaration dec, IRow parentRow,
    String dummyName, IAgileList subtype)
    throws APIException {
   try {
    HashMap params = new HashMap();
   ITable tblMfrPartComp =
    dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTCOMPOSITION)
    ;
    params.put(DeclarationConstants.ATT_PARENT_ROW, parentRow);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, dummyName);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_TYPE, subtype);
    IRow dummyRow = tblMfrPartComp.createRow(params);
     return dummyRow;
   } catch (APIException ex) {
       System.out.println(ex);
   }
```

```
}
```

Adding Examples to Substances

The following examples show how to add:

- Substances to the Manufacturer Part Composition Table of a Homogeneous Material Declaration
- Substances to the Manufacturer Part Composition Table of a Substance Declaration

Adding Substances to Manufacturer Part Composition Table of Homogeneous Material Declarations

The following example shows how to add substances to a Manufacturer Part Composition table of a Homogeneous Material Declaration. The table has four levels: subparts, materials, substance groups, and substances. When you add a substance row to the table, we recommend that you pass a substance object (`ISubstance`) instead of a substance name (`String`) as the input parameter.

Example: Adding Homogeneous Material Level substances to a Manufacturer Part Composition table

```
public void addHomogeneousMaterialComp(IAgileSession m_session) throws
APIException {
  try {
   HashMap params = new HashMap();
    // Create a Declaration
 String num =
   "MDTEST001";
 ISupplier supplier =
   (ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE,
   "DISTRIBUTOR00007");
   params.put(DeclarationConstants.ATT_COVER_PAGE_NAME, num);
   params.put(DeclarationConstants.ATT_COVER_PAGE_SUPPLIER, supplier);
 IAgileClass declClass =
   m_session.getAdminInstance().getAgileClass(DeclarationConstants.CLAS
   S_HOMOGENEOUS_MATERIAL_DECLARATION);
 IDeclaration dec =
   (IDeclaration)m_session.createObject(declClass, params);
    // Add a Homogeneous Material Level spec to the Specifications
table
 ITable tblSpec =
   dec.getTable(DeclarationConstants.TABLE_SPECIFICATION);
  params.clear();
   ISpecification spec =
   (ISpecification)m_session.getObject(ISpecification.OBJECT_TYPE,
   "Lead Homogeneneous Material Level");
    IRow rowSpec = tblSpec.createRow(spec);
    // Add a Manufacturer Part to the Manufacturer Parts table
 ITable tblMfrParts =
   dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTS);
   params.clear();
   params.put(DeclarationConstants.ATT_MANUFACTURER_PARTS_MFR_PART_NUMB
   ER, "Widget103");
```

```
    params.put(DeclarationConstants.ATT_MANUFACTURER_PARTS_MFR_NAME,
    "ACME");
   IManufacturerPart mfrPart =
    (IManufacturerPart) m_session.
    getObject(IManufacturerPart.OBJECT_TYPE, params);
   IRow rowMfrParts =
    tblMfrParts.createRow(mfrPart);
     // Add a subpart to the Composition table
 ITable tblMfrPartComp =
    dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTCOMPOSITION)
    ;
 ISubstance subpart =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBPART,
    "Steel Casing");
    params.clear(); params.put(DeclarationConstants.ATT_PARENT_ROW,
    rowMfrParts);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, subpart);
   IRow rowSubpart =
     tblMfrPartComp.createRow(params);
     // Add a material
ISubstance material =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_MATERIAL,
    "Steel");
    params.clear();
    params.put(DeclarationConstants.ATT_PARENT_ROW, rowSubpart);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, material);
    IRow rowMaterial =
      tblMfrPartComp.createRow(params);
     // Add a substance group
ISubstance sg =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE_G
    ROUP,"Lead Componds");
    params.clear(); params.put(DeclarationConstants.ATT_PARENT_ROW,
    rowMaterial);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, sg);
   IRow rowSubGroup =
     tblMfrPartComp.createRow(params);
     // Add a substance
ISubstance sub =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE,"
    Lead");
    params.clear();
    params.put(DeclarationConstants.ATT_PARENT_ROW, rowSubGroup);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, sub);
    IRow rowSubs =
      tblMfrPartComp.createRow(params);
   } catch (APIException ex) {
       System.out.println(ex);
   }
 }
```

Adding Substances to Manufacturer Part Composition Table of Substance Declarations

The following example shows how to add substances to a Manufacturer Part Composition table of a Substance Declaration. The table has two levels: substance groups and substances.

Example: Adding Part Level substances to a Manufacturer Part Composition table

```
public void addSubstanceComp(IAgileSession m_session) throws
APIException {
 try {
   HashMap params = new HashMap();
    //Create a Declaration
 String num =
   "MDTEST001";
 ISupplier supplier =
   (ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE,
   "DISTRIBUTOR00007");
   params.put(DeclarationConstants.ATT_COVER_PAGE_NAME, num);
   params.put(DeclarationConstants.ATT_COVER_PAGE_SUPPLIER, supplier);
 IAgileClass declClass =
   m_session.getAdminInstance().getAgileClass(DeclarationConstants.CLAS
   S_SUBSTANCE_DECLARATION);
 IDeclaration dec =
   (IDeclaration)m_session.createObject(declClass, params);
    //Add a Specification to the Specifications table
 ITable tblSpec =
   dec.getTable(DeclarationConstants.TABLE_SPECIFICATION);
    params.clear();
    // Part Level
  ISpecification spec =
   (ISpecification)m_session.getObject(ISpecification.OBJECT_TYPE,
   "Lead Part Level");
 IRow rowSpec =
   tblSpec.createRow(spec);
    //Add a Manufacturer Part to the Manufacturer Parts table
 ITable tblMfrParts =
   dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTS);
   params.clear();
   params.put(DeclarationConstants.ATT_MANUFACTURER_PARTS_MFR_PART_NUMB
   ER, "Widget103");
   params.put(DeclarationConstants.ATT_MANUFACTURER_PARTS_MFR_NAME,
   "ACME");
 IManufacturerPart mfrPart =
   (IManufacturerPart)
   m_session.getObject(IManufacturerPart.OBJECT_TYPE, params);
 IRow rowMfrParts =
   tblMfrParts.createRow(mfrPart);
    //Add a substance group
 ITable tblMfrPartComp
   =
   dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTCOMPOSITION)
   ;
```

```
 ISubstance sg =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE_G
    ROUP,"Lead Componds");
   params.clear();
    params.put(DeclarationConstants.ATT_PARENT_ROW, rowMfrParts);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, sg);
 IRow rowSubGroup =
    tblMfrPartComp.createRow(params);
     //Add a substance
ISubstance sub =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE,"
    Lead");
   params.clear();
    params.put(DeclarationConstants.ATT_PARENT_ROW, rowSubGroup);
    params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SU
    BSTANCE_NAME, sub);
    IRow rowSubs =
      tblMfrPartComp.createRow(params);
  } catch (APIException ex) {
      System.out.println(ex);
  }
}
```

Adding Substances to a Specification

The Substances table of a specification is important to Agile PG&C because it identifies which substances are restricted and their threshold mass parts per million (PPM). Only substances and substance groups can be added to Substances table of Specification. To add a substance to the Substances table, use the `ITable.createRow()` method. You can pass an `ISubstance` or a `Map` object to create the new row.

Example: Adding a substance to a specification

```
public void addSubstanceToSpec(ISpecification spec, ISubstance
substance)
  throws Exception {
  IRow row = null;
  //Add a substance to the Substances table
  ITable tableSub =
spec.getTable(SpecificationConstants.TABLE_SUBSTANCES);
  row = tableSub.createRow(substance);
  if (row!=null){
    //Set value of Restricted
    ICell cell =
row.getCell(SpecificationConstants.ATT_SUBSTANCES_RESTRICTED);
    IAgileList list = (IAgileList)cell.getAvailableValues();
    list.setSelection(new Object[] {"Yes"});
    cell.setValue(list);

    //Set value of Threshold Mass PPM
```

```
row.setValue(SpecificationConstants.ATT_SUBSTANCES_THRESHOLD_MASS_PPM,
new Integer(10));
  }
}
```

Adding Specifications to a Declaration

The Specifications table of a declaration lists specifications related to the items, manufacturer parts, and part groups contained in the declaration. The purpose of a declaration is the ensure that suppliers comply with any restrictions stated in the specifications.

Rules for Adding Specifications

Specifications are optional for declarations. If you submit a declaration without a specification, it means you intend to collect raw data (mass or PPM) at the substance level. The supplier must provide information on all materials and substances.

If you add a specification to a declaration, note that declaration classes support different types of specifications. The following table lists the specification requirements for each type of declaration:

Declaration Type	Supported Specification Validation Types
Homogeneous Material Declaration	Homogeneous Material Level
IPC 1752-1 Declaration	Part Level
IPC 1752-2 Declaration	Homogeneous Material Level
JGPSSI Declaration	Part Level
Part Declaration	Part Level and Homogeneous Material Level
Substance Declaration	Part Level
Supplier Declaration of Conformance	Part Level and Homogeneous Material Level

Specifications may concern many substances, including those not used by the parts contained in the declaration. When the declaration is opened to the supplier, any relevant substances from the specifications are automatically added to the Item Composition, Manufacturer Part Composition, and Part Group Composition tables. This ensures that you are properly tracking any restricted substances contained in parts listed in the declaration.

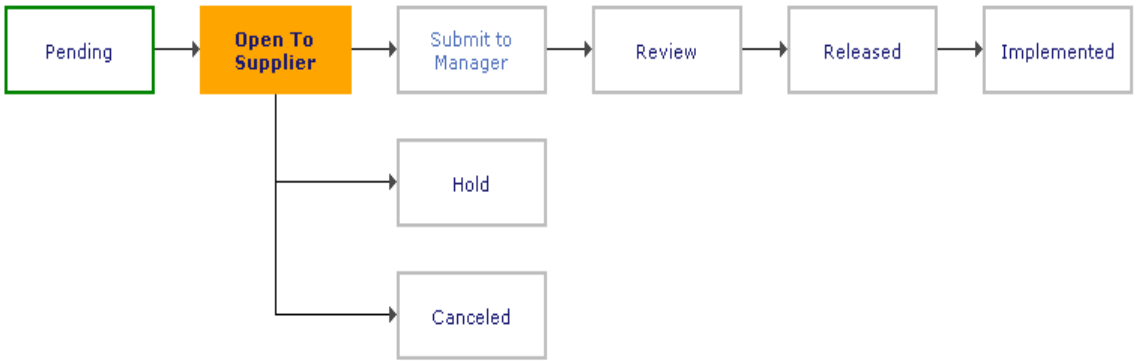
Example: Adding specifications to the Specification table

```
private void addSpecifications(IDeclaration dec, ISpecification[]
specs) throws Exception {
  ITable tableSpecs =
dec.getTable(DeclarationConstants.TABLE_SPECIFICATION);
  for (int i = 0; i < specs.length; ++i) {
    ISpecification spec = specs[i];
    IRow row = tableSpecs.createRow(spec);
  }
}
```

Routing Declarations

The Default Declarations Workflow follows a straightforward process flow, as shown in the following figure.

Figure 18: Default Declarations workflow



The following table describes each status in the Default Declarations workflow.

Status	Description
Pending	The Compliance Manager creates a new declaration, adding new items, manufacturer parts, or part groups. He also adds specifications to the declaration.
Open To Supplier	The Compliance Manager opens the declaration to the supplier, asking him to confirm whether parts comply with specifications. When the Workflow status of a declaration is changed from “Pending” to “Open To Supplier,” the Agile PLM server automatically populates the declaration’s Substances tables with any substances listed on its specifications.
Submit to Manager	The supplier electronically “signs” and submits the declaration back to the Compliance Manager.
Review	The Compliance Manager and other reviewers verify and approve the contents of the declaration.
Released	The Compliance Manager releases the declaration, thereby publishing the materials into the product record.
Implemented	Once the parts are manufactured and disseminated in the field, the Compliance Manager implements the declaration, thereby completing the workflow.

Before you can route a declaration, you should set values for the following three `Cover Page` fields:

- `Cover Page.Compliance Manager`
- `Cover Page.Workflow`
- `Cover Page.Due Date`

Technically, only the `Compliance Manager` and `Workflow` fields are required to route the declaration. The `Due Date` field is optional but should be specified for tracking purposes. The

following example shows how to set values for these three fields.

Example: Setting values for the Compliance Manager, Workflow, and Due Date fields

```
public void setFieldsNeededForRouting(IDeclaration dec) throws
Exception {
  //Set the Compliance Manager field
  IUser user = m_session.getCurrentUser();
  dec.setValue(DeclarationConstants.ATT_COVER_PAGE_COMPLIANCE_MANAGER,
user);
  //Set the Workflow field
  IWorkflow workflow = dec.getWorkflows()[0];
  dec.setWorkflow(workflow);

  //Set the Due Date field
  DateFormat df = new SimpleDateFormat("MM/dd/yy");
  dec.setValue(DeclarationConstants.ATT_COVER_PAGE_DUE_DATE,
df.parse("05/01/05"));
}
```

To change the status of a declaration, use the `IRoutable.changeStatus()` method. Once a declaration is opened to a supplier, only the supplier's contact users can edit it. For other users, including the Compliance Manager, the declaration becomes read-only. The following example shows how the Compliance Manager can change the status of a declaration to "Open To Supplier."

Example: Opening a declaration to a supplier

```
public void openToSupplier(IDeclaration dec) throws Exception {
  IStatus status = null;
  // Get the Open to Supplier status type
  IStatus[] stats = dec.getNextStatuses();
  for (int i = 0; i < stats.length; i++) {
    if (stats[i].toString().equals("Open To Supplier")) {
      status = stats[i];
      break;
    }
  }
  // Change to the Open to Supplier status
  dec.changeStatus(status, false, null, false, false, null, null, null,
false);
}
```

For more information about Agile APIs that support Workflow processes, see Managing Workflow on page 203.

Completing a Declaration

When a declaration is opened to a supplier, the supplier is responsible for completing the declaration and disclosing if any restricted substances are contained in the components and subassemblies it provides and whether those substances comply with specifications. To complete and sign off on declarations, one or more contact users for the supplier must be assigned the (Restricted) Material Provider role.

The Material Provider user should do the following to complete a declaration:

- Fill in the Mass, Declared PPM, and Declared Compliance fields for every substance listed on the Item Composition, Manufacturer Part Composition, and Part Group Composition tables, particularly for substances that are restricted by specifications.
- Complete other flex fields on the Composition tables as necessary.
- Add or remove substances from the declaration.

When the declaration is complete, the Material Provider user can sign off and submit the declaration to the Compliance Manager, described below.

Submitting Declarations to Compliance Managers

When the supplier changes the status of the declaration from “Open to Supplier” to “Submitted to Compliance Manager,” he must sign-off on the declaration. Therefore, he must use the `changeStatus()` method that has an additional `password` parameter:

```
changeStatus(IStatus newStatus, boolean auditRelease, String comment,
boolean notifyOriginator, boolean notifyCCB, Object[] notifyList,
Object[] approvers, Object[] observers, boolean urgent, String
password)
```

The following example shows how the supplier can sign off and submit the declaration to the Compliance Manager.

Example: Signing off and submitting a declaration to the Compliance Manager

```
public void submitToCM(IDeclaration dec) throws Exception {

  IStatus status = null;
  // Get the Submitted to Compliance Manager status type
  IStatus[] stats = dec.getNextStatuses();
  for (int i = 0; i < stats.length; i++) {
    if (stats[i].toString().equals("Submit To Manager")) {
      status = stats[i];
      break;
    }
  }
  // Change to the Submitted to Compliance Manager status (signoff
password is "agile")
  dec.changeStatus(status, false, null, false, false, null, null, null,
false, "agile");
}
```

Publishing a Declaration

The Agile API does not provide a method to publish a material declaration to the product record. Instead, a declaration is automatically published when it is released. Therefore, as far as the API is concerned, the substances table for an item, manufacturer part, or part group always reflects the last released declarations. However, Agile Web Client allows you to select an earlier declaration

and publish it, thereby updating the substances information in the product record.

Getting and Setting Weight Values

Unit of Measure fields have been implemented in Agile PLM to support mass (weight) values for Agile PG&C objects. The Unit of Measure datatype is a compound datatype that includes a numeric value and a unit, for example, grams or ounces.

You can configure and manage weight fields using the following interfaces:

- IMeasure
- IUnit
- IUnitOfMeasure
- IUnitOfMeasureManager

Although the Agile PLM administrator can define new measures from the UOM node in Agile Java Client, the Agile API supports only the Weight measure for Agile PG&C objects. You cannot use the Agile API to define new measures.

In Agile 9.2.1, the Title Block.Weight field for items was changed to Title Block.Mass. However, the Agile API constant for the field is still ItemConstants.TITLE_BLOCK_WEIGHT.

The following example shows how to get and set values for the Title Block.Mass field of an item.

Example: Getting and setting the mass (weight) value for an item

```
private IUnitOfMeasure getMassValue(IItem item) throws APIException {

  IUnitOfMeasure uom =
(IUnitOfMeasure)item.getValue(ItemConstants.ATT_TITLE_BLOCK_WEIGHT);
  System.out.println("Value: " + uom.getValue());
  System.out.println("Unit: " + uom.getUnit().toString());
  return uom;
}
private void setMassValue(IItem item, double value, String unit) throws
APIException {
  IUnitOfMeasure uom = null;
  IUnitOfMeasureManager uomm =
(IUnitOfMeasureManager)m_session.getManager(
    IUnitOfMeasureManager.class);
  uom = uomm.createUOM(value, unit);
  item.setValue(ItemConstants.ATT_TITLE_BLOCK_WEIGHT, uom);
  System.out.println("Value: " + uom.getValue());
  System.out.println("Unit: " + uom.getUnit().toString());

}
```

If you create a query to search for items by mass, only the numeric value is searched, not the unit. The server converts mass values to the standard unit before returning query results. For example, the following query returns all items whose mass value is between 1.0 and 2.0 grams (the default

standard unit). Items with a mass between 1000 and 2000 milligrams would also be included in the search results.

Example: Searching for items by mass

```
try {
  IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
    "select * from [Items] where [Title Block.Weight] between (1.0,
2.0)"
  );
  ITable results = query.execute();
} catch (APIException ex) {
  System.out.println(ex);
```

Adding Substance Compositions for Manufacturer Parts

With appropriate privileges, you can modify the Specifications, Compositions, and Substances tables of a manufacturer part directly without submitting a declaration. This feature is useful for manufacturing partners that want to specify composition information for their parts. To add a row to the Specifications, Compositions, and Substances tables, use the `ITable.createRow(Object)` method.

Note	Once a row has been added to the Compositions and Substances tables of a Manufacturer Part, you cannot update or remove it.

The procedure for adding rows to the Substances table of a Manufacturer Part is similar to the way you add rows to the composition tables for a declaration. Follow these steps to add substance compositions into a manufacturer part:

1. Optionally, add a specification to the Specifications table.
2. Add a row to the Compositions table. You must specify a value for the `ManufacturerPartConstants.ATT_COMPOSITIONS_COMPOSITION_TYPE` attribute.
3. Add one or more rows to the Substances table. Each row must reference the parent row from the Compositions table. Use the virtual attribute `ManufacturerPartConstants.ATT_PARENT_ROW` to specify the parent row. When you add a substance, specify the substance name and substance type.

For additional rules about adding substances to the Substances table, see Rules for Adding Substances on page 296.

The Composition Type attribute for the parent row determines the types of substances you can add to the Substances table. There are three possible Composition Type values:

- Homogeneous Material Composition – You can create a multilevel Bill of Substances with subparts, materials, substance groups, and substances. The composition must contain either a subpart or a material as a direct child. It can also include substances and substance groups, but they can only be attached to a subpart or material.
- Substance Composition – The Substances table can contain only substance groups and substances.
- Part Composition – You can't add rows to the Substances table.

Specifications that you reference in a row in the Compositions table must match the Composition Type attribute for that row. For example, if the Composition Type for the row is Homogeneous Material Composition, the validation type for a specification referenced in that row must be Homogeneous Material Level.

The following example shows how to define a Homogeneous Material composition for a manufacturer part. The Substances table has four levels: subparts, materials, substance groups, and substances.

Example: Adding specifications, compositions, and substances to a Manufacturer Part

```
public void addMfrPartSubs(IAgileSession m_session) throws APIException
{
  try {
    // Create a Manufacturer Part
   HashMap params = new HashMap();
   params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_P
   ART_NUMBER,"Widget");
   params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_N
   AME, "ACME");
 IManufacturerPart mfrPart =
   (IManufacturerPart)
   m_session.createObject(ManufacturerPartConstants.CLASS_MANUFACTURER_
   PART, params);
    // Add a Specification to the Specifications table
 ITable tblSpec =
   mfrPart.getTable(ManufacturerPartConstants.TABLE_SPECIFICATIONS);
ISpecification spec =
   (ISpecification)m_session.getObject(ISpecification.OBJECT_TYPE,"Lead
   Spec");
   // Homogeneous Material Level
  IRow rowSpec =
    tblSpec.createRow(spec);
    // Get the Compositions table
 ITable tblComp =
   mfrPart.getTable(ManufacturerPartConstants.TABLE_COMPOSITIONS);
       // Add a row to the Compositions table that references the
   specification params.clear();
   params.put(ManufacturerPartConstants.ATT_COMPOSITIONS_SPECIFICATION,
   spec.getName());
   params.put(ManufacturerPartConstants.ATT_COMPOSITIONS_COMPOSITION_TY
   PE,
     "Homogeneous Material Composition");
   IRow rowComp =
     tblComp.createRow(params);
    // Get the Substances table
ITable tblSubs =
   mfrPart.getTable(ManufacturerPartConstants.TABLE_SUBSTANCES);
    // Add a subpart
ISubstance subpart =
   (ISubstance)m_session.
                        getObject(SubstanceConstants.CLASS_SUBPART,
"Steel Casing");
   params.clear();
   params.put(ManufacturerPartConstants.ATT_PARENT_ROW, rowComp);
```

```
    params.put(ManufacturerPartConstants.ATT_SUBSTANCES_SUBSTANCE_NAME,
    subpart);
    IRow rowSubpart =
       tblSubs.createRow(params);
     // Add a material
 ISubstance material =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_MATERIAL,
    "Steel");
    params.clear();
    params.put(ManufacturerPartConstants.ATT_PARENT_ROW, rowSubpart);
    params.put(ManufacturerPartConstants.ATT_SUBSTANCES_SUBSTANCE_NAME,
    material);
    IRow rowMaterial =
      tblSubs.createRow(params);
     // Add a substance group
 ISubstance sg =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE_G
    ROUP, "Lead Componds");
    params.clear();
    params.put(ManufacturerPartConstants.ATT_PARENT_ROW, rowMaterial);
    params.put(ManufacturerPartConstants.ATT_SUBSTANCES_SUBSTANCE_NAME,
    sg);
    IRow rowSubGroup =
      tblSubs.createRow(params);
     // Add a substance
 ISubstance sub =
    (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE,"
    Lead");
    params.clear();
    params.put(ManufacturerPartConstants.ATT_PARENT_ROW, rowSubGroup);
    params.put(ManufacturerPartConstants.ATT_SUBSTANCES_SUBSTANCE_NAME,
    sub);
    IRow rowSubs =
      tblSubs.createRow(params);
  } catch (APIException ex) {
      System.out.println(ex);
  }
}
```

Rolling Up Compliance Data

After gathering compliance data for items, manufacturer parts, and part groups, compliance managers review the completed declarations to determine if the data is ready for publication into the product record. Once declarations are published with the data written through to parts and part groups on BOMs, compliance managers must examine and test BOMs to ensure the assemblies and products are compliant. This process is called compliance validation and is fulfilled through compliance rollups. Rollups are built into the system. They are easy to use and rollup results are available on the UI. For more information on rolling up compliance data and the business logic behind this process, refer to the *Agile Product Governance & Compliance User Guide.*

The SDK supports calling the PG&C Rollup function on the server side. This is the same rollup function that is called by the UI. The `IPGCRollup` interface in `IItem` supports this feature.

Understanding the IPGCRollup Interface

The `IPGCRollup` interface provides the following methods to support rolling up compliance data:

- `rollup()`
- `rollup(Date)`

One of these methods has no parameters and the other has *Date* as a parameter. The Date parameter in the rollup API is used by the system to set the timestamp for the rollup, when it is done.

Example: IPGCRollup methods

```
public interface IPGCRollup {
    public void rollup()
      throws APIException;
    public void rollup(Date rollupDate)
      throws APIException;
}
```

Note After invoking `rollup(Date)`, it is necessary to call `IDataObject.refresh()` to make sure the rollup function is taken effect. Otherwise, the system will display the results obtained in the previous rollup if the timestamp of the recent rollup is the same as the Date parameter.

Passing the Date Parameter

If you do not pass the date, the system will use the current time provided by the system. When a rollup is performed on a set of items, if the timestamp of the recent rollup on an item is the same as the passed Date parameter, the system will not repeat the rollup process on that item. Instead, it will display the results obtained in the previous rollup. You may want to use this date feature if there is a large number of items to rollup and you want to use the SDK to call all of them. In this case, you will get the current date first, and then the pass that date for the subsequent SDK `Rollup(Date)` call. For example, you want to use the SDK to roll up data for Assembly 1 and Assembly 2. In this case, the SDK is called twice. The first instance, to roll up data for Assembly 1, and the second instance, to rollup data for Assembly 2. With the date parameter already inside the rollup when performing the rollup on Assembly 2, the system will reuse the previous rollup data obtained for Item1.

```
Assembly 1
   Item1
   Iitem2
Assembly 2
   Item1
   Item3
```

Using the IPGCRollup Interface

The following examples roll up the assembled data on Item and Manufacturer Part

- Item (latest released ECO or MCO)
- MPN (latest released ECO or MCO)

Rolling Up Assembled Data on Items

This example calls an existing API using the SDK to identify the top level parent of a given *Item* (its latest released ECO or MCO). Next, it will call the rollup API on the top level parent returned by the previous API to ensure the assemblies and products are compliant.

Example: Identifying the top level parent for and Item

```
public void itemRollup(String itemStr) throws Exception{
  try {
  IItem item =
    (IItem)m_session.getObject(IItem.OBJECT_TYPE, itemStr);
 IQuery query =
    (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,
    ItemConstants.CLASS_ITEM_BASE_CLASS);
//      IQuery query = (IQuery)
m_session.createObject(IQuery.OBJECT_TYPE, ItemConstants.CLASS_PART);
    query.setSearchType(QueryConstants.WHERE_USED_TOP_LEVEL);
    query.setCriteria("[1001] Equal To '"+item.getName()+"'");
//

query.setCriteria("["+SDKWrapper.getString("TITLE_BLOCK")+"."+SDKWrappe
r.getString("IQuery_Number")+"] Equal To '"+item.getName()+"'");
    ITable results=query.execute();
    if (results.size() > 0) {
      Iterator it =
         results.getReferentIterator();
    if (it.hasNext()) {
      IItem obj =
      (IItem)it.next();

    IItem tlaItem =
      IItem)m_session.getObject(IItem.OBJECT_TYPE, obj.getName());
       tlaItem.rollup();
           }
       }
       else {
       item.rollup();
       }

       } catch (APIException e) {
       throw e;
       }
  return;
  }
```

Rolling Up Assembled Data on MPNs

This example calls an existing API using the SDK to identify the top level parent of a given *MPN* (its latest released ECO or MCO). Next, it will call the rollup API on the top level parent returned by the previous API to ensure the assemblies and products are compliant.

Example: Identifying the top level parent for an MPN

```
public void testMfrPartRollup() throws Exception{
IManufacturerPartmfrp =
    (IManufacturerPart)
    m_session.getObject(IManufacturerPart.OBJECT_TYPE,
    "HARRIS::IS82C55A96");//
ITable whereused =
    mfrp.getTable(ManufacturerPartConstants.TABLE_WHEREUSED);
Iterator it =
    whereused.iterator();
while(it.hasNext())
  {
    IRow r = (IRow)it.next();
     // read item number
String itemStr =
    r.getValue(ManufacturerPartConstants.ATT_WHERE_USED_ITEM_NUMBER).toS
    tring();
       try {
       itemRollup(itemStr);
           } catch (APIException e) {
            int error =
               ((Integer)e.getErrorCode()).intValue();
           }
}
  return;
}
```

Setting Values in the Calculated Compliance Field for Item Objects

Use the following API to set the value of the *Calculated Compliance* field on Specifications table, for `Item` and `ManufacturerPart` objects:

```
Public void setCalculatedComplianceForPartSpec(Object specName, Object
complianceEntryValue) throws APIException
```

In this API, the `specName` parameter is the name of the `Specification` object, and the `complianceEntryValue` parameter is the actual value of the Calculated Compliance field, which can be any entry in the Calculated Compliance list. Both parameters are of type String.

When this value is set by the SDK Client, it is never overwritten during the Rollup. This API allows users to set the calculated compliance value based on their own defined logic, instead of using the system's default logic.

Example: Setting the value of the Calculated Compliance field for Item objects

```
// COMPLIANT, the actual value of the Calculated Compliance field shows
the Specification
```

```
            is compliant or not based on customized calculated
compliance result
String COMPLIANT = "Compliant";
// spec_num is the Specification Name in Item object's Specification
Table
String spec_num =
    row.getValue(ItemConstants.ATT_SPECIFICATIONS_SPECIFICATION).toStrin
    g();
item.setCalculatedComplianceForPartSpec(spec_num, COMPLIANT);
```

Setting Values in the Calculated Compliance Field for Declaration Objects

This is similar to the previous API that enabled setting the *Calculated Compliance* field for Item objects. You can use this API to set the value of the Calculated Compliance field in Item table and Manufacturer Part table for Declaration objects.

```
Public void setCalculatedComplianceForMDOPartSpec (Object partName, Object
partClassName, Object changeNumber, Object specName, Object
complianceEntryValue)) throws APIException
```

The system recognizes the SDK Client has set this value and will use the new setting in the subsequent response during Rollup. In this API, the parameter `changeNumber` is optional. When the Declaration object has only one revision of an item, you can set the value of `changeNumber` to null. If the Declaration object has more than one revisions of an item, you must set the value of `changeNumber` for the proper execution of the API.

Similar to the previous API, when this value is set by the SDK Client, it is never overwritten during the Rollup within the declaration. This API allows users to set the calculated compliance value based on their own defined logic, instead of using the system's default logic.

Note If the SDK developer intends to pass the `changeNumber` field to `setCalculatedComplianceForDeclarationPartSpec()`, the developer must have the Discover Change privilege mask for that change.

Example: Setting the value of the Calculated Compliance field for Declaration objects

```
// complianceValue -- This is the customized calculated compliance
   value and shows if the part is compliant to a Spec
String ComplianceValue = "Compliant";
// partName is the Item/Mfr Part name in Declaration's Item/MfrPart
table.  If it is a mfr part, it should be like "MfrName::MfrPartName"
String partName ="P00001"; String partClassName = "Parts";
// If the added part in Declaration is an Item, the changeName should
be the
   Change number corresponding to the Item's revision.
// If the added part in Declaration is a Mfr Part, the
   changeName should be "null"
String changeName = "C00001";
// spec_num is the Specification Name in Declaration object's
   Specification Table
String specName = "Rohs";
Declaration.setCalculatedComplianceForDeclarationPartSpec
    (partName, partClassName, changeName, specName, complianceValue);
```

Chapter 19

Handling Exceptions

This chapter includes the following:

- About Exceptions 313
- Exception Constants 314
- Getting Error Codes 314
- Disabling and Enabling Error Codes with Bulk APIs 314
- Getting Error Messages 315
- Disabling and Enabling Warning Messages 316
- Checking if APIException is Warning and not Error 317
- Saving and Restoring State Enabled and Disabled Warnings 317
- Deleting Warnings Automatically Disabled by Agile API 318

About Exceptions

Errors that cause a problem in a Java program are called exceptions. When Java throws an exception that is not caught, your program may quit, or errors may appear onscreen. To handle an exception gracefully, your program must:

- Protect code that contains a method that might throw an exception by putting it in a `try` block.
- Test for and handle any exceptions that are thrown inside a `catch` block.

The Agile API provides a subclass of `Exception` called `APIException`. This is a general-purpose exception class that is used throughout the Agile API to handle Agile PLM runtime errors. In the Agile API HTML reference, each method indicates the types of exceptions it throws. Generally, any Agile API method that requires interaction with the Agile Application Server throws `APIException`. The table below lists the `APIException` class methods for handling exceptions:

Method	Description
`getErrorCode()`	Returns the number of the error code associated with the `APIException`.
`getMessage()`	Returns the error message associated with the `APIException`.
`getRootCause()`	Returns the root cause of the `APIException`, if any.
`getType()`	Returns the type of exception.

Exception Constants

The `ExceptionConstants` class contains String constants for all Agile Application Server and Agile API runtime error and warning codes. For a description of each of these constants, refer to the API Reference files at http://edelivery.oracle.com/.

Several of `ExceptionConstants` are for exceptions that are used to display an Agile PLM warning message before completing an action. All constants for warning messages end with the suffix `WARNING`. If you don't want to use Agile PLM warning messages in your code, you can disable them. For more information, see Disabling and Enabling Warning Messages on page 316.

Getting Error Codes

To properly trap warning errors, you may need to retrieve the error code of the exception and then handle it appropriately. Generally, this involves displaying a confirmation dialog box to let the user choose whether to complete the action. The following example shows how to check for the error code of an exception in the `catch` block.

Example: Getting Agile PLM error codes

```
private void removeApprover(IChange change, IUser[] approvers, IUser[]
observers, String comment) {
  try {
    // Remove the selected approver
   change.removeApprovers(change.getStatus(), approvers, observers,
   comment);
  } catch (APIException ex) {
   if
   (ex.getErrorCode().equals(ExceptionConstants.APDM_RESPONDEDUSERS_WAR
   NING))
   JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning",
   JOptionPane.YES_NO_OPTION);
  }
}
```

Disabling and Enabling Error Codes with Bulk APIs

The SDK supports the following bulk operations in `IAgileSession` to disable/enable all error codes or for a given set of error codes:

- `IAgileSession.enableWarnings(Integer[])`
- `IAgileSession.disableWarnings(Integer[])`
- `IAgileSession.enableAllWarnings()`
- `IAgileSession.disableAllWarnings()`

The process is similar to the previous example. The following example shows how to use these bulk APIs to suppress warnings while releasing a Change.

Example: Disabling and enabling error codes in the bulk mode

```
public static void releseECO(IAgileSession session, IChange change)
     throws Exception {
    // Set workflow
    IWorkflow workflow = change.getWorkflows()[0];
    change.setWorkflow(workflow);

    IStatus submit = getStatus(workflow, StatusConstants.TYPE_SUBMIT);
    IStatus ccb = getStatus(workflow, StatusConstants.TYPE_REVIEW);
    IStatus released = getStatus(workflow,
    StatusConstants.TYPE RELEASED);
    session.disableWarnings(new Integer[] {
          ExceptionConstants.APDM_WFL_ATLEASTONECHGANALYST_WARNING,
          ExceptionConstants.APDM_MISSINGFIELDS_WARNING });
    // instead you can use session.disableAllWarnings()
    // route to SUBMIT
    change.changeStatus(submit, false, null, false, false, new
    Object[]{}, new Object[]{}, new Object[] {}, false);
    // Change status to CCB
    change.changeStatus(ccb, false, null, false, false, new Object[]{},
    new Object[]{}, new Object[]{}, false);
    // route from CCB to release
    change.changeStatus(released, false, "release", false, false, new
    Object[]{}, new Object[]{}, new Object[]{}, false);
    session.enableWarnings(new Integer[] {
          ExceptionConstants.APDM_WFL_ATLEASTONECHGANALYST_WARNING,
          ExceptionConstants.APDM_MISSINGFIELDS_WARNING });
    // instead you can use session.enableAllWarnings()
}

public static IStatus getStatus(IWorkflow workflow, StatusConstants
status)
        throws Exception {
          IStatus[] states = workflow.getStates(status);
          IStatus state = states[0];
          return state;
}
```

Getting Error Messages

If your program throws an `APIException`, which indicates an Agile PLM runtime error, you may want to display an error message. You can use the `getMessage()` method to return the error message string and then display it in a message dialog box, as shown in the following example.

Example: Getting an error message

```
// Display an error message dialog
void errorMessage(APIException ex) {
  try {
    JOptionPane.showMessageDialog(null, ex.getMessage(), "Error",
    JOptionPane.ERROR_MESSAGE);
  } catch (Exception e) {}
}
```

For a list of Agile PLM error messages, refer to the API Reference files (at http://edelivery.oracle.com/) under `ExceptionConstants`.

Disabling and Enabling Warning Messages

Several Agile PLM error messages are warnings that give you the option to stop or continue with an operation. By default, most error messages, including warning messages, are enabled. If you try to perform an action that triggers a warning, an exception will be thrown. To avoid the exception, you can disable the warning message before performing the action.

The following example shows how to check whether attempting to release a change causes an exception to be thrown. If the error code for the exception is `ExceptionConstants.APDM_UNRESPONDEDCHANGE_WARNING`, the program displays a warning. The user can click Yes in the warning dialog box to release the change.

Example: Disabling and enabling error codes

```
private void releaseChange(IAgileSession m_session, IChange chgObj) {
    IStatus nextStatus = null;
    try {
     // Get the default next status
    nextStatus = chgObj.getDefaultNextStatus();
     // Release the Change
    chgObj.changeStatus(nextStatus, false, "", false, false, null, null,
    null, false);
   } catch (APIException ex) {
     // If the exception is error code
     // ExceptionConstants.APDM_UNRESPONDEDCHANGE_WARNING,
     // display a warning message
    if (ex.getErrorCode() ==
    ExceptionConstants.APDM_UNRESPONDEDCHANGE_WARNING) {
      int i =
       JOptionPane.showConfirmDialog(null, ex.getMessage(),
           "Warning", JOptionPane.YES_NO_OPTION);
       if (i == 0) {
    // If the user clicks Yes on the warning, disable the error code and
    release the change
         try {
     // Disable the warning
    m_session.disableWarning(ExceptionConstants.APDM_UNRESPONDEDCHANGE_W
    ARNING);
           // Release the Change
    chgObj.changeStatus(nextStatus, false, "", true, true, null, null,
    null, false);
       // Enable all warnings
    m_session.enableWarning(ExceptionConstants.APDM_UNRESPONDEDCHANGE_WA
    RNING);
        } catch (APIException exc) {}
       }
     }
   }
}
```

Checking if APIException is Warning and not Error

As noted above, if you try to perform an operation that triggers a warning, an exception will be thrown. Warning messages are helpful for interactive GUI clients, like Agile Web Client, but you may not want to use them in your Agile API program, particularly if it performs batch processes.

You can use `APIException.isWarning()` to check whether an Agile PLM exception is a warning. If so, you can disable the warning to continue the operation.

Example: Checking if an APIException is a warning

```
private void checkIfWarning(IAgileSession m_session) {
    boolean gotWarning = true;
    while (gotWarning) {
     try {
     // Add some API code here that throws an exception
      m_session.doNothing();
     gotWarning = false;
   } catch (APIException e) {
      try {
       if (e.isWarning())
      m_session.disableWarning((Integer)e.getErrorCode());
       } catch (Exception ex) {}
       continue;
     }
     break;
   }
}
```

Saving and Restoring State Enabled and Disabled Warnings

Rather than keep track of which warning messages are disabled or enabled before beginning a particular operation, you can use `IAgileSession.pushWarningState()` to save the current state of enabled and disabled warnings. After completing the operation, you can restore the previous state of enabled and disabled warnings using `IAgileSession.popWarningState()`.

Example: Using pushWarningState() and popWarningState()

```
private void pushPopWarningState(IAgileSession m_session, IItem item)
throws APIException {
  // Save the current state of enabled/disabled warnings
m_session.pushWarningState();
  // Disable two AML warnings
   m_session.disableWarning(ExceptionConstants.APDM_WARN_MFRNAMECHANGE_
   WARNING);
   m_session.disableWarning(ExceptionConstants.APDM_ONEPARTONEMFRPART_W
   ARNING);
  // Get the Manufacturers table
   ITable aml = item.getTable(ItemConstants.TABLE_MANUFACTURERS);
  // Create a new row and set a value for the row
   HashMap amlEntry = new HashMap();
   amlEntry.put(ItemConstants.ATT_MANUFACTURERS_MFR_NAME, "MFR_TEST3");
```

```
    amlEntry.put(ItemConstants.ATT_MANUFACTURERS_MFR_PART_NUMBER,
    "MFR_PART3");
    IRow rowAML1 = aml.createRow(amlEntry);
    rowAML1.setValue(ItemConstants.ATT_MANUFACTURERS_REFERENCE_NOTES,
    "new note");
  // Restore the previous state of enabled/disabled warnings
m_session.popWarningState();
}
```

Deleting Warnings Automatically Disabled by Agile API

In the Agile Web Client, when you try to delete an object a warning message appears. These warning messages are not appropriate for batch processes in an Agile API program. Therefore, the Agile API implicitly disables the following warnings, which saves you the trouble of disabling them in your code.

- `ExceptionConstants.APDM_HARDDELETE_WARNING`
- `ExceptionConstants.APDM_SOFTDELETE_WARNING`

For more information about deleting objects, Deleting and Undeleting Objects.

Chapter 20

Performing Administrative Tasks

This chapter includes the following:

- About Agile PLM Administration 319
- Privileges Required to Administer Agile PLM 320
- Administrative Interfaces 320
- Getting an IAdmin Instance 321
- Working with Nodes 321
- Managing Agile PLM Classes 326
- Working with Attributes 330
- Working with Properties of Administrative Nodes 334
- Managing Users 335
- Managing User Groups 340

About Agile PLM Administration

Agile Java Client provides administrative functionality that lets you manage the Agile Application Server. It lets you quickly and easily adapt your Agile PLM system to fit the way you do business. You can customize the Agile PLM system in several ways:

- Modify Agile PLM database properties
- Define object classes and subclasses
- Set preferences
- Create and configure user accounts
- Define user groups
- Define roles and privileges
- Define *SmartRules*, which set how you manage your change control process

The Agile API provides read/write access to all nodes of Agile PLM's administrative functionality. This means you can create Agile API programs that let users read and modify Agile PLM subclasses, and add, modify, or delete Agile PLM users. The Agile API does not allow you to create new nodes in the administrative tree structure. Therefore, you can't create workflows, criteria, and roles. However, you can create users and user groups because those objects have been implemented as data objects; `IUser` and `IUserGroup` both extend `IDataObject`.

Privileges Required to Administer Agile PLM

Before you can administer the Agile Application Server, you must have proper privileges. For access to administrative functionality, you should have the Administrator privilege. The Administrator role grants the Administrator privilege to all administrative functionality available on the server. The User Administrator role grants the Administrator privilege for functionality related to users and user groups.

Without the Administrator privilege, you cannot modify administrative nodes, users, and user groups. If you have not yet been granted Administrator rights to the Agile PLM system, contact the Agile PLM administrator.

To create users and user groups, you need the Create privilege for those objects. Several roles supplied with the Agile PLM system, such as the Administrator, User Administrator, and Change Analyst roles, include the Create privilege for users and user groups.

Administrative Interfaces

The following table lists interfaces related to Agile PLM's administrative functionality.

Interface	Description
`IAdmin`	Interface that lets you get Agile PLM classes, nodes, users, or user groups
`IAgileClass`	Class definition used to identify the category to which an object belongs
`IAgileList`	A general-purpose list interface for all SingleList or MultiList attributes and properties
`IAttribute`	Provides detailed information about a particular data member in an object
`IAutoNumber`	An AutoNumber source, which is a predefined, consecutive number series used to automatically number Agile PLM objects
`ICriteria`	A reusable set of search criteria used primarily for queries and Workflows
`INode`	A node in the administrative hierarchy. Each node is equivalent to an Admin node in the Agile Java Client
`IProperty`	A property of an Agile PLM administrative node
`IRoutableDesc`	Metadata that describes any object that implements the `IRoutable` interface, you can use `IRoutableDesc` to get the workflows for a class without instantiating an object of that class
`ITableDesc`	Metadata that describes an Agile PLM table, you can use `ITableDesc` to get table attributes without loading a table
`ITreeNode`	A generic node in a hierarchical tree structure. Several administrative interfaces, such as `INode` and `IFolder`, are subinterfaces of `ITreeNode` and therefore inherit its functionality Note: There is also a deprecated `ITree` interface which provides similar functionality to `ITreeNode`. Be sure to use `ITreeNode` instead.
`IUser`	An Agile PLM user
`IUserGroup`	A user group. Use user groups to define project teams, site-related groups, departments, or global groups

Interface	Description
IWorkflow	A Workflow node

Getting an IAdmin Instance

The IAdmin interface provides access to most administrative functionality for the Agile Application Server. To use the IAdmin interface, you first get an instance of IAdmin from the current session. The following example shows how to log in to the Agile Application Server and get an IAdmin instance.

Example: Getting an IAdmin instance

```
public IAgileSession m_session;
public IAdmin m_admin;
public AgileSessionFactory m_factory;

try {
 HashMap params =
   new HashMap();
 params.put(AgileSessionFactory.USERNAME, "jdassin");
 params.put(AgileSessionFactory.PASSWORD, "agile");
 m_factory =
   AgileSessionFactory.getInstance("http://agileserver/virtualPath");
 m_session =
   m_factory.createSession(params);
 m_admin =
   m_session.getAdminInstance();
} catch (APIException ex) {
  System.out.println(ex);
}
```

Once you have an IAdmin instance, you can:

- Traverse the server nodes
- Traverse the folder hierarchy.
- Get Agile PLM classes and subclasses.
- Get users.
- Get user groups.

Working with Nodes

The `INode` object represents a single node or object within Agile PLM's administrative tree. Similar to the Windows Explorer interface, each `INode` can be expanded to show child nodes. This simple hierarchy lets you navigate the administrative tree structure on the Agile Application Server. Examples of nodes are the root node (also called the Database node), Classes, Preferences, Roles, Privileges, and SmartRules.

This table shows the mapping of Agile Java Client nodes to Agile API administrative functionality.

Agile Java Client node	Agile API equivalent
Data Settings	
Classes	`NodeConstants.NODE_AGILE_CLASSES`
Character Sets	`NodeConstants.NODE_CHARACTER_SETS`
Lists	Not supported
Process Extensions	Not supported
AutoNumbers	`NodeConstants.NODE_AUTONUMBERS`
Criteria	`NodeConstants.NODE_CRITERIA_LIBRARY`
Workflow Settings	
Workflows	`NodeConstants.NODE_AGILE_WORKFLOWS`
User Settings	
Account Policy	Not supported
Users	Create a query of users
User Groups	Create a query of user groups
Supplier Groups	Not supported
Roles	`NodeConstants.NODE_ROLES`
Privileges	`NodeConstants.NODE_PRIVILEGES`
User Monitor	Not supported
Deleted Users	Not supported
Deleted User Groups	Not supported
System Settings	
SmartRules	`NodeConstants.NODE_SMARTRULES`
Viewer & Files	`NodeConstants.NODE_VIEWER_AND_FILES`
Notifications	`NodeConstants.NODE_NOTIFICATION_TEMPLATES`
Full Text Search	Not supported

Agile Java Client node	Agile API equivalent
UOM	Not supported
Company Profile	Not supported
Currency Exchange Rates	`IAdmin.getConversionRates()`
Commodities	Not supported
Product Cost Management	
Ship To Locations	Not supported
Projects Execution	
Projects Health	Not supported
Cost Status	Not supported
Quality Status	Not supported
Resource Status	Not supported
Dashboard Management	Not supported
Default Role	Not supported
Agile Content Service	
Subscribers	`NodeConstants.NODE_SUBSCRIBERS`
Destinations	`NodeConstants.NODE_DESTINATIONS`
Events	`NodeConstants.NODE_EVENTS`
Filters	`NodeConstants.NODE_FILTERS`
Package Services	Not supported
Response Services	Not supported
Product Governance & Compliance	
Sign Off Message	Not supported
Server Settings	
Locations	`NodeConstants.NODE_SERVER_LOCATION`
Database	`NodeConstants.ROOT`
Preferences	`NodeConstants.NODE_PREFERENCES`
Licenses	`NodeConstants.NODE_SERVER_LICENSES` `NodeConstants.NODE_USER_LICENSES`
Task Monitor	Not supported
Task Configuration	Not supported
Example	

Agile Java Client node	Agile API equivalent
Example Roles	Not supported
Example Privileges	Not supported
Example Criteria	Not supported

Agile Web Client allows you to view and edit system and user settings by choosing `Admin` and `Settings` from the menu, respectively. The following table identifies how Agile Web Client administrative functionality maps to the Agile API.

Agile Web Client Node	Agile API equivalent
Tools > My Settings	
User Profile	User.General Info page
Change Password	`IUser.changeLoginPassword()` and `IUser.changeApprovalPassword()`
Transfer Authority	Not supported
Organize Bookmarks	My-Inbox folder
Organize Searches	Searches folder
Organize Reports	Not supported
Personal Groups	My-Inbox folder
Deleted Personal Groups	Not supported
Personal Criteria	Not supported
Personal Supplier Groups	Not supported
Tools > Administration > Web Client Settings	
Themes	Not supported
Tools > Administrator > User Settings	
Users	Create a query of users
User Groups	Create a query of user groups
Supplier Groups	Not supported
Deleted Users	Not supported
Deleted User Groups	Not supported
Dashboard Configuration	Not supported

Admin nodes in Agile PLM Clients do not have names that match up identically to their respective `NodeConstants`. For example, the Notifications node in Agile Java Client is equivalent to `NodeConstants.NODE_NOTIFICATION_TEMPLATES`. Similarly, the hierarchy of nodes that are represented in the Agile PLM database does not exactly match Agile Java Client node hierarchy.

If your Agile API program provides a tree view of the Agile PLM administrative nodes, you can use the view to interactively retrieve `INode` objects. From each `INode` object you can get the child nodes. If you continue to traverse the administrative node hierarchy, you can reach all node levels.

The following example shows how to retrieve the root node and its children, thus displaying the top-level nodes on the Agile Application Server.

Example: Retrieving top-level nodes

```
private void getTopLevelNodes() throws APIException {
 INode root =
   m_admin.getNode(NodeConstants.ROOT);
  if (null != root) {
   System.out.println(root.getName() + ", " + root.getId());
 Collection childNodes =
   root.getChildNodes();
  for (Iterator it =
   childNodes.iterator();it.hasNext();) {
    INode node =
      (INode)it.next();
   System.out.println(node.getName() + ", " + node.getId());
    }
  }
}
```

Note When you call `getChildNodes()` on the root node, the results include several undocumented Agile PLM nodes. Any undocumented nodes are not supported by the Agile API.

For faster access, you can also retrieve a node by specifying its node ID constant. The `NodeConstants` class lists all administrative nodes that are directly accessible. The following example shows how to retrieve the `SmartRules` node and its properties.

Example: Retrieving SmartRules values

```
private void getSmartRules() throws APIException {
  //Get the SmartRules node in Agile Administrator
  INode node = m_admin.getNode(NodeConstants.NODE_SMARTRULES);
  System.out.println("SmartRules Properties");
  //Get SmartRules properties
  IProperty[] props = (IProperty[])node.getProperties();
  for (int i = 0; i < props.length; i++) {
    System.out.println("Name : " + props[i].getName());
    Object value = props[i].getValue();
    System.out.println("Value : " + value);
  }
}
```

Another way to get a node is to locate a parent node and then get one of its children using the `ITreeNode.getChildNode()` method. The `getChildNode()` method lets you specify a node by name or ID. You can also specify the path to a subnode, separating each node level with a slash character (/). The following example shows how to use the `getChildNode()` method to retrieve a node.

Example: Retrieving nodes using ITreeNode.getChildNode()

```
private INode getChildNode(INode node, String childName) throws
APIException {
  Node child = (INode)(node.getChildNode(childName));
  return child;

}
```

Working with the Classes Node

The Classes node and its subnodes are similar to the `IAgileClass` objects that are returned by the `IAdmin.getAgileClasses()` method. The difference is that `getAgileClasses()` returns several virtual classes, such as Item and Change, that are not represented as nodes. To modify the properties of the attribute of a particular node, Agile recommends using the `IAdmin.getAgileClasses()` or `IAdmin.getAgileClass()` methods. Although it's possible to modify a subclass by traversing the Classes node and its subnodes, it is much easier to work with `IAgileClass` objects. For more information, see Managing Agile PLM Classes on page 326.

Managing Agile PLM Classes

The Agile Classes node provides a framework for classifying Agile PLM objects, such as parts, changes, and packages. Using Agile Java Client, you can define new subclasses for your organization. Although you can't use the Agile API to create new subclasses, you can read or modify any of the existing subclasses. For example, you can customize a subclass by defining the attributes that are visible in each table or on each page.

The Agile PLM classes framework is based on the types of objects that are created in Agile PLM. The objects that are available on your Agile PLM system depend on the Agile PLM agreement with Oracle.

Each Agile PLM class has at least one subclass. The following table lists Agile PLM base classes, classes, and Agile-supplied subclasses. Your Agile PLM system may include other user-defined subclasses.

Base Class	Classes	Predefined Subclasses
Changes	Change Orders	ECO
	Change Requests	ECR
	Deviations	Deviation
	Manufacturer Orders	MCO
	Price Change Orders	PCO
	Site Change Orders	SCO
	Stop Ships	Stop Ship
Customers	Customers	Customer
Declarations	Homogeneous Material Declarations	Homogeneous Material Declaration
	IPC 1752-1 Declarations	IPC 1752-1 Declaration

Base Class	Classes	Predefined Subclasses
	IPC 1752-2 Declarations	IPC 1752-2 Declaration
	JGPSSI Declarations	Japan Green Procurement Survey Standardization Initiative Declaration
	Part Declarations	Part Declaration
	Substance Declarations	Substance Declaration
	Supplier Declarations of Conformance	Supplier Declaration of Conformance
Discussions	Discussions	Discussion
File Folders	File Folders	File Folder
	Historical Report File Folders	Schedule Generated
		User Saved
Items	Documents	Document
	Parts	Part
Manufacturer Parts	Manufacturer Parts	Manufacturer Part
Manufacturers	Manufacturers	Manufacturer
Packages	Packages	Package
Prices	Published Prices	Contract
		Published Price
	Quote Histories	Quote History
Product Service Requests	Non-Conformance Reports	NCR
	Problem Reports	Problem Report
Projects	Activities	Phase
		Program
		Task
	Gates	Gate
Quality Change Requests	Audits	Audit
	Corrective and Preventive Actions	CAPA
Reports1	Custom Reports	Custom Report
	External Reports	External Report
	Standard Reports	Administrator Report
		Standard Report
Requests for Quote	Requests for Quote	RFQ
RFQ Responses	RFQ Responses	RFQ Response

Base Class	Classes	Predefined Subclasses
Sites	Sites	Site
Sourcing Projects	Sourcing Projects	Sourcing Project
Specifications	Specifications	Specification
Substances	Materials	Material
	Subparts	Subpart
	Substance Groups	Substance Group
	Substances	Substance
Suppliers	Suppliers	Broker
		Component Manufacturer
		Contract Manufacturer
		Distributor
		Manufacturer Representative
Transfer Orders	Automated Transfer Orders	ATO
	Content Transfer Orders	CTO
User Groups	User Groups	User Group
Users	Users	User

Note Report objects are not supported by the Agile API.

Concrete and Abstract Classes

Agile PLM super classes, such asItem and Change, are abstract classes that serve as the parent classes for other abstract classes, such as Parts Class, Documentation Class, and Engineering Change Order Class. Abstract superclasses and classes cannot be instantiated.

Concrete classes are user-defined subclasses that can be instantiated by the Agile API. Examples of concrete classes are Part, Document, ECO, and ECR.

When you load an object using the `IAgileSession.getObject()` method, you can specify either a concrete or an abstract Agile PLM class. For example, all of the following methods load the same specified part.

Example: Loading an object using abstract or concrete classes

```
try {
  IItem item;
  // Load a part using the Item base class
 item =
   (IItem)m_session.getObject(ItemConstants.CLASS_ITEM_BASE_CLASS,
   "1000-02");
  // Load a part using the Parts class
 item =
```

```
    (IItem)m_session.getObject(ItemConstants.CLASS_PARTS_CLASS, "1000-
    02");
   // Load a part using the Part subclass
  item =
    (IItem)m_session.getObject(ItemConstants.CLASS_PART, "1000-02");
 } catch (APIException ex) {
   System.out.println(ex);
 }
```

To get an array of classes, use the `IAgileClass.getAgileClasses()` method. You can specify a range of classes to return. For example, specify `IAdmin.CONCRETE` for the `range` parameter to return only concrete classes or `IAdmin.ALL` to return all classes.

Example: Getting classes

```
private void getConcreteClasses() throws APIException {
 IAgileClass[] classes =
   m_admin.getAgileClasses(IAdmin.CONCRETE);
 for (int i = 0; i < classes.length; i++) {
   System.out.println("Class Name : " + classes[i].getName());
   System.out.println("ID : " + classes[i].getId());
  }
}
void getAllClasses() throws APIException {
 IAgileClass[] classes =
   m_admin.getAgileClasses(IAdmin.ALL);
 for (int i = 0; i < classes.length; i++) {
   System.out.println("Class Name : " + classes[i].getName());
   System.out.println("ID : " + classes[i].getId());
  }
}
```

When you create a new object using the `IAgileSession.createObject()` method, you must specify a concrete Agile PLM class, that is, one of the user-defined subclasses. Remember, abstract classes cannot be instantiated. The following example shows how to create an object of the Part subclass.

Example: Creating a part

```
try {
 Map params =
    new HashMap();
 params.put(ItemConstants.ATT_TITLE_BLOCK_NUMBER, "1000-02");
 IItem item =
   (IItem)m_session.createObject(ItemConstants.CLASS_PART, params);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Referencing Classes

You can reference Agile PLM classes in the following ways:

- by object (an `IAgileClass`)

- by class ID constant, such as `ItemConstants.CLASS_PART` or `ChangeConstants.CLASS_ECO`. All Agile API constants are contained in classes that have a suffix name "Constants." For example, `ItemConstants` contains all constants related to `IItem` objects.
- by class name, such as "Part" or "ECO".

In general, avoid referencing classes by name for the following reasons:

- Class names can be modified.
- Class names are not necessarily unique. It's possible to have duplicate class names. Consequently, if you reference a class by name you may inadvertently reference the wrong class.
- Class names are localized; that is, the names are different for different languages.

Identifying the Target Type of a Class

Each class has a specified target type, which is the type of Agile PLM object that the class can create. For example, the target type for the Part subclass is `IItem.OBJECT_TYPE`. You can use the target type to classify the user-defined subclasses that have been defined in your Agile PLM system. For example, if you want to create a user interface that displays item classes, you can list the classes at run time by selecting those with the target type `IItem.OBJECT_TYPE`.

Example: Getting the target type for a class

```
private void getConcreteItemClasses() throws APIException {
 IAgileClass[] classes =
   m_admin.getAgileClasses(IAdmin.CONCRETE);
  for (int i = 0; i < classes.length; i++) {
   if (classes[i].getTargetType()
     == IItem.OBJECT_TYPE) {
    System.out.println("Class Name : " + classes[i].getName());
    System.out.println("ID : " + classes[i].getId());
     }
  }
}
```

There are two predefined concrete classes for the Item class, Document and Part. If your company hasn't added any Item subclasses to the Agile PLM system, the code in the previous example should print the following results:

```
Class Name : Document
ID : 9141
Class Name : Part
ID : 10141
```

Working with Attributes

Each object that you can retrieve in an Agile API program has a set of attributes. An attribute represents metadata for a particular business object. It defines the properties and values of the object. For example,

“Title Block.Number,” “Title Block.Description,” and “Title Block.Part Category” are three of the Title Block attributes for a Part.

When you create an instance of an object in your program, each `IAttribute` in your object classes is equivalent to a field, or an `ICell` object. `IAttribute` objects directly correspond with `ICell` objects for an object that you created or opened in your program. For more information about `ICell` objects, see Working with Data Cells on page 91.

Referencing Attributes

You can reference Agile PLM attributes in the following ways:

- by object (an `IAttribute`)
- by attribute ID constants

  All Agile API constants, including attribute ID constants, are contained in classes that have the suffix “Constants.” For example, `ItemConstants` contains all constants related to `IItem` objects.
- by fully qualified name, such as "Title Block.Number" or "Cover Page.Change Category"
- by short name, such as “Number”. However, attribute short names are not unique in Agile PLM. If you are referencing multiple attributes, you may run into a conflict if two different attributes have the same short name.

Note Because attribute names can be modified and referencing attributes by ID number or constant is difficult to identify or remember, Agile recommends using the APIName filed for this purpose. For information and procedures, see Accessing PLM Internal Data with Field on page 123. Many of the examples in this manual reference attributes by name because they were constructed before the introduction of this field.

The following example shows how to reference an attribute ID constant.

Example: Referencing an attribute ID constant

```
Integer attrID =
    ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION;
try {
    v = item.getValue(attrID);
    } catch (APIException ex) {
    System.out.println(ex);
}
```

A fully qualified attribute name is a string with the following format:

```
TableName.AttributeName
```

`TableName` is the name of the table in which the attribute appears. `AttributeName` is the current value for the Name property of an attribute. All attributes have default names, but the names can be changed. In particular, Page Two and Page Three attributes that have been made visible in your Agile PLM system are likely to have been assigned more meaningful names than “Text01,” “List01,” and “Date01.”

“Cover Page.Reason for Change” and “Title Block.Number” are two examples of fully qualified attribute names.

The following example shows how to reference to a fully qualified attribute name.

Example: Referencing an attribute name

```
Object v;
String attrName = "Title Block.Description";
try {
  v = item.getValue(attrName);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Note Attribute names are case-sensitive.

Retrieving Attributes

`IAttribute` objects are associated with a particular subclass. For example, the attributes for a Part are different from those of an ECO. Therefore, if you know the subclass of an object you can retrieve the list of attributes for it. The following table lists methods that can be used to retrieve attributes.

Method	Description
`IAgileClass.getAttribute()`	Retrieves the specified `IAttribute` object for a class
`IAgileClass.getAttributes()`	Retrieves an array of `IAttribute` objects for all tables of a class
`IAgileClass.getTableAttributes()`	Retrieves an array of `IAttribute` objects for a specified table of the class
`ITable.getAttributes()`	Retrieves an array of `IAttribute` objects for a table
`ICell.getAttribute()`	Retrieves the `IAttribute` object for a cell

The following example shows how to retrieve BOM table attributes.

Example: Retrieving BOM table attributes for the Part subclass

```
try {
  // Get the Part subclass
 IAgileClass partClass =
    (IAgileClass)m_admin.getAgileClass(ItemConstants.CLASS_PART);
  // Get the collection of BOM table attributes for the Part subclass
 IAttribute[] attrs =
   partClass.getTableAttributes(ItemConstants.TABLE_BOM);
} catch (APIException ex) {
  System.out.println(ex);
}
```

Another way to retrieve the attributes for a particular table is to first get the table, then get its attributes using the `ITable.getAttributes()` method.

Example: Retrieving the collection of BOM table attributes from the table

```
try {
  // Get Part P200
 IItem item =
```

```
    (IItem)m_session.getObject(IItem.OBJECT_TYPE, "P200");
   // Get the BOM table
  ITable bomTable =
    item.getTable(ItemConstants.TABLE_BOM);
   // Get BOM table attributes
  IAttribute[] attrs =
    bomTable.getAttributes();
 } catch (APIException ex) {
   System.out.println(ex);
 }
```

Retrieving Individual Attributes

If you know the attribute you want to retrieve, you can get it by using the `IAgileClass.getAttribute()` method. The following example shows how to get the "Cover Page.Reason Code" attribute for an ECO.

Example: Retrieving the "Cover Page.Reason Code" attribute

```
try {
  // Get the ECO subclass
 IAgileClass classECO =
   m_admin.getAgileClass("ECO");
  // Get the "Cover Page.Reason Code" attribute
 IAttribute attr =
   classECO.getAttribute(ChangeConstants.ATT_COVER_PAGE_REASON_CODE);
  // Get available values for Reason Code
 IAgileList availValues =
   attr.getAvailableValues();
} catch (APIException ex) {
  System.out.println(ex);
}
```

Editing the Property of an Attribute

Agile PLM classes have attributes, and attributes have properties. To modify the properties of an attribute for a particular subclass, follow these steps:

1. Use the `IAdmin.getAgileClass()` method to get an Agile PLM class.
2. Use the `IAgileClass.getAttribute()` method to get an attribute for the class.
3. Use the `IAttribute.getProperty()` method to get a property for the attribute.
4. Use the `IProperty.getValue()` method to get the current value for the property.
5. Use the `IProperty.setValue()` method to set a new value for the property.

Working with User-Defined Attributes

For each Agile PLM subclass, you can define additional attributes on the Page Two and Page Three tables. These user-defined attributes, also known as customer flex fields, behave the same as predefined Agile PLM attributes. You can retrieve them and edit their properties.

User-defined attributes are custom extensions to the Agile PLM system. Consequently, their IDs are not included in the `CommonConstants` class. However, you can view the base ID for any attribute, including user-defined attributes, in Agile Java Client. You can also write a procedure to programmatically retrieve the ID for a user-defined attribute at run time.

Working with Properties of Administrative Nodes

If you use the Agile API to retrieve a `INode` object, you can also view the `INode`'s property values. An `IProperty` object represents a single property for an administrative node. To return an array of all properties for a node, use the `INode.getProperties()` method.

The following example shows how to get the property value for the Reminder/Escalation Weekend Setting preference. The last part of this example converts the available list values for this SingleList property to a comma-delimited string.

Example: Getting Property values

```
 private void getReminderEscalationWeekendProp() throws APIException {
   //Get the General Preferences node
  INode node =
    m_admin.getNode(NodeConstants.NODE_PREFERENCES);
   //Get the Reminder/Escalation Weekend Setting property
  IProperty prop =
    node.getProperty(PropertyConstants.PROP_REMINDER_ESCALATION_WEEKEND_
    SETTING);
   //Get the Reminder/Escalation Weekend Setting property value
  Object value =
    prop.getValue();
    System.out.println("Reminder/Escalation Weekend Setting : " +
    value);
  IAgileList avail =
    prop.getAvailableValues();
   if (avail != null) {
    String strAvail =
      listToString(avail);
    System.out.println("Available Values : " + strAvail);
   }
}
    private String listToString(IAgileList list)
    throws APIException {String strList = "";
 Collection children =
    list.getChildNodes();
   for (Iterator it =
    children.iterator();it.hasNext();) {
     IAgileList childList =
      (IAgileList)it.next();
     strList =
      strList + childList.getValue();
     if (it.hasNext()) {
       strList = strList + ", ";
     }
```

```
    }
    return strList;
  }
```

The `SingleList` and `MultiList` properties are different from other types of properties. You cannot use the `IProperty.getValue()` and `IProperty.setValue()` methods to directly modify a property that contains a list of values. Instead, you use the `IAgileList.setSelection()` method to select a list node, and then use the `IProperty.setValue()` method to set the value. For more information about how to modify SingleList and MultiList properties, see Getting and Setting List Values on page 96.

Managing Users

Users are data objects that you can create, like items and changes. Consequently, you can work with users directly without traversing the administrative node hierarchy. If you have the proper Agile PLM privileges, you can create, modify, and delete users. For example, you could create a program that periodically synchronizes Agile PLM users with data available from a corporate directory.

Getting All Users

To retrieve all Agile PLM users, run a query for User objects. The following example retrieves all users and prints the username, first name, and last name for each user.

Example: Getting all users

```
private void getAllUsers() throws APIException {
  IQuery q =
    (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,  "select * from
    [Users]");
  ArrayList users =
    new ArrayList();
  Iterator itr =
    q.execute().getReferentIterator();
   while (itr.hasNext()) {
     users.add(itr.next());
   }
  for (int i = 0; i < users.size(); i++) {
   IUser user =
     (IUser)users.get(i);
   System.out.println(
    user.getValue(UserConstants.ATT_GENERAL_INFO_USER_ID) + ", " +
    user.getValue(UserConstants.ATT_GENERAL_INFO_FIRST_NAME) + ", " +
    user.getValue(UserConstants.ATT_GENERAL_INFO_LAST_NAME)
    );
   }
}
```

Creating a User

A user is like other data objects that you can create with the Agile API. To create a user, you define the user's parameters and pass them to the `IAgileSession.createObject()` method. The

required parameters you must specify are username and login password. You can also specify other user attributes, which are listed in the `UserConstants` class.

Note If an LDAP directory server is used to authenticate users for your Agile PLM system, you can create only supplier users, which have restricted access to the Agile PLM system. You must create and maintain other users on the directory server.

The passwords you specify for a new user are default values. If you specify an approval password, it must be different from the login password unless the `UserConstants.ATT_GENERAL_INFO_USE_LOGIN_PASSWORD_FOR_APPROVAL` cell is set to "Yes." The user can change passwords later.

Example: Creating a user

```
public IAgileSession m_session;
public IAdmin m_admin;
public AgileSessionFactory m_factory;

private void userTest() {
  try {
    //Add code here to log in to the Agile Application Server
       //After logging in, create a new user IUser user =
   createUser("akurosawa");
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
private IUser createUser(String newUser) throws APIException {
  //Create the new user
   Map params =
     new HashMap();
   params.put(UserConstants.ATT_GENERAL_INFO_USER_ID, newUser);
   params.put(UserConstants.ATT_LOGIN_PASSWORD, "agile");
   IUser user =
     (IUser)session.createObject
        (UserConstants.CLASS_USER, params);
  return user;
}
```

By default, when you create a new user it's assigned the Concurrent user category and the My User Profile role, a combination that allows the user to view objects but not to create, approve, or modify them. To create and modify objects, the user must be assigned roles with the appropriate create or modify privileges. For an example showing how to change a user's Role settings, see Configuring User Settings on page 338.

Creating a Supplier User

Supplier users are assigned to the Restricted user category by default, which restricts their access to the Agile PLM system. The Restricted user category allows supplier users to respond to RFQs and use other features of Agile Product Cost Management (PCM).

To create a supplier user, define the user's parameters and pass them to the `IAgileSession.createObject()` method. You must specify the username, login password,

and supplier name. You can also specify other user attributes, which are listed in the `UserConstants` class.

Example: Creating a supplier user

```
private IUser createSupplierUser(String userName, String supplier)
throws APIException {
 HashMap userParams =
       new HashMap();
   userParams.put(UserConstants.ATT_GENERAL_INFO_USER_ID, userName);
   userParams.put(UserConstants.ATT_LOGIN_PASSWORD, "agile");
   userParams.put(UserConstants.ATT_SUPPLIER, supplier);
   return (IUser)m_session.createObject(UserConstants.CLASS_USER,
   userParams);
}
```

Saving a User to a New User

You can use the `IDataObject.saveAs()` method to save an existing user to a new user. The `saveAs()` method serves as a handy shortcut because it allows you to assign a new user the same roles, privileges, and sites as an existing user. When you use the `saveAs()` method to save a user, you must specify parameters for the new user's user name and login password.

Example: Saving an object as a new object

```
private void saveAsUser(IUser user, String newUserName) {
 try {
    //Set parameters for the new user
  Map params =
     new HashMap();
   params.put(UserConstants.ATT_GENERAL_INFO_USER_ID, newUserName);
   params.put(UserConstants.ATT_LOGIN_PASSWORD, "agile");
    // Save the new user
user.saveAs(UserConstants.CLASS_USER, params);
  } catch (APIException ex) {
    System.out.println(ex);
  }
}
```

Checking for Expired Passwords

You can set Agile PLM passwords to expire at a specified time. When a user's login password expires, the user cannot log in to the Agile Application Server. If the login password has expired, the Agile API program can allow the user to specify a new password. The following example shows how to check for an Agile API error related to an expired password.

Example: Checking for Expired Passwords

```
HashMap params = new HashMap();
   params.put(AgileSessionFactory.USERNAME, user.getName());
   params.put(AgileSessionFactory.PASSWORD, "agile");
   params.put(AgileSessionFactory.URL,"http://localhost/Agile");
// Pass one element array to get the login exception
   APIException[] exception = new APIException[1];
   params.put(AgileSessionFactory.STATUS,  exception);
   IAgileSession session=AgileSessionFactory.createSessionEx(params);
   System.out.println(exception[0].getErrorCode());
```

```
// When error code is ExceptionConstants.APDM_PWDNOCHANGE
    If(exception[0].getErrorCode() ==
    ExceptionConstants.APDM_PWDNOCHANGE) {
// Prompt for new password and change the passsword
    Session.getUser().changeLoginPassword (oldPassword, newPassword);
}
```

Configuring User Settings

An `IUser` object, unlike administrative nodes, is a dataobject. Therefore, an `IUser` object has data cells, not properties, and you use the `ICell` interface to configure a user's settings. The following example shows how to get visible cells on the General Info and Page Two tables for a user. To access cells on other user tables, use the `IDataObject.getTable()` method to load the table.

Example: Getting user cells for General Info and Page Two

```
private void getUserCells(IUser user) throws APIException {
 ICell[] cells = user.getCells();
  for (int i = 0; i < cells.length; i++) {
   System.out.println(cells[i].getName() + " : " +
   cells[i].getValue());
  }
}
```

Two important settings for a user are User Category and Roles. The User Category setting defines the broad range of actions a user can perform on the Agile PLM system. Select from one of the following User Category values:

- Power – User can log in to the server at any time with unrestricted use of the Agile PLM system. Power users are not subject to the limited number of concurrent users.
- Concurrent – User can log in to the server only if a concurrent user agreement is available.

  Note Licenses are controlled by your agreement with Oracle.

- Restricted – User has restricted access to the Agile PLM system. Supplier users are by default assigned the Restricted category, which allows them to respond to RFQs and use other features of Agile Product Cost Management (PCM). Restricted users are not subject to the limited number of concurrent users.

The Roles setting further defines the capabilities of a user, assigning roles and privileges. A user won't be able to create objects without the proper roles and privileges. For more information about Agile PLM user roles, and privileges, refer to the *Agile PLM Administrator Guide*.

The following example shows how to set a user's User Category and Roles settings.

Example: Setting the User Category and Roles settings for a user

```
private void setCategory(IUser user) throws APIException {

  //Get the User Category cell
 ICell cell =
   user.getCell(UserConstants.ATT_GENERAL_INFO_USER_CATEGORY);
  //Get the available values for the cell
 IAgileList license  =
   cell.getAvailableValues();
```

```
   //Set the selected value to "Concurrent"
    license.setSelection(new Object[] { "Concurrent" });
   //Change the cell value
   cell.setValue(license);
  }
 private void setRoles(IUser user) throws APIException {
   //Get the Role cell
  ICell cell =
    user.getCell(UserConstants.ATT_GENERAL_INFO_ROLES);
   //Get the available values for the cell
  IAgileList roles  =
    cell.getAvailableValues();
   //Set the selected roles to Change Analyst and Administrator
    roles.setSelection(new Object[] {"Change
    Analyst","Administrator","My User Profile"});
   //Change the cell value
    cell.setValue(roles);
  }
```

Resetting User Passwords

Administrators with User Administrator privileges can reset the password of other users to a new value. Users without this privilege are not able to reset user passwords. This feature enables resetting large numbers of passwords in a batch mode, which is preferable to manually, using them one at a time using UI.

The `changeLoginPassword()` method which supports this feature allows passing the null value instead of the current password value. The following example shows how to use this method to reset a user's password using `null` instead of the current password.

Example: Resetting a password to a new value

```
public void changeLoginPassword(null, String newPassword)
            throws APIException;
```

Deleting a User

To delete a user, use the `IDataObject.delete()` method. Like other dataobjects, an object deleted for the first time is "soft-deleted," which means it is disabled but not removed from the database. The Agile Application Server does not allow you to permanently delete a user.

Example: Deleting a user

```
private void removeUser(IUser user) throws APIException {
  user.delete();
  user = null;
}
```

Note In Agile Java Client, deleted users can be listed by choosing Admin > User Settings > Deleted Users.

Managing User Groups

A user group is an object that contains a list of Agile PLM users. You can use user groups to define project teams, departments, and global groups and their assigned users. User groups are not site-related, like items and changes, but you can create groups of users based on their location. Whenever you add a user to a user group, that change is reflected in the user's Groups setting, whose attribute ID is `UserConstants.ATT_GENERAL_INFO_GROUPS`.

Note In Agile Clients such as Agile Web Client, you can send an object, such as a change, to a user group. The Agile API does not support sending objects to user groups. However, you can retrieve users from the Users table of a User Group object and then send them an object.

Getting All Users Groups

To retrieve all Agile PLM user groups, run a query for User Group objects. You can iterate through the user groups to find a particular group. The following example retrieves all user groups and prints the name, description, maximum number of users, and enabled status for each user group.

Example: Getting all user groups

```
private void getAllUserGroups() throws APIException {
 IQuery q =
   (IQuery)m_session.createObject(IQuery.OBJECT_TYPE,  "select * from
   [User Groups]");
 ArrayList groups = new ArrayList();
 Iterator itr =
   q.execute().getReferentIterator();
  while (itr.hasNext()) {
    groups.add(itr.next());
  }
  for (int i = 0; i < groups.size(); i++) {
  IUserGroup ug  =
    (IUserGroup)groups.get(i);
  System.out.println(
   ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_NAME) + ", " +
   ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_DESCRIPTION) + ", "
   +
   ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_MAX_NUM_OF_NAMED_USE
   RS) + ", " +
   ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_STATUS)
  );
  }
}
```

Creating a User Group

A user group, like a user, is a dataobject and not an administrative node on the Agile Application Server. To create a user group, you define the user group's parameters, such as its name, and pass the parameters to the `IAgileSession.createObject()` method. The only required

parameter you must specify is the name, whose attribute ID is `UserGroupConstants.ATT_GENERAL_INFO_NAME`. You can also specify other user attributes, which are listed in the `UserGroupConstants` class. To enable a user group, make sure the Enabled cell is set to Yes.

After creating a user group, you need to add users to the Users table to make the group meaningful. To create a new row in the Users table, use the `ITable.createRow(java.lang.Object)` method.

Example: Creating a user group

```
public IAgileSession m_session;
public IAdmin m_admin;
public AgileSessionFactory m_factory;

private void userGroupTest() throws APIException {
  //Add code here to log in to the Agile Application Server
  //After logging in, create a new user group IUserGroup group =
createGroup("Swallowtail Project");
  //Add users to the Western project group
  IUser[] selUsers = new IUser[] {
    m_session.getObject(IUser.OBJECT_TYPE, "jford"),
    m_session.getObject(IUser.OBJECT_TYPE, "hhawkes"),
    m_session.getObject(IUser.OBJECT_TYPE, "speckinpah")
  };
  addUsers(group, selUsers);
}
private IUserGroup createGroup(String groupName) throws APIException {
  //Create the user group
  IUserGroup group =

(IUserGroup)m_session.createObject(UserGroupConstants.CLASS_USER_GROUP,
groupName);
  //Enable the user group
  ICell cell =
group.getCell(UserGroupConstants.ATT_GENERAL_INFO_STATUS);
  IAgileList list = cell.getAvailableValues();
  list.setSelection(new Object[] { "Active" });
  cell.setValue(list);

  return group;
}
private void addUsers(IUserGroup group, IUser[] users) throws
APIException {
  ITable usersTable = group.getTable(UserGroupConstants.TABLE_USERS);
  for (int i = 0; i < users.length; i++) {
    IRow row = usersTable.createRow(users[i]);
  }
}
```

User groups can be global or personal. Global user groups are accessible to all Agile PLM users.

Personal user groups are accessible only to the person who created the group. The following example shows how to make a user group global.

Example: Making a user group global

```
private void setGlobal(IUserGroup group) throws APIException {
  //Get the Global/Personal cell
  ICell cell =
group.getCell(UserGroupConstants.ATT_GENERAL_INFO_GLOBAL_PERSONAL);
  //Get the available values for the cell
  IAgileList values  = cell.getAvailableValues();

  //Set the selected value to "Global"
  values.setSelection(new Object[] { "Global" });

  //Change the cell value
  group.setValue(UserGroupConstants.ATT_GENERAL_INFO_GLOBAL_PERSONAL,
values);
}
```

Adding a User Group to the User's User Group Table

`IUserGroup` can not be passed to `createRow()` to add a user group to the user's user group table. You must use a `Map` as shown in the following example.

Example: Using a Map to add a User Group to a User Group Table

```
ITable ugTable = user.getTable(UserConstants.TABLE_USERGROUP);
Map map = new HashMap();
map.put(UserConstants.ATT_USER_GROUP_GROUP_NAME, ug.getName());
ugTable.createRow(map);
```

Listing Users in a User Group

The users contained within a user group are listed on the Users table. Therefore, to get the list of users in the user group, use the `IDataObject.getTable()` method and then iterate over the table rows to access data for each user. The following example shows how to list the users in a user group.

Example: Listing the users in a user group

```
private void listUsers(IUserGroup group) throws APIException {
 ITable usersTable =
   group.getTable(UserGroupConstants.TABLE_USERS);
 Iterator it =
   usersTable.iterator();
 while (it.hasNext()) {
   IRow row = (IRow)it.next();
   System.out.println(row.getValue(UserGroupConstants.ATT_USERS_USER_NA
   ME));
  }
}
```

Chapter 21

Developing Process Extensions

This chapter includes the following:

- About Process Extensions 343
- Developing Custom Autonumber Sources 344
- Developing Custom Actions 347
- Defining and Deploying URL-Based Process Extensions 353
- Creating an External Report 359
- Deploying Process Extensions in a Clustered Environment 360
- Best Practices for Copying third Party JAR Files 360
- Process Extensions FAQ 363

About Process Extensions

Process extensions (PX) is a framework for extending the functionality of the Agile PLM system. The functionality can be server-side extensions, such as custom workflow actions and custom auto numbering, or extensions to client-side functionality, such as external reports or new commands added to the Actions menu or the Tools menu. Regardless of the type of functionality a process extension provides, all custom actions are invoked on the Agile Application Server rather than the local client.

Note In addition to server-side functionalities you can develop in the PX framework, Agile PLM's Event framework also supports developing extensions using Java and Groovy Script, called Java PXs and Script PXs respectively. Although you can migrate some of the custom actions developed in PX framework to Events framework, the two frameworks have their own unique interfaces and are not the same.

Process extensions enable the Agile PLM server and Agile PLM users to connect to external systems. You can also use process extensions to add functionality not provided by the standard Agile PLM client. Using a simple yet powerful approach, process extensions open the Agile PLM system, allowing you to tailor the application to your business requirements.

A process extension is either a Java class deployed on the Agile Application Server, or a link to a URL. The URL can be a simple Web site or the location of a Web-based application.

You can use process extensions to create:

- Custom reports
- User-driven and workflow-triggered custom actions
- Custom tools accessible through Agile PLM clients
- Custom auto numbering

What types of custom actions and tools can you create within the process extensions framework? Technically, there are few limitations on what a custom action can do. After all, you define it. Consequently, it's an open-ended solution. Agile Solutions Delivery and Agile partners can help your company develop the process extensions it needs.

Multiple process extensions can be linked together in a chain with each process extension performing a discrete business function. Process extensions can also be used to make requests to Web services, such as services built with Agile's Web service extensions framework.

There are five integration points for process extensions available in Agile PLM clients. You can invoke process extensions from the following areas:

- External reports
- Actions menu
- Tools menu
- Workflow Status
- Autonumber sources

Developing Custom Autonumber Sources

This section describes how to develop a custom autonumber source.

Most Agile PLM object classes have at least one default autonumber source that lets you create a new object and automatically number it with the next number in the sequence. Autonumbers can have an alphanumeric prefix and/or suffix. You can also specify the length of the autonumber (a string) and which numeric characters to use.

Despite the flexibility that autonumbers provide, some companies have specific numbering requirements that can't be accommodated by Agile PLM's standard autonumbering capabilities. Such companies can define custom autonumber sources and add them to the Agile PLM system using the process extensions framework.

If you have administrator privileges, you can define autonumber sources in Agile Java Client. An autonumber source can use the client's standard numbering capabilities, or it can be associated with a custom autonumber source. When an Agile PLM client uses a custom autonumber source to create a new object, the Agile Application Server invokes the custom Java code to generate the number.

Defining a Custom Autonumber Source

To define a custom autonumber source, create a Java class that implements the `ICustomAutoNumber` interface, a server-side API in the `com.agile.px` package. The code should define the autonumbering logic, for example, prefix, suffix, number of digits, character set, and so on, and the persistence mechanism. Regarding persistence, the location where your custom autonumber source stores numbers is entirely up to your program. For example, you can store numbers in a SQL database like Oracle or in a file.

The Agile PLM server gets the next number from the custom autonumber source by calling the `getAutoNumber()` method, which must be provided in your class. The following example shows

how to implement a Java class for a custom autonumber source.

Example: Defining the class for a custom autonumber source

```
package autonumbers;

import com.agile.px.*;
import com.agile.api.*;

public class ResistorNumber implements ICustomAutoNumber
{
    public ActionResult getAutoNumber(IAgileSession session, INode
    actionNode)
    {
      String num;
    // Write your code here to define the custom autonumber source for
    Resistors
    return new ActionResult(ActionResult.STRING, num);
    }
}
```

Packaging and Deploying a Custom Autonumber Source

After you develop classes for a custom autonumber source, follow these instructions to properly package and deploy them.

To package and deploy a custom autonumber source:

1. Use your Java development environment or the Java Archive tool (or JAR tool) to create one or more JAR files for the custom autonumber source. Make sure the JAR file(s) includes a META-INF/services directory that contains a file named `com.agile.px.ICustomAutoNumber`, which is a text file that lists the fully qualified Java class names, one class per line, for the custom autonumber source.

   Multiple custom autonumber sources can be included in one package. For example, the `com.agile.px.ICustomAutoNumber` file could look like this:

   ```
   autonumbers.ResistorNumber
   autonumbers.CapacitorNumber
   autonumbers.DiodeNumber
   ```

   Note Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file has a name with all uppercase or all lowercase characters. Otherwise, the custom autonumber source will not be deployed.

2. Place the JAR file(s) in the agile_home/integration/sdk/extensions folder on the same computer where the Agile Application Server is installed.

   Note If you have several application servers in a clustered environment, you must deploy process extension files on each server in the cluster.

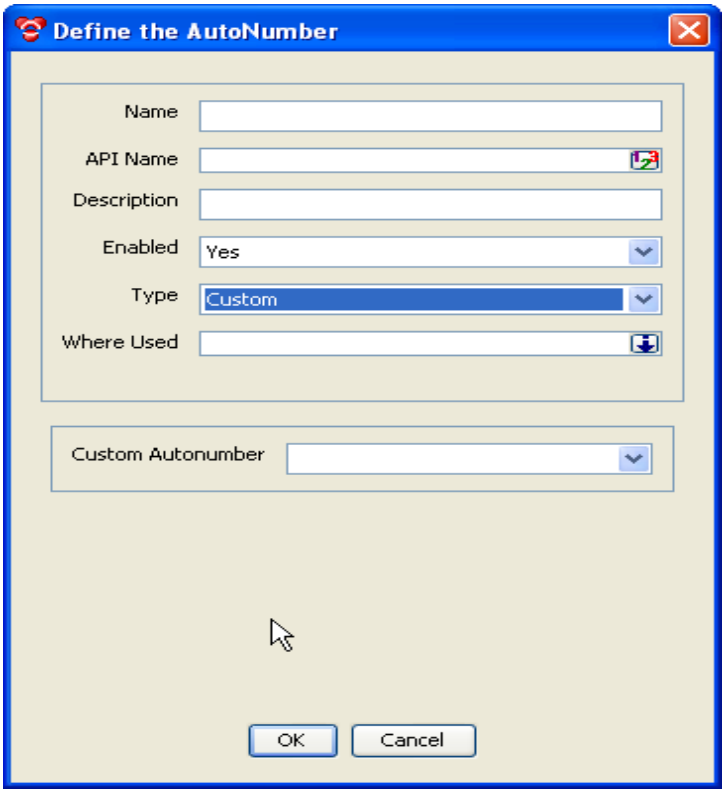
Configuring Custom Autonumber Sources in Java Client

In Agile Java Client, you can define autonumber sources in the Admin module. To configure Agile PLM system settings, you must have an administrator account.

To add a custom autonumber source:

1. Log in to Agile Java Client as an administrator.
2. Click the Admin tab.
3. Go to Settings > Data Settings > AutoNumbers.
4. Click the AutoNumbers node.
5. In the AutoNumbers window, click . The Define the Autonumber dialog box appears.

Figure 19: Define the Autonumber dialog box



6. Provide the following information:
   - Name – Type the name of the autonumber source.
   - API Name – This field is automatically filled in after completing the Name field. Refer to Accessing PLM Metadata with APIName Field in *SDK Developer Guide - Using the APIs*.
   - Description – Type a brief description of the autonumber source.
   - Enabled – Select Yes or No.
   - AutoNumber type – Select Custom. This activates the Custom AutoNumber field.

- Where Used – Select the subclass(es) that can use the autonumber source.
- Custom AutoNumber – Select a custom autonumber source from the list.

7. Click OK to save the autonumber definition.

Assigning Autonumber Sources to a Subclass

When you define an autonumber source, you can specify the subclasses where it's used in the Where Used field. You can also assign an autonumber source to a subclass in the Classes node.

To assign autonumber sources to a subclass:

1. Log in to Agile Java Client as an administrator.
2. Click the Admin tab.
3. Open the Data Settings folder.
4. Open the Classes node.
5. In the Classes window, double-click a subclass. The subclass window appears.
6. In the Autonumber Source field, click . A popup window appears.
7. Select autonumber sources in the Choices list, and then click to move them into the Selected list. When you are finished, click OK.
8. Click Save to save settings.

Developing Custom Actions

This section describes how to develop custom actions in Java classes. The Agile PLM clients can make direct method calls into these classes to perform the actions.

You can initiate a custom action from the following areas of Agile PLM clients:

- Actions menu
- Tools menu
- External reports
- Workflow Status

Defining a Custom Action

To define a custom action, create a Java class that implements the `ICustomAction` interface, a server-side API in the `com.agile.px` package. The code should define the action to perform. The Agile PLM server initiates the action by calling the `doAction()` method, which must be provided in your class.

The following example shows the code for a HelloWorld class. When the `doAction()` method is called, the method returns "Hello World." If you invoke the HelloWorld custom action from Actions menu, the string "Hello World" will be logged to the object's History table. If you invoke the

HelloWorld custom action from a workflow, the string “HelloWorld” will be logged to the change order’s History table when it enters the appropriate workflow status.

Example: Defining a HelloWorld class for a custom action

```
package actions;

import com.agile.px.*;
import com.agile.api.*;

public class HelloWorld implements ICustomAction
{
    public ActionResult doAction(IAgileSession session, INode
    actionNode,
            IDataObject affectedObject)
    {
         return new ActionResult(ActionResult.STRING, "Hello World");
    }
}
```

The above HelloWorld class does not perform a useful action, it simply demonstrates how to implement the class for a custom action.

Formatting New Lines (Line Breaks) in PLM Clients

You can configure new lines in `ActionResult` outputs in Web and Java clients using the `'\n'` character.

Note SDK only supports the character `'\n'` for new lines in both Web and Java clients.

Example: Formatting new lines in SDK string outputs

1. Create a Process Extension that includes with the following `ActionResult` string.

   ```
   return ActionResult(ActionResult.STRING, "Hello \n This example tests
   formatting new lines \n in the SDK");
   ```

2. Run the Process Extension in the Web and Java clients from Actions menu and Tools menu.

   The output should be as below

   ```
   Hello
   This example tests formatting new lines
   in the SDK
   ```

Custom Actions and User Sessions

When an Agile PLM client invokes a process extension, it does so within the current user’s session. Therefore, the process extension should not create any additional `IAgileSession` objects using the Agile API within the process extension code or any code directly invoked from the process extension. Stated simply, process extensions never directly create new Agile PLM sessions.

If you have written a Web service extension (WSX) and want to make use of that code from within a process extension, you can directly invoke Java methods contained in WSX classes without using the Web services infrastructure, provided those methods do not create a new `IAgileSession` object.

Do not mix Process extension (PX) invocations with Web service extension (WSX) invocations. The

PX code must not invoke any WSX code directly, especially when the PX and WSX reside in the same application container. If a process extension makes use of Web services, that WSX will likely create a new Agile PLM session, which is distinct from the session used by the process extension.

URL-based process extensions can call an external application that communicates with the Agile PLM server and performs some action upon the currently selected business object. To perform such an action, the external application can use the Agile API to create another Agile PLM session. For more information, see Creating an Agile PLM Session from the Target System on page 355.

Packaging and Deploying a Custom Action

After you develop classes for a custom action, follow these instructions to properly package and deploy them.

To package and deploy a custom action:

1. Use your Java development environment or the Java Archive tool (or JAR tool) to create one or more JAR files for the custom action. Make sure the JAR file(s) includes a META-INF/services directory that contains a file named `com.agile.px.ICustomAction`, which is a text file that lists the fully qualified Java class names, one class per line, for the custom action.

   Multiple custom actions can be included in one package. For example, the `com.agile.px.ICustomAction` file could look like this:

   ```
   actions.HelloWorld
   actions.RFQConsolidation
   actions.RefreshCustomerFromCRM
   actions.StartMfg
   actions.ObsoletePartReplacer
   actions.WorkflowConflictResolver
   ```

   Note Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file has a name with all uppercase or all lowercase characters. Otherwise, the custom action will not be deployed.

2. Place the JAR file(s) in the `agile_home/integration/sdk/extensions` folder on the same computer where the Agile Application Server is installed.

   Note If you have several application servers in a clustered environment, you must deploy process extension files on each server in the cluster.

Roles and Privileges for Custom Actions

When you configure a custom action in Agile Java Client, you can specify the roles it uses. By default, a custom action uses the roles and privileges of the current user. However, you can configure a custom action to have expanded privileges. This is an important feature of process extensions. In effect, you can enforce the business logic of a custom action by granting it more privileges than those given to ordinary users. When a custom action is invoked in the Agile PLM client, its roles and privileges override the roles and privileges of the current user. Once the custom action is completed, the client reverts to the user's roles and privileges.

User Privileges for Configuring Process Extensions

To configure a Process Extension, you must have necessary user privileges to get the user's language setting. If a PX fails, the error message should display in the user's current language. If the user's roles are not set to include the privilege to load current user object info, the server will display all messages in the default system language.

Configuring Custom Actions in Agile Java Client

In Agile Java Client, you can define custom actions in the Admin module. To configure Agile PLM system settings, you must log in as a user with administrator privileges.

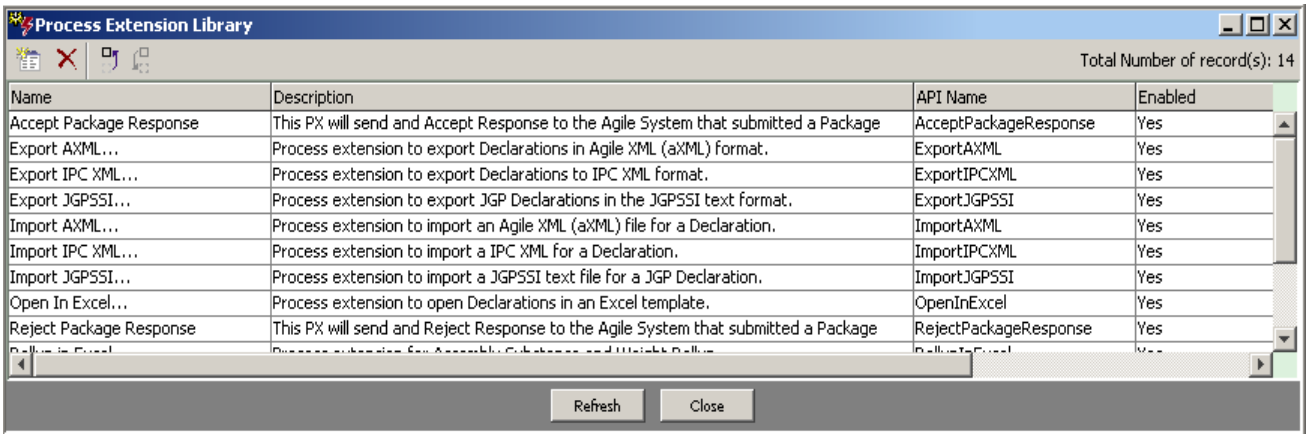
Using the Process Extension Library

The Process Extension Library is where you define the custom actions that can be used in Agile PLM clients. When you add a custom action to the Process Extension Library, you specify how to initiate that action from the client.

To add a custom action to the Process Extension Library:

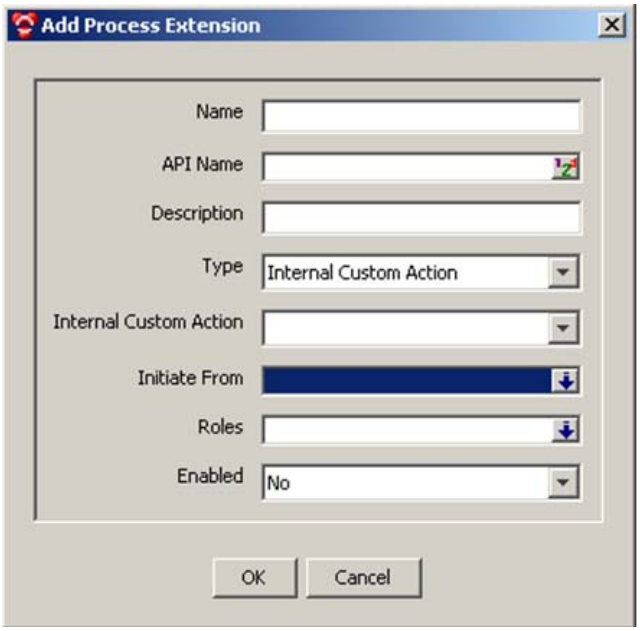
1. Log in to Agile Java Client as an administrator.
2. Click the Admin tab.
3. Open the Data Settings folder.
4. Open the Process Extensions node.

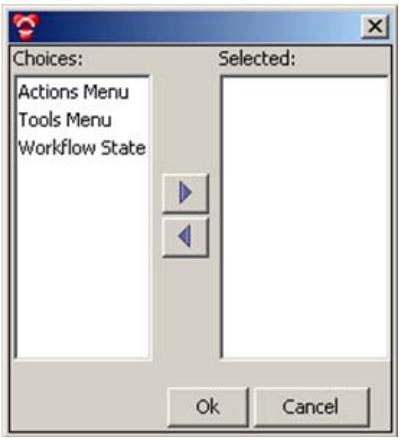
Figure 20: The Process Extension Library



5. In the Process Extension Library window, click the Add Process Extension button to open the Add Process Extension dialog box.

Figure 21: The Add Process Extension dialog





6. Type the following information:
   - Name – Type the name of the process extension.
   - API Name – This field is automatically filled in after completing the Name field. Refer to Accessing PLM Metadata with APIName Field in *SDK Developer Guide - Using the APIs*.
   - Description – Type a brief description of the process extension.
   - Type – Select Internal Custom Action. This activates the Internal Custom Actions field.
   - Internal Custom Action – Select a custom action from the list.
   - Initiate From – Select one or more locations from which the process extension can be initiated. Choose from the following options:

- Actions menu – Allows you to select the custom action from the Actions menu of a properly configured class.
- External report – Allows you to generate a report by accessing an external resource or URL. If the process extension is an internal custom action, the External Report option is unavailable.
- Tools menu – Allows you to select the custom action from the Tools menu.
- Workflow status – Invokes the custom action whenever a properly configured workflow enters a particular status.

  If you specify that a process extension is initiated from the Actions menu or a workflow status, you can configure subclasses or workflows to use it. If you specify that a process extension is used to generate an external report, you can use Agile Web Client to create the report. If you specify that a process extension is initiated from the Tools menu, it is always available in the Agile PLM client.

- Roles – Select one or more roles to use for the custom action. To use the roles and privileges of the current user, leave this field blank. To temporarily override roles and privileges of the current user, select one or more roles. Once the custom action is completed, the client reverts to the current user's roles and privileges.
- Enabled – Select Yes or No.

1. Click OK to save the new process extension.

Assigning Process Extensions to Classes

To add custom actions to the Actions menu of an Agile PLM object (such as a Part or an ECO), you configure the object's class. Each base class, class, and subclass has a Process Extensions tab. The custom actions that you assign to a class must be previously defined in the Process Extension Library.

Process Extensions are inherited from classes and base classes. Consequently, if you assign a process extension to a base class, it is also assigned to classes and subclasses beneath the base class.

Note Process extensions can be assigned to only one level in a class hierarchy. For example, if a process extension is assigned to the Part subclass, it can't be assigned to the Item base class.

To assign process extensions to a class:

1. Log in to Agile Java Client as an administrator.
2. Click the Admin tab.
3. Open the Data Settings folder.
4. Open the Classes node.
5. In the Classes window, double-click a base class, class, or subclass.
6. Click the Process Extensions tab.
7. In the toolbar, click . The Assign Process Extension dialog box appears.
8. Select custom actions in the Choices list, and then click to move them into the Selected list. When you are finished, click OK.
9. Click OK to save settings.

Assigning Process Extensions to Workflow Statuses

For each workflow status except the Pending status, you can assign one or more custom actions that are initiated when the workflow enters that status. The custom actions you assign to a workflow status must be previously defined in the Process Extension Library.

Note Automated Transfer Orders (ATOs) do not support workflow-triggered process extensions.

To assign process extensions to a workflow status:

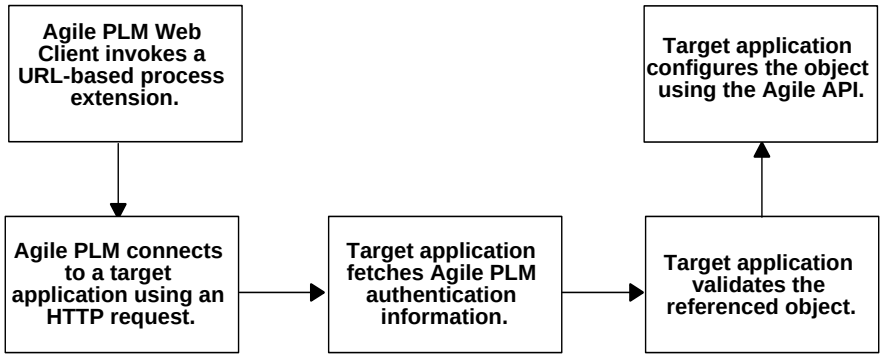
1. Log in to Agile Java Client as an administrator.
2. Click the Admin tab.
3. Open the Workflow Settings folder.
4. Open the Workflows node.
5. In the Workflows window, double-click a workflow
6. Click the Status tab.
7. Select a status other than Pending. The selected status updates the Workflow Criteria properties table that appears below the status table.
8. Double-click the selected status in the Workflow Criteria properties table.
9. In the Process Extensions list, click . A popup window appears.
10. Select custom actions in the Choices list, and then click to move them into the Selected list. When you are finished, click OK.
11. Click Save to save settings.

Defining and Deploying URL-Based Process Extensions

URL-Based Process Extensions are used by Agile Web Client to provide access from the Web Client to external applications. When the Agile PLM Web Client invokes a process extension that references a URL, the client displays the Web page in a new browser window.

What types of Web-based applications could be used for URL-based process extensions? Again, there are few limitations. One example might be a Web-based application that performs business rules validation for an Agile PLM object and updates the object accordingly. The following figure shows the program flow of such a process extension.

Figure 22: Process flow for a possible URL-based process extension



You can also use URL-based process extensions to reference a Web-based report engine. To create an external report that uses a URL-based process extension, choose Create > Report > External in Agile Web Client. For more information, see Creating an External Report on page 359.

Note Agile Java Client does not support URL PXs.

Before Building a URL-Based Process Extension

Please note the following requirements and constraints when building a URL-based process extension:

- URL-based process extensions cannot be initiated by a change in a workflow, because an Agile PLM client may not be active to trigger the change in status
- URL-based process extensions are not supported for Souring projects (`IProject`)

Defining a URL-Based Process Extension

To define a URL-based process extension do as follows:

1. Log into the Agile Java Client as an administrator.
2. Click the Admin tab.
3. Open the Data Settings folder.
4. Open the Process Extensions node.
5. In the Process Extension Library window, click . The Add Process Extension dialog box appears.
6. Enter the following information:
   - Name — Type the name of the process extension.
   - Description — Type a brief description of the process extension.
   - Type — Select the URL.
   - Address — Specify the address of a Web page. You must specify the complete URL, including the protocol. For example, to specify the Agile Corporation Web site, you would type "http://www.agile.com", not "www.agile.com".
   - Initiate From — Select one or more locations from which the Web page can be initiated. Choose from the following options:
     - Actions menu — Allows you to select the Web page from the Actions menu of a properly configured class.
     - Dashboard — See Developing Dashboard Management Extensions on page 387.
     - External report — Use this to generate a report by accessing the Web page.
     - Tools menu — Use this to select the Web page from the Tools menu.

     If you specify that a process extension is initiated from the Actions menu, you can configure subclasses to use it. If you specify that the process extension is used to generate an external report, you can use the Agile Web Client to create the report. If you specify that the process extension is initiated from the Tools menu, it is available at all times in the Agile PLM client.
   - Enabled — Select Yes or No.
7. Click OK to save the new process extension.

Passing Encoded Agile PLM Information to Other Applications

Agile SDK 9.2.2 does not support single sign-on via password protected external application servers.

Important	Agile Web Client can propagate encoded user credentials which can be reused by the SDK when your PX application uses the Agile SDK. If you want to password protect access to an external Application Server, you need to hard code the username and password to access the external servlet into your code.

If a URL-based process extension is initiated from an object's Actions menu, the object's composite key and class ID, as well as the current username, are encoded in the URL using the GET method. The client encodes the data as ID=value pairs and appends it to the end of the URL. Each ID is prefixed with "agile," as shown in the following example.

```
http://www.acoolwebsite.com/?agile.username=wangsh&agile.classId=10141&
agile.1001=
➥1000-02&agile.1014=A&agile.siteName=Taipei
```

Note	Unlike the Actions menu, there isn't an Agile PLM object associated with commands on the Tools menu. Consequently, if a URL-based process extension is initiated from the Tools menu, the URL is not augmented with encoded object data.

In addition to information encoded in the URL of a URL-based process extension, the encrypted username and its associated password are available from the j_username and j_password cookies, respectively, which are automatically passed to the target system if the following conditions are met:

- The user initiates a URL-based process extension from Agile Web Client.

Note	Your Web application must reside in the domain specified in the cookie.domain property of agile.properties. Otherwise, security cookies will not propagate.

- The target system is permitted to receive cookies.
- The target system is in the same domain as the Agile PLM system.

Note	If the target system is located outside the company firewall, it should be a secure Web server using SSL.

Creating an Agile PLM Session from the Target System

Using authentication information contained in the HTTP request received from a URL-based process extension initiated from Agile Web Client, the target application can use the Agile API to create an `IAgileSession`. The Agile API client can then retrieve and configure the Agile PLM object referenced by the HTTP request.

When a user logs into Agile Web Client, the authentication process creates a pair of cookies (`j_username` and `j_password`) on the server computer that store the user's encrypted user name and password.

Note These cookies will expire after the duration set by the Agile PLM Administrator.

When you initiate a URL-based process extension from Agile Web client, the target system can use cookies to create an Agile PLM session. In effect, Agile Web client and the Agile API client on the target system can share a single sign-on.

Note The Agile Java Client, unlike the Web client, does not create client-side cookies. Therefore, it does not support the single sign-on feature for process extensions.

Cookies are designed to be shared among computers within the same domain. For example, if during installation of Agile PLM, you configure the domain to be ".agile.agilesoft.com" and all computers ending with "`.agile.agilesoft.com`" can use the `j_username` and `j_password` cookies.

For more information, see http://wp.netscape.com/newsref/std/cookie_spec.html.

The following example shows how to use the Agile API to extract cookie information from the HTTP servlet request and use that information to generate an `IAgileSession`. The value of the `AgileSessionFactory.PX_REQUEST` field, which is the key used to create the session, is set to be equal to the servlet request.

Example: Creating an IAgileSession from a servlet request using the PX_REQUEST field

```
private IAgileSession connect(HttpServletRequest request) throws
ServletException {
  factory =
AgileSessionFactory.getInstance("http://agileserver/Agile");
  HashMap params = new HashMap();
  params.put(AgileSessionFactory.PX_REQUEST, request);
  session = factory.createSession(params);
  return session;
}
```

If the target application is not servlet-based, there is another way to use the cookie information to create a session. Rather than using `AgileSessionFactory.PX_REQUEST`, you can use `AgileSessionFactory.PX_USERNAME` and `AgileSessionFactory.PX_PASSWORD` fields as keys for the HashMap. The values of these fields should be the values of the j_username and j_password cookies, respectively.

Example: Creating an IAgileSession using PX_USERNAME and PX_PASSWORD fields

```
private IAgileSession connect(Cookie[] cookies) throws Exception {
  factory =
AgileSessionFactory.getInstance("http://agileserver/Agile");
  HashMap params = new HashMap();
  String username = null;
  String pwd = null;
  for (int i = 0; i < cookies.length; i++) {
    if (cookies[i].getName().equals("j_username"))
      username = cookies[i].getValue();
    else if (cookies[i].getName().equals("j_password"))
      pwd = cookies[i].getValue();
  }
```

```
    params.put(AgileSessionFactory.PX_USERNAME, username);
    params.put(AgileSessionFactory.PX_PASSWORD, pwd);
    session = factory.createSession(params);
    return session;
}
```

Retrieving an Agile PLM Object from an HTTP Request

If you invoke a URL-based process extension from an object's Actions menu, you may want the target application to retrieve the Agile PLM object and modify it. The object's composite key and class ID are encoded in the URL using the `GET` method. To make it easier for the target application to retrieve the referenced `IAgileObject`, the Agile API provides an overloaded use of the `IAgileSession.getObject()` method, as shown in the following example. The SDK extracts the object ID information from the request and uses it to retrieve the specified object.

Example: Retrieving an Agile PLM object from an HTTP request

```
private IAgileObject getAgileObject(HttpServletRequest request) throws
ServletException {
    IAgileObject obj = session.getObject(null, request);
    return obj;
}
```

If the target application is not servlet-based, you can use the normal `IAgileSession.getObject()` methods to retrieve the referenced object. For the `params` parameter of `getObject()`, specify a HashMap containing all required attributes for the object's class; the necessary attribute/value pairs are contained in the encoded URL. For a list of identifying attributes for each Agile PLM class, see the following section.

Identifying Attributes for Agile PLM Classes

Each Agile PLM class has a different set of identifying attributes that could be passed as parameters in an encoded URL. For example, a Change object would pass its class ID and Cover Page.Number attribute. The following table lists the identifying attributes for each Agile PLM class.

Class	Parameter	Description
Change	`agile.classID`	Class ID of selected object
	`agile.1047`	Cover Page.Number
Customer	`agile.classID`	Class ID of selected object
	`agile.5110`	General Info.Customer Number
Commodity	`agile.classID`	Class ID of selected object
	`agile.agile.2000 004284`	Title Block.Name
Declaration	`agile.classID`	Class ID of selected object
	`agile.agile.2000 002615`	Title Block.Name
Discussion	`agile.classID`	Class ID of selected object
	`agile.18417`	Cover Page.Number

Class	Parameter	Description
File Folder	agile.classID	Class ID of selected object
	agile.6173	Title Block.Number
	agile.7951	Title Block.Version
Item	agile.classID	Class ID of selected object
	agile.1001	Title Block.Number
	agile.1014	Title Block.Rev
	agile.siteName	Site name — If All is selected, this parameter is omitted
Manufacturer Part	agile.classID	Class ID of selected object
	agile.1647	General Info.Manufacturer Name
	agile.1648	General Info.Manufacturer Part Number
Manufacturer	agile.classID	Class ID of selected object
	agile.1754	General Info.Manufacturer Name
Package	agile.classID	Class ID of selected object
	agile.3110	Cover Page.Package Number
Price	agile.classID	Class ID of selected object
	agile.10355	General Information.Number
	agile.10357	General Information.Rev
Program	agile.classID	Class ID of selected object
	agile.18041	General Info.Number
Sourcing Project	agile.classID	Class ID of selected object
	agile.14824	General Info.Number
PSR	agile.classID	Class ID of selected object
	agile.4856	Cover Page.Number
QCR	agile.classID	Class ID of selected object
	agile.4029	Cover Page.QCR Number
Report1	agile.classID	Class ID of selected object
	agile.8071	General Info.Name
RFQ	agile.classID	Class ID of selected object
	agile.13925	CoverPage.RFQ Number
RFQ Response	agile.classID	Class ID of selected object
	agile.14472	CoverPage.RFQ Number
	agile.14452	CoverPage.SupplierName

Class	Parameter	Description
Site	agile.classID	Class ID of selected object
	agile.11882	General Info.Name
Specification	agile.classID	Class ID of selected object
	agile.2000001969	Title Block.Name
Substances	agile.classID	Class ID of selected object
	agile.2000001124	Title Block.Name
Supplier	agile.classID	Class ID of selected object
	agile.17761	General Info.Number
Transfer Order	agile.classID	Class ID of selected object
	agile.12673	Cover Page.Transfer Order Number
User	agile.classID	Class ID of selected object
	agile.11617	General Info.Username
User Groups	agile.classID	Class ID of selected object
	agile.12077	General Info.Name

Note Although the process extensions framework can encode Report information in a URL, Report objects are not supported by the Agile API. Therefore, you cannot use the Agile API to retrieve Report objects referenced in a URL.

Creating an External Report

In Agile Web Client, you can connect to an external resource or URL to generate an external report. Before you create an external report, you must add the URL associated with the report to the Process Extension Library. For more information, see Defining URL-Based Process Extensions on page 353.

To create reports in Agile Web Client, you must have the Create Reports privilege.

To create an external report:

1. Log in to Agile Web Client.

   Note You cannot create external reports in Agile Java Client.

2. Choose Create > Report > External. The Report Creation Wizard appears.
3. Type the name of the report. Click Next.
4. Type the following General Information:
   - Description – Type a brief description of the report.
   - Process Extension – Select a process extension. The process extension you select is associated with a URL, such as the location of Web-based report engine.

- Folder – Select the report's parent folder.

5. Click Finish.

Deploying Process Extensions in a Clustered Environment

If the Agile PLM installer was not run on a server in the application server cluster, the /agile_home/integration/sdk/extensions folder will not exist on that server. If so, you must create the folder manually and copy any process extension JAR files into that folder.

To manually create deployment folders for process extensions:

1. Create the following folder on all application servers in the cluster (if it does not exist): /agile_home/integration/sdk/extensions
2. Put all process extension JAR files in the /agile_home/integration/sdk/extensions folder on each server in the cluster.

Best Practices for Copying third Party JAR Files

If your PX code uses 3rd party JAR files such as axis.jar, you can copy them in the shared library and add them to classpath using the following procedures.

Use te following procedure to configure your shared libraries

To configure shared libraries in Oracle Application Server:

1. Place all 3rd party JAR files in a folder, for example, D:\commonLib.
2. Stop the Agile server.
3. Navigate to <OAS_HOME>\j2ee\home\application-deployments\Agile.
4. Open orion-application.xml in a text editor.
5. Append a line to specify your library path:

   ```
   <library path="ABSOLUTE_PATH_TO_YOUR_LIBRARY" />
   ```

   <!-- Specify the absolute path to the folder having those required JAR files)

   For example, <library path="D:\commonLib" /> after these two lines:

   ```
   <library path="../APP-INF/lib" />
   <library path="../APP-INF/classes" />
   ```

6. Start the Agile Server.

Procedures in this section show how to deploy the following dependent JAR files:

- The dependent JAR file from 3rd party JAR files that do not dependent on Agile JAR files
- Any dependent JAR file from 3rd party JAR files that are dependent or not dependent on Agile JAR files

To deploy the 3rd party JAR files on WebLogic:

1. Stop all servers in the domain.
2. Copy the shared JAR file(s) to a `lib` subdirectory of the domain directory.

   For example, `cp c:\3rdpartyjars\utility.jar agile_home\agileDomain\lib`

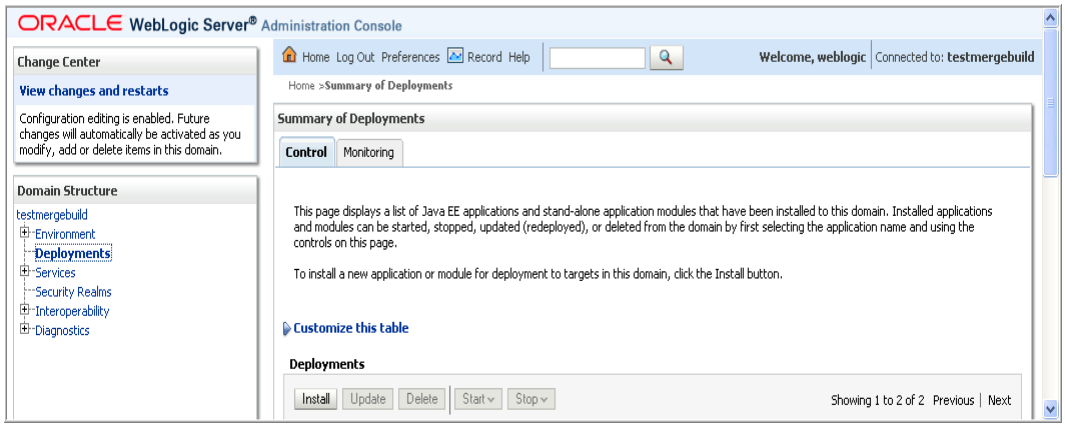
   Note The WebLogic Server must have read access to the `lib` directory during startup.

   Note The Administration Server does not automatically copy files in the `lib` directory to Managed Servers on remote machines. If you have Managed Servers that do not share the same physical domain directory with the Administration Server, you must manually copy the JAR file(s) to the `domain_name/lib` directory on the Managed Servers.

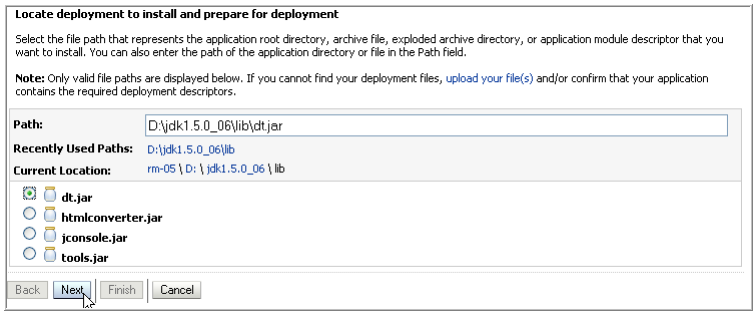
3. Start the Administration and the Managed Servers in the domain.
   Start the Administration Server and all Managed Servers in the domain. WebLogic Server appends JAR files found in the lib directory to the system classpath. Multiple files are added in alphabetical order.

To deploy the dependent JAR files from a WebLogic Admin console:

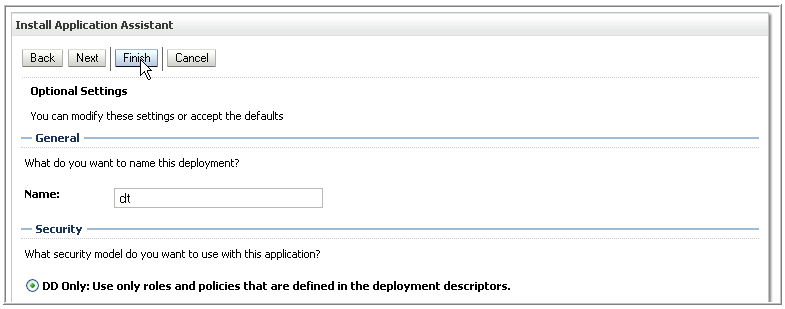
1. In WLS Administration console, select Deployments > Control > Install.



2. Select the JAR file and then click Next.



3. In Install Application Assistant accept the default settings and click Finish.



4. Make sure the library pointd to the Agile Server/cluster.
5. Create a file called `weblogic-application.xml` with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
            <wls:weblogic-application
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:wls=http://www.bea.com/ns/weblogic/weblogic-
            application
            xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogi
            c-application http://www.bea.com/ns/weblogic/weblogic-
            application.xsd http://java.sun.com/xml/ns/j2ee
            http://java.sun.com/xml/ns/j2ee/j2ee_1_4.xsd">

            <wls:library-ref>

              <wls:library-name><Shared Library Jar file
            name></wls:library-name>

   </wls:library-ref>

</wls:weblogic-application>
```

6. Place the `weblogic-application.xml file in agile_home\agiledomain\applications\application.ear\META-INF` folder in the Administration server.
7. Restart the Administration and all the Managed servers.

To deploy the properties files on WebLogic:

1. Stop all servers in the domain.
2. Copy the properties file(s) to a directory as in `agile_home/pxConfig` in each server.

Note	WebLogic Server must have read access to the directory during startup.
Note	The Administration Server does not automatically copy files in the directory to Managed Servers that are on remote machines. If you have Managed Servers that do not share the same physical domain directory as the Administration Server, you must manually copy the properties file(s) to the `agile_home/pxConfig` directory on the Managed Servers.

3. Add the directory containing the properties file(s), for example, `agile_home/pxConfig` to the WebLogic CLASSPATH as shown below:

a. Edit `agile_home/agileDomain/bin/setEnv.bat` in all Managed Servers

b. Add **`agile_home/pxConfig`** folder to the CLASSPATH

```
set CLASSPATH=%JAVA_HOME%/lib/tools.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/ojdbc14.jar
setCLASSPATH=%CLASSPATH%;%WLS_HOME%/server/lib/
   weblogic_sp.jar;%WLS_HOME%/server/lib/weblogic.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/agbase.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/wlsauth.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/crypto.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/xercesImpl.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/jdom.jar;
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/log4j.jar;
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/jobaccess.jar;
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/colt.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/jms.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/jndi.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/tibjms.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/oc4jclient.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/oc4j.jar
set CLASSPATH=%CLASSPATH%;../ldaplib/ldaputil.jar
set CLASSPATH=%CLASSPATH%;D:\Agile931b28/agileDomain/config
set CLASSPATH=%CLASSPATH%;agile_home/pxConfig
```

4. Start the Administration Server and all Managed Servers in the domain.

Process Extensions FAQ

This section answers common questions about process extensions.

What are process extensions?

Process extensions extend the functionality of Agile PLM clients through custom actions, external reports, custom autonumbering and tools, thus tailoring the system to fit a customer's business. Process extensions can be used to connect the Agile PLM server and Agile PLM users to external systems.

What types of actions can you define with process extensions?

Process extensions support two types of process extensions actions. They are custom autonumber sources and custom actions. Custom autonumber sources define the numbering sequences used by classes of objects. Custom actions are programs that can be run from Agile PLM clients.

A process extension can also be a reference to a URL. The URL can be a simple Web site or the location of a Web-based application.

Can Process Extensions support asynchronous operations?

Agile Process Extensions only support synchronous operations. If your Process Extension requires asynchronous behavior, you must modify your PX code to implement asynchronous solutions of your choice. For example, you can spawn a thread.

Can I use Agile's Java API within a process extension program?

Yes. You can use Agile's Java API and other external Java APIs. The only requirement is that you implement either the `ICustomAutoNumber` or the `ICustomAction` interface, depending on the type of extension.

How do you initiate a process extension in an Agile PLM client?

Custom actions can be triggered in the following ways:

- A change to a Workflow status.
- Selecting a custom action from the Tools menu.
- Selecting a custom action from the object's Actions menu.
- Selecting an external report that uses a custom action.
- Creating an object of a class that uses a custom autonumber source.

Do process extensions have special security requirements?

No. The process extensions stack sits on the Agile Application Server, so custom actions and custom autonumber sources operate within an environment where the user has already been authenticated and authorized.

How are roles and privileges defined for custom actions?

By default, a custom action uses the roles and privileges of the current user. However, if you want to configure a custom action to have expanded privileges, you can specify the roles required for a custom action in the Process Extension Library of Agile Java Client. When you use a custom action in the Agile PLM client, roles and privileges that are specified for the custom action, override the roles and privileges of the current user. Once the custom action is completed, the client reverts to the user's original roles and privileges.

How do I configure and deploy a process extension?

Place the JAR file(s) for a process extension in the agile_home/integration/sdk/extensions folder on the application server. Included with the JAR file(s) should be a file named com.agile.px.ICustomAutoNumber or com.agile.px.ICustomAction in the META-INF/services directory. The contents of these files are the fully qualified Java class n*ames, one* class per line, for a custom autonumber source or a custom action, respectively.

After I deploy a process extension program on the application server, how do I enable it?

Once process extensions have been deployed, you can configure them for use within Agile PLM clients. In Agile Java Client, you can add custom actions to the Process Extension Library and custom autonumbers to the Autonumbers table.

After I've deployed JAR file(s) for a custom action or custom autonumber source, do I need to update the application server classpath?

No. The classpath is updated automatically by a special-purpose classloader. The classloader extends the application server classpath with any classes located in agile_home/`integration/sdk/extensions` (or the location specified for the `sdk.extensions` property in the agile.properties file).

How do you create a custom autonumber source?

Create a Java class that implements the `ICustomAutoNumber` interface, a server-side API in the `com.agile.px` package. The code defines the autonumbering logic, for example, prefix, suffix, number of digits, and so on, and the persistence mechanism. The Agile PLM system gets the next number from the custom autonumber source by calling the `getAutoNumber()` method.

How do you assign custom autonumber sources in Agile Java Client?

In the Classes node, you assign autonumber sources to specific subclasses. In the AutoNumbers node, you can also assign subclasses to an autonumber source.

How do you create a custom action?

Create a Java class that implements the `ICustomAction` interface, a server-side API in the `com.agile.px` package. The code defines the custom action, whether to modify the current object, create an external report, integrate the Agile PLM client with an external system, or perform some other business logic. When an Agile PLM client initiates a custom action, the Agile PLM system calls the `doAction()` method.

How do you associate custom actions with the Tools menu, the Actions menu, Workflow statuses, and external reports?

In Agile Java Client, open the Process Extensions node to add and configure custom actions. You can associate custom actions with Workflow statuses, the Tools menu, the Actions menu for classes, and external reports. A custom action associated with a Workflow status is initiated automatically when the Workflow assumes that status. A custom action appears on the Tools menu when its Initiate From property is set to Tools menu. A custom action appears on the Actions menu for an object when you add it to the Process Extensions tab for a subclass. A custom action associated with an external report is triggered automatically when that report is executed.

In what order do process extensions appear on the Tools menu or Actions menu of Agile PLM clients?

If you add process extensions to either the Tools menu or an object's Actions menu, they are listed after standard menu commands in the order they were created. You cannot reorder or otherwise manage commands on the Tools menu or Actions menu.

What is the inheritance model of custom actions that are assigned to classes?

Custom actions can be defined at the base class level, the class level, or the subclass level. A custom action defined at the base class level is available to all classes and subclasses beneath the base class. A custom action defined at the subclass level is available only to that subclass.

Where do I put PX and WSX configuration property files?

After deployment changes in Agile PLM Release 9.2.2.2, the agileDomain\config directory is no longer in the classpath. You can put PX and WSX property files in this directory: \oas\j2ee\home\applications\Agile\APP-INF\classes\.

Chapter 22

Developing Web Service Extensions

This chapter includes the following:

- About Web Service Extensions 367
- About Web Services 369
- Developing and Deploying a Web Service 372
- Using a Web Service 373
- Authenticating Users 374
- Using Single Sign-On Cookies for Client-Server Access 374
- Preparing Environment for MyFirstWebService 376
- Building MyFirstWebService Sample 378
- About Web Service Clients 379
- Creating MyFirstClient 380
- Microsoft .NET Interoperability 382
- Web Service Extensions FAQs 383

About Web Service Extensions

Web service extensions (WSX) is a Web service engine enabling communication between Agile PLM and disparate systems both internal and external including Enterprise Resource Planning (ERP) systems, Customer Resource Management (CRM) systems, Business-to-Business Integration systems (B2Bi), other Agile PLM systems, and supply chain partners. WSX can streamline the process for new product introduction (NPI), product changes, and rapid ramp-up of manufacturing resources. It can also simplify the process for aggregating raw product content and making critical product content available in real time to other core systems. WSX contains the tools and framework to develop new Agile PLM Web services.

You can use WSX to:

- Make product content available to Enterprise Application Integration (EAI) systems, which can then feed the data to a broad array of internal applications.
- Share product content with product design, manufacturing planning, shop floor, Enterprise Resource Planning (ERP), and Customer Relationship Management (CRM) applications.
- Make product content available to Business-to-Business (B2B) systems, which can transfer Agile Application server data across corporate boundaries to a wide range of external applications.
- Provide content to exchanges, reports, and custom applications and import Product content data from ERP and other supply chain applications.

Note Agile Integration Services (AIS) is a set of Web services that is built with WSX technology to provide programmatic import and export capabilities for the Agile PLM system. AIS is a separately licensed product. For more information about AIS, refer to the *Agile Integration Services Developer Guide*.

Key Features

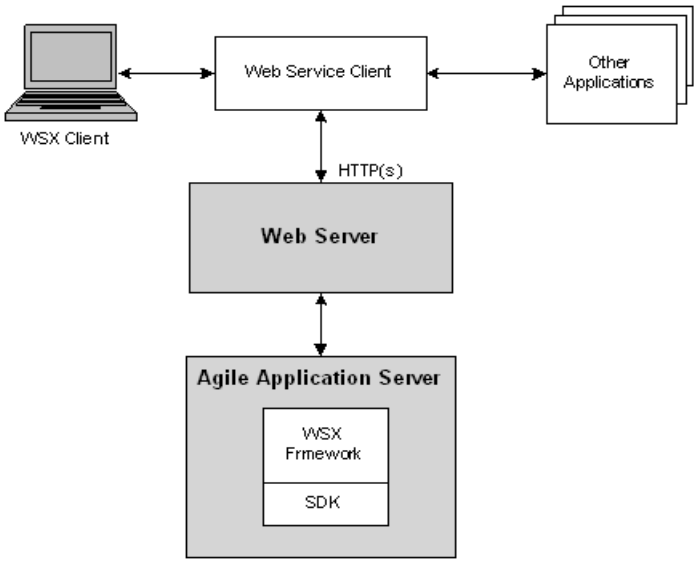
WSX includes the following key features:

- Programmatic access to data – WSX provides programmatic access to data stored in Agile PLM systems and other data resources, allowing you to create custom applications to automate content transfer.
- Accessibility – WSX provides accessibility of Agile PLM product content outside the corporate firewall using standard HTTP(S) technology.
- Multiple programming language support – WSX supports any language that can create and understand Simple Object Access Protocol (SOAP) and/or Web Services Description Language (WSDL).
- Multiple output format support – WSX supports aXML and PDX 1.0. You can also use XSL to transform XML data into any format, or develop Web services that return data in any format.
- Security – WSX communicates with XML-compliant applications using Internet-standard communication and security protocols (HTTP and SSL), so the interface is both firewall-friendly and secure.

WSX Architecture

To connect to Agile PLM and the WSX framework, you use standard Web service invocation methodologies.

Figure 23: WSX architecture



About Web Services

Web services is a technology for building distributed applications. These services, which can be made available over the Internet, use a standardized XML messaging system and are not tied to any one operating system or programming language. Through Web services, companies can encapsulate existing business processes, publish them as services, search for and subscribe to other services, and exchange information throughout and beyond the enterprise. Web services are based on universally agreed upon specifications for structured data exchange, messaging, discovery of services, interface description, and business process design.

A Web service makes remote procedure calls across the Internet. It uses HTTP(S) or other protocols to transport requests and responses and the Simple Object Access Protocol (SOAP) to communicate request and response information.

The key benefits provided by Web services are:

- Service-oriented Architecture – Unlike packaged products, Web services can be delivered as streams of services that allow access from any platform. Components can be isolated; only the business-level services need be exposed.
- Interoperability – Web services ensure complete interoperability between systems.
- Integration – Web services facilitate flexible integration solutions, particularly if you are connecting applications on different platforms or written in different languages.
- Modularity – Web services offer a modular approach to programming. Each business function in an application can be exposed as a separate Web service. Smaller modules reduce errors and result in more reusable components.
- Accessibility – Business services can be completely decentralized. They can be distributed over the Internet and accessed by a wide variety of communications devices.
- Efficiency – Web services constructed from applications meant for internal use can be used for externally without changing code. Incremental development using Web services is relatively simple because Web services are declared and implemented in a human readable format.

Like any technology, Web services have some limitations. When developing Web services, you should consider the following:

- SOAP is a simple mechanism for handling data and requests over a transport medium. It is not designed to handle advanced operations such as distributed garbage collection, object activation, or call by reference.
- Because Web services are network-based, they are affected by network traffic. The latency for any Web service invocation can often be measured in hundreds of milliseconds. Thus, the amount of functionality provided by the service should be significant enough to warrant making a high-latency call.
- Web services are not good at conversational programming. Thus, when designing services to be exposed, you should try to make the service as independent as possible.

Web Services Architecture

You can view Web services architecture in terms of roles and the protocol stack:

- Web service roles:
  - Service provider – This provides the service by implementing it and making it available on the Internet.
  - Service requestor – This is the user of the service who accesses the service by opening a network connection and sending an XML request.
  - Service registry –This is a centralized directory of services where developers can publish new services of find existing ones.
- Web services protocol stack:
  - Service transport layer – uses HTTP to transport messages between applications. Other transports will be supported in future AIS releases.
  - XML messaging layer – encodes messages in XML format by using SOAP, a platform-independent XML protocol used for exchanging information between computers. It defines an envelope specification for encapsulated data being transferred, the data encoding rules, and RPC conventions.
  - Service description layer –describes the public interface to a specific Web service by using the Web Service Description Language (WSDL) protocol. WSDL defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages, which contain either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a network protocol and message format. WSDL allows description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A WSDL document defines services as collections of network endpoints (called ports). A port is defined by associating a network address with a reu*sabl*e binding, and a collection of ports define a service.
  - Service discovery layer – centralizes services into a common registry by using the Universal Description, Discovery, and Integration (UDDI) protocol.

Note WSX does not currently support UDDI or other service discovery layers.

Security

WSX communicates with XML-compliant applications using Internet-standard communication and security protocols (HTTP and SSL). Communication between WSX and its clients (via the Web server) may be encrypted via Secure Sockets Layer (SSL) and a server-side certificate, thus providing authentication, privacy, and message integrity. Using standard Java cryptography libraries, you can encrypt and decrypt files, create security keys, digitally sign a file, and verify a digital signature.

The Web service extensions framework forces any invocation request received from outside the firewall to be secure. In other words, all external requests to WSX must be secured using HTTPS or an equivalent protocol. Internal requests to WSX can be conducted insecurely, that is, using HTTP.

There are several ways to enforce username and password security when invoking a Web service. If you are using the Agile API to develop your Web service, you can specify the username and

password in the `createSession()` parameters just as you would with any API program.

For more information about Java security and cryptography support, see http://java.sun.com/j2se/1.3/docs/guide/security/index.html

Tools

There is no single set of tools needed to access Web services. The tools you choose depend very much on the environment you use to develop clients. Basically, you'll need tools that enable you to generate and process XML, and process HTTP request/responses messages.

The WSX framework is based on the Apache eXtensible Interaction System (AXIS), which is a SOAP processor. However, you can use other implementations of SOAP tools, regardless of source language, to build Web service clients.

Note The WSX Java samples included with the Agile SDK show how to use AXIS. For detailed information about AXIS, its features, and how to use it, refer to the AXIS Web site: http://xml.apache.org/axis

Finding Additional Information About Web Services

This is a list of some Web sites to explore:

- WebServices.Org – http://www.webservices.org/
- Web Services Architect – http://www.webservicesarchitect.com/
- Web Services Journal – http://www.sys-con.com/webservices/
- webservices.xml.com – http://webservices.xml.com/
- O'Reilly Web Services – http://webservices.oreilly.com/
- Apache Axis – http://ws.apache.org/axis/
- Java Web Services Developer Pack 1.1 – http://java.sun.com/webservices/webservicespack/html
- Sun ONE Web Services Platform Developer Edition – http://forums.sun.com/thread.jspa?threadID=5020700
- Microsoft .Net Framework – http://msdn.microsoft.com/netframework/
- SOAP::Lite for Perl – http://www.soaplite.com/
- Soap Tutorial – http://www.w3schools.com/soap/default.asp

Developing and Deploying a Web Service

Writing your own Web service is a simple task, consisting of a few steps:

1. Define your Web service's entry point(s). A Web service entry point (or operation) corresponds to a public method on a Java class.
2. Code your Web service operation's logic. You need not follow any special rules when coding the logic for your Web service operation. You may take advantage of third party code libraries as well as Agile-provided libraries, including the Agile API.
3. Compile your Java code as you normally would.
4. Copy the compiled JAR file(s) to *AGILE_HOME*\integration\sdk\extensions on the Agile Application Server computer. The deployment descriptor for the Web service should also be in the JAR file(s) in a file named META-INF/services/com.agile.wsx.Deployment.wsdd.

   Note If you have several application servers in a clustered environment, you must deploy Web service files on each server in the cluster.

The Agile Application Server automatically deploys all Web services listed in the deployment descriptor, ensuring that your latest changes have been applied.

About Deployment Descriptors

The Web service deployment descriptor file (Deployment.wsdd) is an XML file that is formatted according to Axis's Web Service Deployment Descriptor (WSDD) format. It declares and describes the set of Web services and Web service operations that are to be exposed via WSX. The WSDD file also defines any additional behavior that should be used when processing incoming SOAP requests (such as authentication, and so on) or responses (such as reformatting outgoing data).

The Axis documentation provides a good introduction to the WSDD format and its use. However, before consulting the Axis documentation, please be aware of the following constraints within WSX:

- The Web service deployment descriptor should not contain global WSX configuration information. The configuration information declared within Deployment.wsdd should be restricted to service-specific declarations.
- WSX does not support the Axis .jws-based Web services. While these sound good on paper, we have found our mechanism of redeploying Web services to be more robust and easier to work with in a development environment.
- For security reasons, WSX does not include the Axis AdminServlet.

For more information about Axis deployment descriptors, refer to the following Axis documentations:

- *Axis User's Guide* – http://ws.apache.org/axis/java/user-guide.html

  See the sections entitled "Custom Deployment - Introducing WSDD" and "Service Styles - RPC, Document, Wrapped, and Message."

- *Axis Reference Guide* – http://ws.apache.org/axis/java/reference.html

  See the sections entitled "Deployment (WSDD) Reference."

Note These sites are subject to periodic change. In that event, use your favorite search engine to locate these documents.

Reserved Web Service Names

The following Web service names are reserved because they are used by Agile Integration Services (AIS). Do not use them to name a Web service that you've created.

- Export
- Importer
- Reserved Service names:
  - FSHelper, DmsService (File manager and Viewer)
  - Export, Importer (AIS)
  - ResponseService, PackageService, AcsStatusService (ACS)

Using a Web Service

Once you have developed and deployed your custom Web service, you will want to use it. You can access your Web service using a URL of the form
`http://<hostname>:<port>/<virtualPath>/integration/ws/`<WebServiceName>

Note You must use the Agile-modified `axis.jar` file that is included with the Agile API. This file
gets installed in the following location when you install Agile's API component:
agile_home`\integration\sdk\lib\axis.jar`

Defining a Web Service Entry Point

A Web service entry point (or operation) corresponds to a public method on a Java class. Not all public methods on a class need be exposed as an operation, but all operations correspond to public methods. Thus, if you have a Java class (such as `MyClass`), that exposes two public methods (such as `methodOne` and `methodTwo`), it is possible for you to expose either or both methods as Web service operations.

As a general rule, the simpler the datatypes used for your parameter and return types, the more interoperable your Web service operation will be. More complex datatypes will require either custom serializers/deserializers or additional support from the Web service framework. More information on the additional serializers/deserializers provided by Axis can be found at http://ws.apache.org/axis/java/apiDocs/org/apache/axis/encoding/Serializer.html and http://ws.apache.org/axis/java/apiDocs/org/apache/axis/encoding/Deserializer.html. These sites are periodically changed. In this case, pleas invoke your favorite search engine to locate the latest information on these interfaces.

Note As a rule, do not try to return an Agile API object, such as `IAgileSession` or `IItem`, from a Web service. Web services should only return data structures.

Authenticating Users

All default out-of-box Web services and user customized versions are protected by the application server. To access a protected Web service, add the following lines in your Web service client stub code:

```
// Configure the stub with the necessary authentication information
    stub.setUsername(cl.getOptionValue(USER_SHRT));
    stub.setPassword(cl.getOptionValue(PASSWORD_SHRT));
    stub.setMaintainSession(true);
```

To remove the Web container protection for a specific Web service, add the following lines in the following applications:

```
application.ear#application.war/WEB-INF/web.xml
```

and

```
application.ear#integration.war/WEB-INF/web.xml files:
<security-constraint>
    <web-resource-collection>
    <web-resource-name>Unprotect web services</web-resource-name>
    <url-pattern>/ws/<web service name></url-pattern>
    <url-pattern>/services/<web service name></url-pattern>
    </web-resource-collection>
</security-constraint>
```

Using Single Sign-On Cookies for Client-Server Access

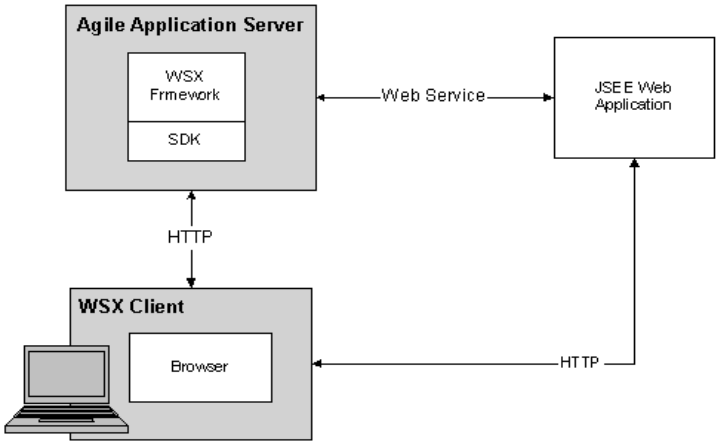
After a user on the WSX client is authenticated by the Agile 9.X server which is protected by third party single sign-on products, the browser is granted a Single sign-on cookie. This cookie is sent to the custom j2ee Web application, provided this application is in the same DNS domain as the Agile 9.X server. Now, to invoke the Web service deployed on Agile 9.X server, you can pass the single sign-on cookie instead of username and password as a valid credential.

Note	If you are using both username/password and single sign-on cookies, the single sign-on cookie has precedence over username/password.

Deployment Architecture

Interactions and the request flow between the Agile server and WSX client is summarized in the following illustration.

Figure 24: Deployment Architecture



Invoking the Web Service Client with a Single Sign-on Cookie

This is accomplished by first, retrieving the single sign-on cookie from the HTTP request followed by modifying the SOAP binding stub code.

Retrieve the Single Sign-On Cookie

Before invoking the Web service client stub, you must retrieve the single sign-on cookie in the HTTP request. By default, the single sign-on cookie provided by SiteMinder is called "SMSESSION." Modify the cookie to the format specified in RFC2965 available at http://www.ietf.org/rfc/rfc2965.txt. The simplest format is "name=value" where name and value are accessed by calling the `javax.servlet.http.Cookie object` method.

Modifying the SOAP Binding Stub Code

Find the Web service SOAP binding stub class which is generated by wsdl2java utility of axis. It is usually called `<service-name>SoapBindingStub.java`. Add a variable named *cookies* and a method to set the value as shown below.

To modify the SOAP binding stub code:

1. Add the following lines in the SOAP stub class:

```
private String cookies = "";
public void setCookies(String cookies) {
      this.cookies = cookies;
}
```

2. Add the line in bold font in `createCall()` method.

```
if (super.cachedPortName != null) {
    _call.setPortname(super.cachedPortName);
}
_call.setProperty(org.apache.axis.transport.http.HTTPConstants.HEADER_
COOKIE, this.cookies);
java.util.Enumeration keys = super.cachedProperties.keys();
```

3. Recompile this class and follow the sample below to invoke the Web service stub.

```
((<soaping binding stub class name>)stub).setCookies(<sso cookies you
got in step 2.1>);
stub.setMaintainSession(true);
```

4. Compare with the documented sample that requires username and password as valid credentials.

```
stub.setUsername(<username>);
stub.setPassword(<password>);
stub.setMaintainSession(true);
```

5. Test the Web service client as part of the j2ee Web application.

Preparing Environment for MyFirstWebService

To explain developing a Web service, a sample that highlights the development process is provided. The sample, called MyFirstWebService, is a simple example that demonstrates how to create a Web service that can use the Agile SDK to retrieve information about a particular Item and return the Item as the result of the Web service operation.

To support the desired operation, the following entry point is defined:

```
public String getItemField(String[] args) throws RemoteException
```

MyFirstWebService sample uses a third party library called Jakarta Commons CLI and parses `args` as a set of command line arguments. Based on these arguments, the results are returned as a `String`. You can find more information on implementation details in the SDK_samples.zip folder described in Client-Side Components on page 2. The path to MyFirstWebService is in `samples\wsx\src\first`. The remaining paragraphs in this section describe the deployment process and do not address implementation details.

Downloading Tools to Build the Sample

Before you can build and deploy the `MyFirstWebService` sample, you need to download the following tools:

Tool	Download Site
Java 2 SDK SE Version 1.5	http://java.sun.com/j2se/1.4.2/download.html
Apache Project's Ant build tool, version 1.6.5	http://ant.apache.org/

Installing the Java SDK

This section provides the instructions to install the Java SDK on Windows and on Solaris platforms. You can skip this section if you already have the proper version of Java installed.

To install the Java SDK on Windows:

1. Double-click the distribution and follow the installation procedures.
2. Set the system variable JAVA_HOME to point to the home directory of your Java SDK installation (for example, D:\j2sdk150).

To install the Java SDK on Solaris:

1. Execute the distribution (for example, $ ./ j2sdk-1_5_0-solaris-sparc.sh) and follow the installation procedures.
2. Set the environment variable JAVA_HOME to point to the home directory of your Java SDK installation (for example, /home/<user>/j2sdk150).
3. Execute your .profile or .cshrc (depending on your shell) file to reinitialize your environment settings.

Installing Ant

This section provides the necessary instructions to install Ant on Windows and Solaris machines.

To install the Ant on Windows:

1. Extract the contents of the Zip archive to a local directory and follow the installation procedures.

   The Ant distribution for Windows is a zip file (for example, apache-ant-1.6.5-bin.zip).
2. Open a command prompt window and verify that Ant can be invoked by entering the following command:

   `%ANT_HOME%\bin\ant -version`

   The following output is displayed:

   ```
   Apache Ant version 1.6.5 compiled on date
   ```

To install the Ant on Solaris:

1. Extract the contents of the tar archive to a local directory (for example, /home/*user*/ant) and follow the installation procedures.

   The ANT distribution for UNIX is a tar file (for example, apache-ant-1.6.5-bin.tar.gz).

2. Execute your .profile or .cshrc (depending on your shell) file to reinitialize your environment settings.

3. From a command prompt, verify that Ant can be invoked by entering the following command:

   `$ANT_HOME/bin/ant -version`

   The following output should be displayed:

   `Apache Ant version 1.6.5 compiled on date`

Building MyFirstWebService Sample

Agile provides several sample programs for the SDK, including a sample Web service program called MyFirstWebService mentioned in . To download the sample program, refer to the Note in Client-Side Components on page 2. The MyFirstWebService sample is in the *wsx* folder in SDK Samples.

The Ant tool reads the build.xml script and builds all targets in the following sequence on the server that is running the WSX samples):

1. Compiles the Java code for the Web service into `MyFirstWebService.jar`.
2. Copies the resulting `MyFirstWebService.jar` file, which includes the Deployment.wsdd file, and the
   commons-cli.jar file into the .../sdk/extensions folder.
3. Generates a script (either runner.bat or runner.sh) that can be used to run the client. (It conveniently sets the CLASSPATH needed to run the client.)
4. Generates client-side stub files and places them in the following folder:

   sdk\samples\wsx\built\src\client located in Oracle Agile PLM 9.3 Event and Web Services Samples. To access this folder, see the Note in Client-Side Components on page 2.

5. Compiles the client classes and places them in the sdk\samples\wsx\built\classes\client flder, located in Oracle Agile PLM's Event and Web Services Samples Web site at: https://codesamples.samplecode.oracle.com/servlets/tracking/id/S614.

To build the WSX sample on the server platform:

1. Copy the SDK_samlpes (ZIP) file. For information to access this file, see the Note in Client-Side Components on page 2.
2. Go to samples/WSX folder.

   Note If there's no `AgileAPI.jar` in this folder, you are not able to compile the WSX sample. In that case, do as follows:

3. Go to $AGILE_HOME/sdk/samples/wsx on the server.
4. Download `wsdl4j-1.5.1.jar` from

http://archive.apache.org/dist/ws/axis/1_2/ (axis-bin-1_2.zip#/lib), and copy it to lib folder and rename the file to `wsdl4j.jar`.

5. Build the MyFirstWebService sample using the sample's `build.xml` file:
   - On Windows – `%ANT_HOME%/bin/ant`
   - On Solaris/Linux – `$ANT_HOME/bin/ant`

   Important: If you are not building the Web service sample under $AGILE_HOME/sdk/samples/wsx, then upload the `wsx/built/MyFirstWebService.jar` into $AGILE_HOME/integration/sdk/extensions. This directory is configurable in agile.properties on the server. Because the SDK will not generate WSDL files or WSXs when you invoke http://hostname:port/virtualPath/services/MyFirstWebService?wsdl, it will not return the required WSDL file. To generate these files do as shown in the next step.

6. Copy the package `wsdl4j.jar` file described above into Agile `application.ear#APP-INF/lib` folder and redeploy the ear file.
7. In the WSX folder, invoke the applicable command below to generate the WSX stub.
   - On Windows –
     ```
     %ANT_HOME%/bin/ant -Dwsx.url=http://webserver/virtualPath/services -Dusername=<username> -Dpassword=<password>
     ```
   - On Solaris/Linux –
     ```
     $ANT_HOME/bin/ant -Dwsx.url=http://webserver/virtualPath/services -Dusername=<username> -Dpassword=<password>
     ```

About Web Service Clients

This section describes the tools that you can use to develop client applications and languages that can generate and process XML files and HTTP request/response messages.

Client Programming Languages

Although Agile tests and certifies Java for use in developing AIS clients, SOAP messages are platform- and language-independent, which means you can use virtually any client programming language that can generate and process XML and process HTTP request/response messages. For example, you can develop clients in Java, Visual Basic.Net, C++, C, or Perl.

There are helpful libraries for Java, .Net, Perl, Python, C++, and C, and for other environments as well. Here are some Web sites where you can find more information:

- Apache Axis – Open source SOAP implementation for Java: http://ws.apache.org/axis/
- Java Web Services Developer Pack (JWSDP) – Sun's Java implementation of the SOAP protocol: http://java.sun.com/webservices/webservicespack.html
- Microsoft .Net – An XML Web services platform for Microsoft Windows that can be used to create Web service clients: http://msdn.microsoft.com/net
- SOAP::Lite for Perl – A Perl implementation of the SOAP protocol: http://www.soaplite.com/

Note For a comprehensive list of other SOAP implementations, see the following Web site: http://www.soapware.org/

Accessing a Web Service

In general, to access a Web service, you need to do the following:

1. Generate a SOAP request – In many cases, a Web-service-aware code library will be able to generate client-side stubs that generate an appropriately formatted SOAP request.
2. Submit that request to WSX via HTTP or HTTPS – Once an appropriate set of client-side stubs has been generated, a client application can use these stubs to submit a request.
3. Process the SOAP response – The client-side stubs usually are responsible for processing the SOAP response and converting the response into an appropriate set of return objects.

The WSX samples provide examples of how SOAP and Web service-related libraries can make this process simple. The following sections illustrate, using the MyFirstWebService sample, the above steps in greater detail.

Creating MyFirstClient

When you build and deploy MyFirstWebService, you also automatically generate the client-side stubs and the client classes. This section uses MyFirstClient as an example to describe some general aspects of how to create a Web service client.

Generating the SOAP Request

In most cases, generating an appropriate SOAP request is as simple as making use of client-side stubs. Many Web-service-aware code libraries are able to generate client-side stubs for you. This entails using a code generation utility along with the WSDL for the desired Web service.

Axis provides a WSDL2Java utility that can be used to generate client-side stubs. Other Web-service-aware libraries have their own client-side stub generation facility. Microsoft .Net includes a wsdl.exe utility. In the case of the WSX samples, the client-side stub generation occurs during the samples' build process.

Within the build.xml file, you will find the following Ant target:

```
<target name="generate-stubs" depends="init" unless="stubs.present">
   <fail unless="wsx.url">wsx.url must be defined</fail>
   <axis-wsdl2java output="${built.dir}/src"
      url="${wsx.url}/MyFirstWebService?wsdl">
   <mapping namespace="http://www.agile.com/ws/SampleWsx"
      package="client"/>
   </axis-wsdl2java>
</target>
```

This Ant target is responsible for generating the client-side stubs for MyFirstWebService. This invocation retrieves the MyFirstWebService WSDL from ${ws.url}/MyFirstWebService?wsdl, generates Java code in the client Java package, and places the source code within the ${built.dir}/src directory. For more information on the WSDL2Java utility, please consult the Axis

documentation, which can be found on the Axis Web site at http://xml.apache.org/axis.

Once the client-side stubs have been generated, the user can use the generated object definitions in order to more easily generate the appropriate SOAP request. Rather than requiring the user to understand how to construct a valid SOAP request, these stubs allow the user to focus on the capabilities of the target Web service operation. Looking at the MyFirstClient.java sample found within ..\samples\wsx\src\client, note that the main method contains all the code used to generate the SOAP request.

Submitting the SOAP Request

The next step in consuming a Web service operation is properly submitting the generated SOAP request to the Web service engine. When dealing with generated client-side stubs, this step is usually as simple as pointing the stubs to the desired server and invoking a method on the stubs. You do not need to worry about opening a connection or manually marshaling your data onto the wire. Instead, the generated stubs handle these details for you.

The MyFirstClient.java sample found within ..\samples\wsx\src\client illustrates how to submit the SOAP request in two places:

- The `getStub()` method is responsible for pointing the client-side stubs to the desired Web service engine.
- The stub.`getItemField()` method invocation found within the main method is responsible for submitting the request to the Web service engine. The submitting of the request is managed by the stubs themselves; you do not need to worry about the connecting, submitting, or marshaling particulars.

The details on how you point the stubs to the desired Web service engine and submit the request vary from code library to code library. Please consult the documentation for your Web-service-aware code library for more information.

Processing the SOAP Response

The processing of the SOAP response is usually handled via the generated client-side stubs. Without these generated stubs, you would be responsible for parsing the XML-based SOAP response and dealing with the many formatting and unmarshaling issues that arise. However, when dealing with generated stubs, all of these details are taken care of for you, allowing you to receive properly typed Java objects. Rather than require you to parse an XML document and discern what the returned data is, the stubs automatically do this for you.

The details on how SOAP responses are processed will vary from code library to code library. Some SOAP servers expect the client to know the datatype through some other means (perhaps WSDL). Please consult the documentation for your Web-service-aware code library for more information.

Running MyFirstClient

To build and deploy the MyFirstWebService sample, use the information provided in the in file SDK_samples.zip folder that lists the necessary CLASSPATH initializations to run the sample on the Web service client. To access this folder, see the Note in Client-Side Components on page 2.

- In Windows environments, the file is called `runner.bat`
- In environments, the file is called `runner.sh`

To print out a usage statement for MyFirstClient, enter the following command:

```
> runner client.MyFirstClient
```

The following usage statement returns the “Title Block.Description” field for part 1000-02:

```
> runner client.MyFirstClient -T 15000 -a "<attribute name>"
    -e <virtual path> -h <host> -l <port> -n <item number> -p <password>
    -u <username>
> runner client.MyFirstClient -T 15000 -a "Title Block.Description"
    -e Agile -h localhost -l 80 -n 1000-02 -p agile -u jeffp
```

Creating an Agile Session inside WSX

By default, the Web container protects the WSX. Therefore, you must specify user credentials when creating an Agile Session inside the WSX. The following example creates an Agile session within a protected WSX.

Example: Setting up a session inside a WSX

```
AgileSessionFactory factory = AgileSessionFactory.getInstance(null);
IAgileSession session = factory.createSession(null);
```

Note Do not override the implicit session.

To have a different user, you need to make an explicit SDK session as if connecting from a remote client. That is, provide an argument to the AgileSessionFactory.getInstance() method.

Example: Creating an explicit session independent of the implicit session

```
AgileSessionFactory factory = AgileSessionFactory.getInstance
("http://...");
HashMap params = new HashMap();
    params.put(AgileSessionFactory.USERNAME, ...);
    params.put(AgileSessionFactory.PASSWORD, ...);
    IAgileSession session = factory.createSession(params);
```

Microsoft .NET Interoperability

Microsoft’s .NET framework technology is a development framework that provides an application programming interface (API) to the services and APIs of classic Windows operating systems, while bringing together a number of disparate technologies that emerged from Microsoft in the late 1990s: ASP, COM+, XML, SOAP, to name a few.

.NET also brings together all the languages provided by the Visual Studio environments provided by

Microsoft such as Visual Basic, J++, and C++. Also, new languages have been developed - such as C# (read C Sharp) and the relatively new language to the .NET family, J# (read J Sharp). J# is actually Java in Microsoft disguise providing integration of Java into the .NET framework. Yet, J# will not work with the Java VM. J#, in essence, acts as a wrapper to contain Java-enabled code to be executed by the .NET Common Language Runtime (CLR), Microsoft's own 'virtual machine'.

The CLR is probably the most important component to the .NET framework. The CLR provides for the activation of objects, security checks, memory management, object execution, and memory cleanup (garbage-collection) when objects are no longer being used.

Another factor behind .NET is that it not only provides for the writing of Windows-based applications or Web-based applications (via ASP.NET) by using any of the languages mentioned, it also can integrate these languages into one common API. This means that developers can write language independent code, inherit from classes, catch exceptions, and take full advantage of polymorphism across differing languages across the .NET framework.

Important	Although the WSX framework (the AXIS SOAP processor) works fine with AXIS Web service clients, it is not completely compatible with .Net. Neither Microsoft nor the Apache group have conducted interoperability tests for AXIS and .Net. For simple data types, AXIS-based Web services should work fine with .Net-based Web service clients. For some complex data types (such as binary attachments), you may experience interoperability problems. For interoperability information about non-AXIS Web service implementations deployed outside of the Agile Application Server, contact the specific Web service vendor.

Web Service Extensions FAQs

This section answers common questions about Web service extensions.

What is Web service extensions (WSX)?

WSX is a framework for Agile customers to extend the functionality of the Agile PLM server using Web services.

What are Web services?

Web services use the SOAP messaging protocol to provide software services over the Internet, allowing software components to interact with each other around the world. Web services are not tied to any one operating system or programming language. They use WSDL to describe a service's public interface, essentially making Web services self-describing and therefore relatively easy to use.

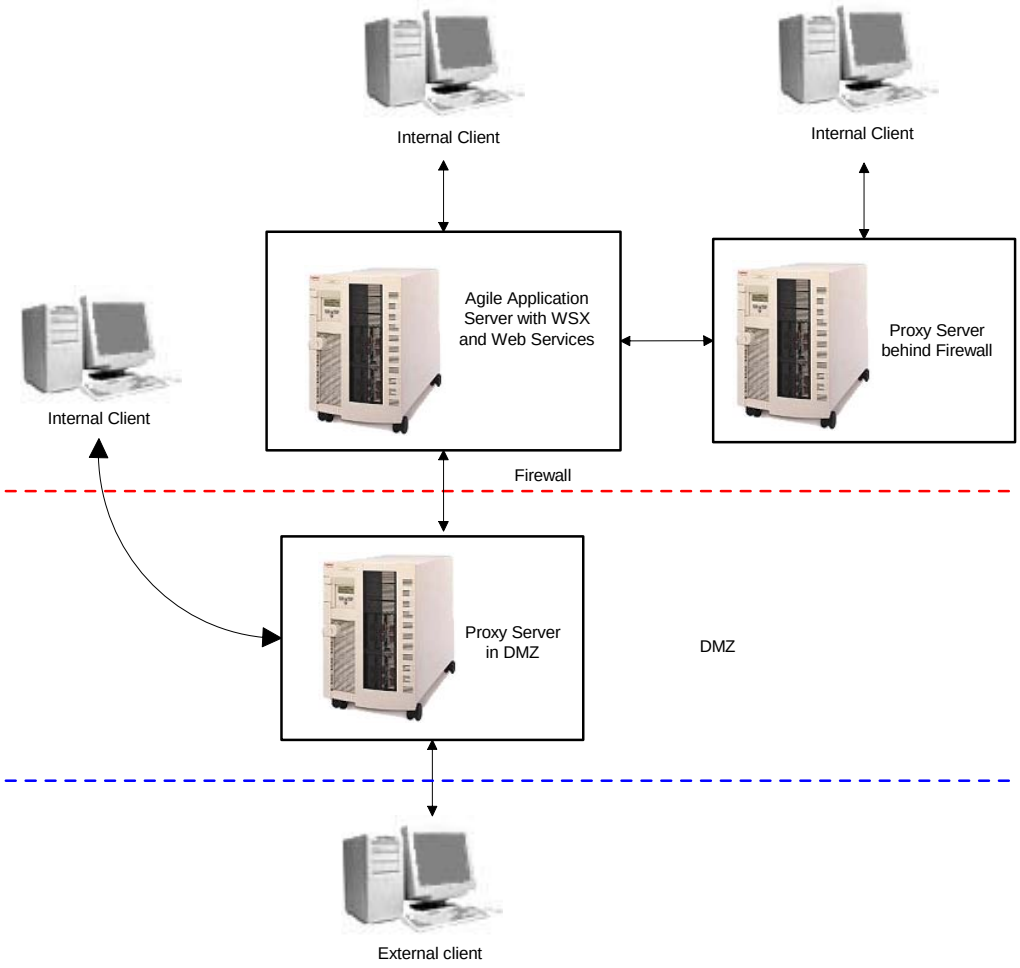
What can I do with WSX that I can't do solely with Agile's Java API?

WSX provides firewall-friendly, XML-based integration with Agile PLM data using the standard HTTP(S) protocol. It supports any SOAP-compliant programming language. For example, you can create Perl or .Net clients for a Web service. WSX enables systems in different companies to interact with each other easily and securely. Services deployed within WSX take advantage of all the scalability, failover, and clustering features provided by the application server. There are also compelling performance benefits to services that run on the application server.

Does WSX support both secured and unsecured connections?

Yes. Requests that come to a Web service from outside the firewall are subject to different security requirements from requests that originate within the firewall. Two separate entry points are provided for each WSX, external (outside the firewall) or internal. External requests are made against a proxy server and then forwarded to the application server. The proxy server resides in the DMZ. Internal requests can be made against the same secure proxy server, another proxy server that doesn't reside in the DMZ, or directly against the application server, as shown in the following figure.

Figure 25: How Web service clients connect to the Agile PLM server



What user authentication services are provided by WSX?

By default, WSX is protected by application server. Username and password security is enforced whenever a WSX client invokes a service that is protected. For more information, see Authenticating Users on page 374.

What SOAP engine does WSX use?

WSX is based on Apache Axis, an open-source implementation of SOAP. For more information about Axis, refer to the Axis Web site at http://ws.apache.org/axis/.

Does WSX handle SOAP attachments?

Yes. In fact, Agile Integration Services provides exportData and importData services that let you export and import binary attachment files.

Does WSX support stateful sessions?

Yes. The Axis Web services engine at the heart of WSX maintains session state between connections. Sessions can be based on HTTP cookies or on SOAP headers. This is useful for generating server-side code that supports more persistent applications instead of simple, one-shot processes. For more information about Web services sessions, refer to Axis documentation. You can start with the Axis FAQ located at http://ws.apache.org/axis/faq.html.

Does WSX support protocols other than HTTP?

No. WSX supports only HTTP-related protocols. For additional security, you can connect to a Web service using HTTPS and SSL. Over time, WSX may support additional protocols as needed.

Does WSX support Perl, Python, PHP, or other Web scripting languages?

WSX supports any client programming language that can send a SOAP message. Although the Agile SDK does not provide WSX client examples in Perl, Python, or PHP, those scripting languages are certainly capable of sending SOAP messages.

Does WSX support UDDI?

No. UDDI is a specification for a universal business registry of Web services that's designed to enable software to automatically discover and integrate with other services. It's currently unnecessary to register Agile PLM Web services on the Internet using UDDI. Agile PLM Web services are typically created for integration with internal software systems or to exchange data with partners or suppliers. However, Agile may consider supporting UDDI as the technology matures.

How do I deploy a Web service?

Place the service's JAR files in the agile_home/integration/sdk/extensions folder on the application server computer. Included with the Web service's JAR file(s) s*hould be a* deployment descriptor file named META-INF/services/com.agile.wsx.Deployment.wsdd.

The deployment descriptor file is an XML file formatted according to Axis' Web Service Deployment Descriptor (WSDD) format. It declares and describes the set of Web services and Web service operations that are exposed via WSX. The WSDD file also defines any additional behavior that should be used when processing incoming SOAP requests (such as user authentication) or responses (such as reformatting outgoing data). For more information about WSDD format, refer to the *Axis Reference Guide* at http://ws.apache.org/axis/.

When I deploy a Web service and its JAR file(s), do I need to update the application server classpath?

No. The classpath is updated automatically by a special-purpose classloader. The classloader extends the application server classpath with any classes located in agile_home/integration/sdk/extensions (or the location specified for the sdk.extensions property in the agile.properties file).

If I make changes to a Web service and redeploy it, do I need to restart the application server?

No. A special-purpose handler ensures that the Web services stack is updated with the latest files

that have been deployed. Whenever a Web service request is made, the handler checks whether any JAR files located in agile_home/integration/sdk/services have been updated, added, or removed. If so, the entire Web services stack is reset. This feature allows you to recompile your code and redeploy a Web service without having to restart the application server, saving you precious development time.

Are there any Agile products that use the WSX framework?

Yes. Agile Content Service (ACS) and Agile Integration Services (AIS) both rely on WSX framework to exchange data with the Agile PLM server.

What are Agile Integration Services?

Agile Integration Services (AIS) are Web services that provide import, export, and partlist functionality. Included with these Web services are sample Java Clients, but you can create other SOAP-compliant AIS clients in other programming languages.

What is basic authentication?

Basic authentication is a simple method of authentication. It allows a client program to provide credentials in the form of an unencrypted user name and password when making a request. There is a new Web module that uses basic authentication for deploying Web service listeners. The URL for accessing Web services with basic authentication is:
http://<hostname>/Agile/ws/MyFirstWebService?wsdl

For example, use this URL for the MyFirstWebService sample:

http://<hostname>/Agile/integration/ws/MyFirstWebService?wsdl

Chapter 23

Developing Dashboard Management Extensions

This chapter includes the following:

- About Dashboard Management Extensions 387
- Developing Custom Chart Dashboard Management Extensions 388
- Developing Custom Table Dashboard Management Extensions 392
- Defining Custom (URL) Extensions 398

About Dashboard Management Extensions

Similar to Process Extensions, Dashboard Management Extensions (DX) extend the functionality of the Agile PLM system. The Extensions support the following formats to access and display PLM data on the Agile PLM Dashboard:

- ChartDataModel for charts
- Collections for tables

When data is defined using these formats, Agile servers can interpret and process this data, and Agile Java Client users with administrator privileges can define Dashboard Tabs and display them in one of the following views or layouts:

- Chart
- Table
- Custom (URL)

The SDK provides the API's that enable connecting the Agile PLM server to internal Agile databases to get the required content, format it as required by the DXs, and display the data in the Agile PLM Dashboard. Similarly, you can use other Java API's such as JDBC to connect to external databases for content.

Briefly, DXs provide the data, Dashboard Tabs and the formats to display the data (tables, charts, and URLs) are configured in Agile Java Client. Finally, Agile PLM users with proper privileges can view the Dashboard Tab and displays in Agile Web Client.

This chapter provides both background information and procedures to develop these methods.

Roles and Privileges in Dashboard Management Extensions

You must set the Dashboard Tab View privilege in Admin>User Settings>Privileges so that PLM users can view the Tabs and the related data in Web Client. In addition, Dashboard Tabs are controlled by privileges, and Agile PLM users must have the necessary roles and privileges to view the data on the Tab. Procedures to configure the views and assign privileges are fully documented in *Agile PLM Administrator Guide*.

Developing Custom Chart Dashboard Management Extensions

The `ICustomChart` interface enables creating the necessary DXs that will display the required data in chart formats. This interface exposes the method which returns an instance of the public `ChartDataModel getChart(IAgileSession session, Map params)`

Note Implementations of this interface must have no-arg constructors and they must be reentrant.

Understanding ChartDataModel and ChartDataSet

The `ChartDataModel` class organizes the input data in a chart format. It is a concrete class that is exposed to the DXs and It contains one or more `ChartDataSet(s)` that you need to construct the chart.

A `ChartDataSet` is another concrete class that is exposed to the DXs. It contains the data required to plot a chart. For example, X-axis and Y-axis values and labels. The `ChartDataModel` is a placeholder for all the data sets.

Note The `ChartDataModel` and `ChartDataSet` classes are exposed in `com.agile.px` package.

Defining a Custom Chart DX Data Source

As indicated above, chart DXs display the data in chart formats. The code in Example 20-1 uses `ICustomChart` and the exposed classes (`ChartDataModel` and `ChartDataSet`) to display the differences between morning and evening temperatures for every day of the week, using predefined input data in a chart format.

Example: Defining a DX to display data in a chart format

```
package dashboard.chart;
import java.util.Map;
import com.agile.api.IAgileSession;
import com.agile.px.ICustomChart;
import com.agile.px.ChartDataModel;
import com.agile.px.ChartDataSet;

/**
    * A Sample Dashboard DX for Charts with predefined data.
```

```
    * This Example displays a comparison chart between
    * Morning and Evening Temperatures for each day of the
    * week with predefined data.
*/

public class TemperatureComparisionChart implements ICustomChart(

/**
    * Returns custom ChartDataModel. ChartDataModel
    * is a placeholder to hold all the
    * ChartDataSet(s) and any other relevant information related to the
    charts.
    * @param session current user session.
    * @param params
    * @return com.agile.px.ChartDataModel
 */
public ChartDataModel getChart(IAgileSession session,Map params) throws
Exception{
// Create a ChartDataModel
ChartDataModel chartDataModel = new ChartDataModel("Temperatures");
// Create a ChartDataSet's for Morning and Evening Temperatures
ChartDataSet chartdataSet[] = new ChartDataSet[2];
// Create a ChartDataSet for Morning Temperatures chartdataSet[0] = new
ChartDataSet("Morning Temperatures",7);
// fill in the Morning Temperatures
double[] morTempValues = {10, 8, 12, 19, 10, 14, 13};
chartdataSet[0].setValues(morTempValues); // or setYValues can be used
instead

// Set the Labels
String[] labels = {"Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"};
chartdataSet[0].setLabels(labels);
// Create a ChartDataSet for Evening Temperatures chartdataSet[1] = new
ChartDataSet("Evening Temperatures",7);
// Fill in the Evening Temperatures double[] eveTempValues = {16, 12,
20, 15, 18, 24, 26};
chartdataSet[1].setValues(eveTempValues);
chartdataSet[1].setLabels(labels);

// Set the ChartDataSets in the Chart Model
chartDataModel.setDataSets(chartdataSet);

return chartDataModel;
}
}
```

Packaging and Deploying a Custom Chart DX Source

After developing the necessary classes for a new Chart, package and deploy them using the following procedure.

To package and deploy a Chart DX source:

1. Use your Java development environment or the Java Archive tool (or JAR tool), to create one or more JAR files for the custom action. Make sure the JAR file(s) includes a META-INF/services directory that contains the file com.agile.px.ICustomChart. This is a text file that lists the fully qualified Java class names, one class per line, for the custom action.

   You can include multiple charts in one package. For example, `com.agile.px.ICustomChart` could look like this:

   ```
   dashboard.chart.TemperatureComparisionChart
   dashboard.chart.AgileObjectsCountChart
   dashboard.chart.ActualVsBudgetedLaborCostChart
   ```

   Note Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file name are either all uppercase or all lowercase characters. Otherwise, the custom action will fail to deploy.

2. Place the JAR file(s) in the agile_home/integration/sdk/extensions folder on the same computer where the Agile Application Server is installed.

   Note If you have several application servers in a clustered environment, you must deploy the Dashboard Extension files on each server in the cluster.

Configuring Chart DXs in Java Client

In Agile Java Client, you can define Chart data sources in the Admin module. To configure the Agile PLM system settings, you must have an administrator account. This is briefly documented in the sequel below. For more information, refer to the *Agile PLM Administrator Guide*.

The data that you provide for a DX, regardless of the layout, is viewed in a Dashboard Management Tab. Because you cannot define a new Table in the Out of Box Tabs such as Executive or Financial, you must define a new Tab and then a Table within the Tab to configure a DX.

To add an optional Dashboard Management tab:

1. In Java Client, select Admin > Systems Settings > Dashboard Management and click the New Dashboard Tab icon in Dashboard Management.
2. In the Create Dashboard Tab dialog, complete the name (For example, call it Dashboard Extensions) and description fields, set the Visible field to Yes, and then click OK. Dashboard Extensions appears as an entry in the Dashboard Management.
3. Click the Order Tabs for Dashboard icon to reorder the tabs as required in Java Client.

Displaying Optional Tabs in Agile Web Client

You can display the new optional Tab in Agile Web Client and users satisfying the role and privileges requirement can view the tabs and the corresponding data. You can find the necessary procedures in *Agile PLM Administrator Guide*.

To configure a Chart type table in the optional tab:

1. Define a new tab, for example, Dashboard Extensions as shown above.
2. In the new tab (Dashboard Extensions), click Tables. The Dashboard Management - Dashboard Extensions page appears.
3. In this page, click New Dashboard Table icon to open the Create Dashboard Table dialog and define the new table.
4. Select Chart from the View List Type drop-down list.

Dashboard Table	Description/Purpose	Possible Settings
Name	Type the name of the table	String
API Name	The Name typed above, converted to *CamelCase* naming convention by Agile PLM	String
Description	Type the description of the table	String (optional)
View List Type	Lists the types of table. Select Chart (when you select Chart, additional options are displayed).	Chart, Table, Custom, Advance Search
Dashboard Extension	Lists all process extensions created for chart type list. Select the chart process extension you want.	
Visible	To enable viewing in Web Client	Yes/No
Chart Type	Select the type of chart you want displayed	Area, Bar, Line, Pie, Polar, Scatter, Stacked Area, Stacked Bar, Table
X axis	Type the X axis label	(optional)
Y axis	Type the Y axis label	(optional)
Show Legend	To display the chart legend on screen	Yes/ No
Legend Position	Select the position where the Legend should be displayed	Bottom, default, left, right, top
3D Style	To view the graph in 3D	Yes/ No
Header	Enter a header note if required	(optional)
Footer	Enter a footer note if required	(optional)

5. Complete the fields and then click OK. The name of the new Chart appears in the Dashboard

Management - Dashboard Extensions view.

Developing Custom Table Dashboard Management Extensions

The `ICustomTable` interface is defined to create DXs to display the required data in tabular formats. This interface exposes the `getTable(IAgileSession session, Map params)` method which returns an instance of the Collection class.

```
public Collection getTable(IAgileSession session, Map params);
```

Note Implementations of this interface must have a no-arg constructor and they must be reentrant.

Understanding Collection and CustomTableConstants

The Tabular Data in DX is a “collection” of Java HashMaps. Each Map key represents an attribute in the Table View and the Map represents a row in the table.

The property “Attribute” of a column in View defines the mapping between the data model and the Table View. The value of this property corresponds to the key of a HashMap entry.

- HashMap keys – For HashMap entries, an attribute is defined in the Table view. For example, a HashMap entry with “name” as its key value, the property “Attribute” of this attribute will have the value “name.” The `get('name')` method will provide the display data for this attribute.
- Link, Image, Money, Text, Date and Numeric Data – These data types are supported in Tabular DX formats and return objects with the following properties.
  - Text – Date, and Numeric Data types do not require any additional properties.
  - Link – A valid URL (as String) serves as the target and label for the display. The properties expected for a link data type are the same for the internal and external links. The DX users resolve the URL for internal links and add them to the URL property. The DX users can specify the target property as "RightPane" for internal links. By default, the links will be targeted to a new window.
  - Image – Images are expected to return an image URL (as String) and label to be displayed as a tool tip on the Image.
  - Money – Currency code (String) and Value (Number) needs to be provided for Money Data types

Note Keys that support the Link, Money and Image data properties are provided as constants in the class CustomTableConstants. A constant SERVER_URL is provided in this class. You can use it get the Server URL in the DX's from the params.

Defining a Custom Table DX Data Source

The sample Dashboard DX in the following example creates a collection of rows with predefined data for display in the Dashboard. Each row is a Java Map object with key-value pairs that correspond to each column in the table. This value appears in each cell of the column in the table. The key is the mapping Attribute name in the View. When creating new Attributes (columns) in the View, it is necessary to supply this key in the Attribute field. Attribute Names and the corresponding Data type for this DX are as follows:

Attribute	Corresponding Data Type
myString	Text
myExternalLink	Link
myDate	Date
myMoney	Money
myNumber	Numeric
myImage	Image

Example: Defining a Dashboard extension to display data in tabular format

```
package dashboard.table;
import java.util.*;
import com.agile.api.IAgileSession;
import com.agile.px.ICustomTable;
import com.agile.px.CustomTableConstants;

/** This Sample Dashboard DX creates a collection
* of rows with predefined data
* in the format to be displayed in the Dashboard.
* Each row is a Java Map object which has
* key-value pairs corresponding to each column in the Table.
* The value is displayed in each Cell of the
* column in the table. The key is the
* mapping Attribute name in the View.
* While creating new Attributes (Columns) in the View,
* you must supply this keyin the Attribute field.
* The corresponding Attribute Names and
* Data type for this DX are listed below.
*  <table border="1">
*  <tr><td><b>Attribute</b> </td><td><b>Data   Tye<b></td></tr>
*  <tr><td>myString         </td><td>Text      </td></tr>
*  <tr><td>myExternalLink   </td><td>Link      </td></tr>
*  <tr><td>myDate           </td><td>Date      </td></tr>
*  <tr><td>myMoney          </td><td>Money     </td></tr>
```

```
*  <tr><td>myNumber         </td><td>Numeric   </td></tr>
*  <tr><td>myImage          </td><td>Image     </td></tr>
*  </table>
*
*/

public class DashboardSampleTable  implements ICustomTable {
 /**
* Returns custom table data in form of collection of rows.
* Row is assumed to be a java Map object.
* @param session the user session
* @param params
* @return : java.util.Collection
 */
public Collection getTable (IAgileSession session,Map params) throws
Exception{
String serverUrl =
    (String)params.get(CustomTableConstants.SERVER_URL);
String baseUrl =
    serverUrl.substring(0,serverUrl.lastIndexOf('/'));
ArrayList result = new ArrayList();
// 1st Row Entry
HashMap row1 = new HashMap();

// For Text type
row1.put("myString","Manoj Yeturu");

// For Numeric type
row1.put("myNumber",new Double(10000));

// For Date Type
row1.put("myDate",new Date());

// For Image Type. The url for image and label (for tooltip) properties
sre set
HashMap hm1Image = new HashMap();
hm1Image.put(CustomTableConstants.URL,baseUrl+"/images/action_noshad.gi
f");
// Tool Tip hm1Image.put(CustomTableConstants.LABEL,"Action_Noshad");
row1.put("myImage",hm1Image);
// For Money Type. The Currency and value properties are set HashMap
hm1Money = new HashMap();
hm1Money.put(CustomTableConstants.MONEY_CURRENCY_CODE,"USD");
hm1Money.put(CustomTableConstants.MONEY_VALUE,new Integer(3000));
row1.put("myMoney",hm1Money);
// For External Link, url, label (display string) and target
(Rightpane,_new etc) are set
HashMap externalLink1 = new HashMap();
externalLink1.put(CustomTableConstants.URL,"http://www.agile.com");
```

```
externalLink1.put(CustomTableConstants.LABEL,"Agile");
externalLink1.put(CustomTableConstants.TARGET,"_new");
row1.put("myExternalLink",externalLink1);
result.add(row1);

// 2nd Row Entry
HashMap row2 = new HashMap();

// For Text type
row2.put("myString","Venkat Tipparam");

// For Numeric type
row2.put("myNumber",new Double(50000));

// For Date Type
row2.put("myDate",(new Date()));

// For Image Type
HashMap hm2Image = new HashMap();
hm2Image.put(CustomTableConstants.URL,baseUrl +
"/images/addressdown.gif");

// Tool Tip
hm2Image.put(CustomTableConstants.LABEL,"Addressdown");
row2.put("myImage",hm2Image);
// For Money Type
HashMap hm2Money = new HashMap();
hm2Money.put(CustomTableConstants.MONEY_CURRENCY_CODE,"INR");
hm2Money.put(CustomTableConstants.MONEY_VALUE,new Integer(4000));
row2.put("myMoney",hm2Money);
// For External Link
HashMap externalLink2 = new HashMap();
externalLink2.put(CustomTableConstants.URL,"http://www.agile.com/servic
es/support.asp");
externalLink2.put(CustomTableConstants.LABEL,"Supprt");
externalLink2.put(CustomTableConstants.TARGET,"_new");
row2.put("myExternalLink",externalLink2);
result.add(row2);

return result;
}
}
```

Packaging and Deploying a Custom Table DX Source

After developing the required classes for a new Table, package and deploy them as shown below.

To package and deploy a Table DX source:

1. Use your Java development environment, or the Java Archive tool (or JAR tool), to create one or more JAR files for the custom action. Make sure the JAR file(s) includes a META-INF/services directory that contains a file named `com.agile.px.ICustomTable`. This is a text file that lists the fully qualified Java class names, one class per line, for the custom action.

   You can include multiple charts in one package. For example, the `com.agile.px.ICustomTable` file could look like this:

   ```
   dashboard.chart.ActualVsBudgetedLaborCostTable
   dashboard.chart.DashboardSampleTable
   dashboard.chart.QueryDashboardPrograms
   ```

   Note Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file has a name with all uppercase or all lowercase characters. Otherwise, the custom action is not deployed.

2. Place the JAR file(s) in the agile_home/integration/sdk/extensions folder on the same computer where the Agile Application Server is installed.

   Note If you have several application servers in a clustered environment, you must deploy the Dashboard Extension files on each server in the cluster.

Configuring Table DXs in Java Client

Similar to Chart type DXs, you can use an existing Dashboard Management tab, or create your own optional tab to add your Table DXs.

To Add a Table to a Tab:

1. Define a new tab. For example, Dashboard Extensions as shown above.
2. In the new tab (Dashboard Extensions), click Tables. The Dashboard Management - Dashboard Extensions page appears.
3. In this page, click the New Dashboard Table icon to open the Create Dashboard Table dialog and define the new table.
4. Select Table from the View List Type drop-down list. The Create Dashboard Table dialog displaying the necessary fields appears.

5. Complete these fields and then click OK. The new table is created.

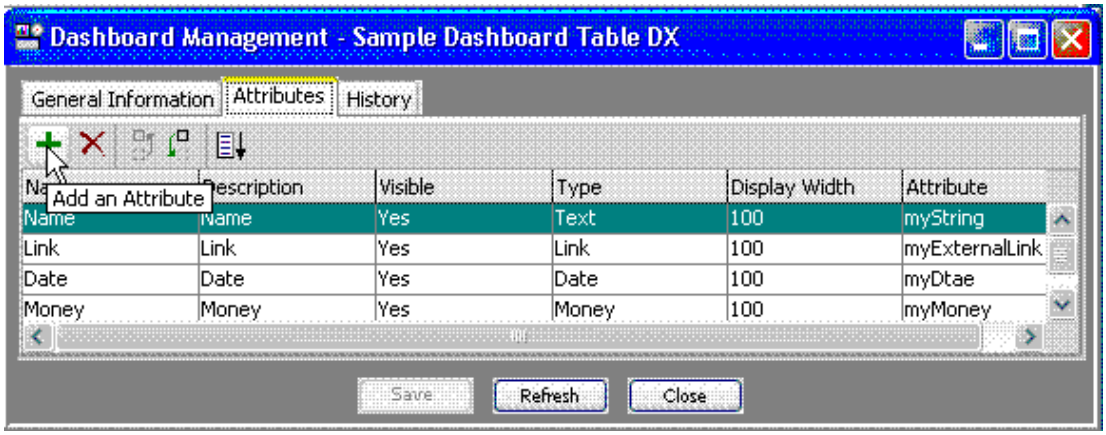
Dashboard Table	Description/Purpose	Possible Settings
Name	Type a name for the table	String
API Name	The Name filed typed above, converted to *CamelCase* naming convention by AgilePLM	String
Description	Type a description of the table	String
View List Type	Lists the types of table. Select **Table**	Chart, Table, Custom, Advance Search
Dashboard Extension	Lists all the process extensions created for the type of Table in view	These are the attributes that were defined in the Packaging and Deploying a Custom Action on page 349.
Variable	To enable in Web client	Yes/No

To Add Data to Tables:

1. Double-click the new table that you created in To Add a Table to Tab on page 396.
2. Click Attributes and then the Add an Attribute icon to create a new attribute.

> **Note** Agile currently supports Text, Numeric, Image, Date, Money, and Link type data as table attributes. They are listed and defined in Packaging and Deploying a Custom Table DX Source on page 396

Figure 26: Sample Dashboard Table DX



3. In the General Information tab, map the Attribute field to the attribute name in the DX.

> **Note** You are now defining the attributes (columns) that will show up in the table on the Dashboard Tab. The property “Attribute” defines the mapping between the data model and the view. For example, if the attribute name in the DX is myString and the selected attribute type is Text, map the attribute field whose attribute name is myString.

4. For more information, refer to the *Agile PLM Administrator Guide*.

Defining Custom (URL) Extensions

A Dashboard PX of the type URL is configured to initiate from Dashboard Management. When defining Custom extensions, simply select Custom as the table type. See “To Add a URL to a Tab" below. No other mapping is required for Dashboard PX's of type URL.

Note URL Process Extensions are defined in the Process Extensions Library to initiate from Dashboard Management.

To Add a URL to a Tab:

1. Define a new tab, for example, Dashboard Extensions as shown above.
2. In the new tab (Dashboard Extensions), click Tables. The Dashboard Management - Dashboard Extensions page appears.
3. In this page, click New Dashboard Table icon to open the Create Dashboard Table dialog and define the new table.
4. Select Custom from View List Type drop-down list. The Create Dashboard Table dialog displaying fields listed in the following appears.
5. Complete the Create Dashboard Table dialog fields and then click OK.

Dashboard Table	Description	Possible Settings
Name	Type a name for the URL	String
Description	Type a description	String
View List Type	Lists types of table. Select Custom	Chart, Table, Custom, Advance Search
Dashboard Extension	Lists all process extensions created for Custom type list.	Employee Portal, Yahoo, Google, Process Extension_URLs
Visible	To enable in Web Client	Yes/No

Chapter 24

Agile PLM Events and Event Framework

This chapter includes the following:

- Understanding Agile PLM Events and Event Framework 399
- Key Components of an Agile PLM Event 399
- Event Generation, Processing, and Execution 403

Understanding Agile PLM Events and Event Framework

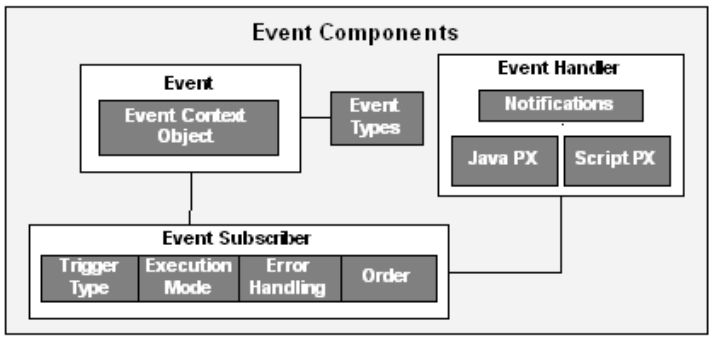
In Agile PLM, Events act as trigger points for generating an automation action within the PLM application. Every Event is generated from a source within Agile PLM applications. The source can be a business action triggered by a user, a UI action, or system initiated source such as a timer. These sources can signal other PLM modules that something (an event) has occurred within the application and it may require an action. The required action can be taken by a user or the PLM module. Event-driven applications greatly facilitate the management of event-based integrations and complex event analysis in real time modes.

In Agile SDK environment, Event framework extends the PX framework which facilitated automating and extending the Agile PLM Applications. Event framework provides a flexible environment for rapid development and deployment of event-driven applications. To support this environment, Event framework provides a comprehensive set of data parameters to create, configure, and execute different types of Agile PLM Events.

Key Components of an Agile PLM Event

Event framework empowers users to configure Events and Event subscribers. The basic components of Event framework are shown in the following illustration and described in the following paragraphs.

Figure 27: Event components



Event Types

In Agile PLM, *Event type* refers to a particular action, for example, *Create Object*, *Delete Object, Audit for Workflow*. Agile PLM provides a list of pre-defined Event types for which an event can occur.

For each Event type, depending on its Handler type, there is a corresponding Java and Script interface for its Event Context object. For example, for *Create Object,* there is the `ICreateEventInfo` Java interface and `ICreateScriptObj` Script interface respectively.

Event type is selected from the drop-down list in Java Client's *Create Event* dialog when the Event is created. To access this dialog, refer to *Agile PLM Administrator Guide*. Default Event types and their descriptions is shown in the figure below.

Figure 28: Event Types and their descriptions

Name	Description
Approve for Workflow	Workflow, Approve
Audit for Workflow	Workflow, Audit
Cancel Check Out Files	Files, Cancel Check Out
Change Approvers or Observers for Workflow	Workflow, Change Approvers or Observers
Change Status for Sourcing Object	Sourcing Object, Change Status
Change Status for Workflow	Workflow, Change Status
Check In Files	Files, Check In
Check Out Files	Files, Check Out
Comment for Workflow	Workflow, Comment
Compliance Rollup on Object	Object, Compliance Rollup
Create Object	Object, Create
Delete Object	Object, Delete
Escalation for Workflow	Workflow, Escalation
Export Object	Object, Export
Extend Actions Menu	Actions Menu, Extend
Extend Tools Menu	Tools Menu, Extend
Get File	File, Get
Incorporate Item	Item, Incorporate
Promotion Failure for Workflow	Workflow, Promotion Failure
Purge Version Files	Files, Purge Version
Reject for Workflow	Workflow, Reject
Reminder for Workflow	Workflow, Reminder
Save As Object	Object, Save As
Scheduled Event	Event, Scheduled
Transfer Authority	Authority, Transfer
Unincorporate Item	Item, Unincorporate
Update Relationship	Relationship, Update
Update Table	Table, Update
Update Title Block	Title Block, Update

Note You can neither create new Event types, nor delete these Event types.

Event Handler and Handler Types

An Event Handler represents a custom action that is called when the Event is raised. They extend the function of an action taken by a user, interface, or the system when the Event is triggered. Information about the Event is passed from the Event to the Handler by the *Event Context object*. Handlers are invoked by Event Triggers.

The SDK and Event framework support the following Event Handler types:

- Java PX – Java PXs are Java process extensions that implement the `IEventAction` interface in `com.agile.px` package and trigger the compiled Java code. See Event Information Objects.
- Script PX – Script PXs are Script process extensions based on Groovy script language.

  The Groovy script code is directly stored in Agile PLM databases. Event Script objects (Event Script PX handlers) that you develop are text files that are deployed on the Agile PLM server using the Event Management Node in Java Client. For information on submitting the text files, see Working with Agile PLM Administrator on page 433. For information about Groovy implementation in Event framework and Event Script PX development process, see Groovy Implementation in Framework Environment on page 461.
- Notification – Agile PLM has the capability to send notifications to users, either when the user is required to take action, or to notify the user of actions that have occurred. Notifications can be triggered from the SDK and Script. Event Notifications are addressed in the Agile PLM Administrator Guide. To use the SDK to programmatically send Notifications, refer to "Sending Notifications with Agile SDK" in *SDK Developer Guide - Using Agile APIs*.

Note SDK developers use the Java PX and Script PX to extend the capabilities of Agile PLM. Notification templates are used by the Agile PLM Administrator to define and configure Notifications for action or information.

Event Subscribers

An Event Subscriber links an Event Handler to a specific Event. Thus, when a particular event occurs, Event Handlers associated with the Event through Event Subscribers are initiated in the order requested by the configuration. The Handler action includes invoking a Java Process Extension, a Script Process Extension, or sending Notifications.

Event Trigger and Trigger Types

Event Trigger is used to determine when an Event Extension is raised. The Event framework provides the hooks to automate actions when an incident happens. For example, when a change of status occurs, a CEO approval is required, the trigger signals the occurrence of an action which will subsequently notify all Event subscribers for that Event. Most Agile PLM Events are associated with business actions. Examples are:

- Create Object Event – associated with the object creation action
- Change Status for Workflow Event – associated with the Workflow status change action

Event Trigger Types

- Pre – This trigger type signals a point prior to the occurrence of an action. The Pre trigger is commonly used for events that require data validation or other preparations for the upcoming action. Event Handlers are invoked synchronously from the “Pre” trigger point.
- Post – This trigger type signals a point immediately after the action's changes are committed to the database. The post trigger is commonly used for Events that perform auditing tasks, notifications, and integration tasks with external systems related to the completed action. Event Handlers are invoked synchronously or asynchronously from the “Post” trigger point.

Synchronous and Asynchronous Execution Modes

Event Handlers are invoked synchronously or asynchronously. In general, the term synchronous means it is run as part of the application within the current action. A synchronous operation blocks a process until the operation completes while an asynchronous operation is non-blocking and only initiates the operation.

In Agile PLM, the difference between the two modes is:

- Synchronous – In this mode, the Event Handler is executed in the same thread (as part of the execution of the action taken) as the Agile PLM thread that triggers the event (for example, a change in a Workflow status). The original Agile PLM action will resume after the handler action is completed.
- Asynchronous – In this mode, the Event Handler has its own thread (runs independent of the execution of the action taken) and it cannot be stopped once it is started. This transaction is either committed or rolled back based on its own status. The Agile PLM thread that triggers the event will continue to run independently regardless of the Handler action has finished or not (Non-block). Notifications are always handled asynchronously.

Event Error Handling Rule

Event Error Handling Rule is used in conjunction with Event Handlers executed synchronously. Options are Continue and Stop. The selected option determines the behavior of Agile PLM when an error is encountered while executing the Event Handler. For more information on error handling rules, refer to Agile PLM Administrator Guide.

- Continue – If there is an error during the execution of a Handler with synchronous execution mode, the error is ignored.
- Stop – If there is an error during the execution of a Handler with synchronous execution mode, the original action and the Event Subscription is ceased.

Event Order

Event Order is a positive integer that determines the sequential "Order" in which the Event handler is invoked. This allows you to control the execution order of Event Handlers when there are multiple Event Subscribers for the same Event type on the same Agile object.

Note	If you have both Custom PXs and Java synchronous PXs configured for a Workflow Change Status action, Java PXs always execute before Custom PXs.

Event Generation, Processing, and Execution

The general flow of an Agile PLM transaction from the moment the Event is triggered from PLM Clients to its termination is shown in the following illustration.

Actions for different conditions and those at the end of the flow are summarized below:

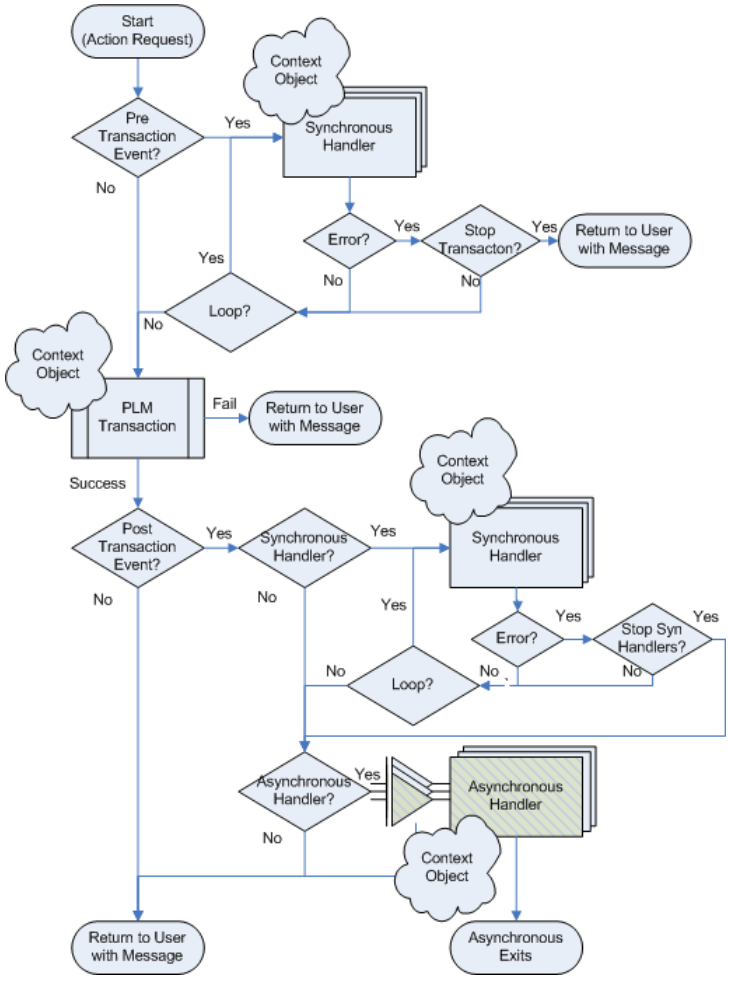
- When an action is performed by a Client, Agile PLM checks whether a Pre-Event for the requested action was configured through a Subscriber. If there is a Subscriber, Agile PLM invokes Event Handlers configured through the Subscriber in the proper order. After a Handler completes its respective tasks, Agile PLM uses the settings configured in Subscriber Error Handling Rule to determine whether it should stop or continue if an error is encountered. Agile PLM invokes the next Event Handler from the remaining Event Handlers until all Event Handlers have completed their respective tasks. The context object retains updates made within each Event Handler. After all Pre-Event Handlers are complete, Agile PLM starts the original action requested by the user with the context object retaining the updates.

  Important — Oracle does not support and recommends not to use the SDK to create or update an operation in Pre-Event. If the Pre-Event is configured in a Script PX, the PX inherits the transaction and in the event the operation in the PX fails, the entire transaction is rolled back and the ensuing behavior undefined.

- Once the original action is performed, a similar process launches all Synchronous Post-Event Handlers. The context object information will be retained throughout this process. Synchronous Post-Event Handlers cannot cause the original action to rollback because it has already occurred. Once all Synchronous Post-Event Handlers are successfully completed, Agile PLM will start Asynchronous Post-Event Handlers (if they exist) as independent processes without a defined order.

- If there is an error occurred when executing the Synchronous Pre-Event Handler, and the error Handler rule is Stop, Agile PLM will return the error message to the user.

Figure 29: Triggering and processing an Agile PLM Event



Chapter 25

Working with Event Context Objects

This chapter includes the following:

- Understanding Event Context Objects 405
- Working with Event Information and Event Script Objects 409
- Event FAQs 437

The flowing paragraphs provide describe the role of Event Context objects, their creation, different Event types, and the information they maintain.

Understanding Event Context Objects

The Context object passes information from the Event to the Handler and between Handlers. When an Event is raised, an Event context object is created. Information maintained by the context object includes Event type, pre- and post-Event triggers, plus business-related data for the given Event, such as the Agile object for which the Event occurred. The business-related data varies for based on Event type.

Different Event types have different Context objects. Interfaces for Context objects are documented in Event Information Objects on page 406 and Event Script Objects on page 408. These interfaces are documented in Javadoc HTML files as *Event Information objects* (Java interfaces) and *Event Script objects* (Script interfaces). To find the Javadoc HTML files in the SDK_samples.zip folder, see the Note in client-Side Components on page 2.

Persistent and Transient Data

In Event framework, objects passed to Event Handlers (Java PX or Script PX) by Context objects contain "Persistent" and "Transient" data.

- Persistent data – This is data that is already in PLM databases. All Agile SDK APIs and Process Extensions including Web or URL extensions deal with this type of data. When you use `getDataObject()` to get the values of an `IDataObject` object, the data you get is already in the database, hence "Persistent."

  Note Only the `getDataObject()` method returns Persistent data. Other context object get data methods will return Transient data unless stated otherwise in Javadoc API definition.

- Transient data – This is data that is not in the PLM database as yet and is in a state of change. Transient data contains information about user requests for the action that triggered the Event.

When the Event is triggered, Agile PLM creates Transient data in the Event Context object. The same Transient data is passed on by the Context object to all Handlers for “Pre” and "Post" unless specified otherwise in the Javadoc API definition.

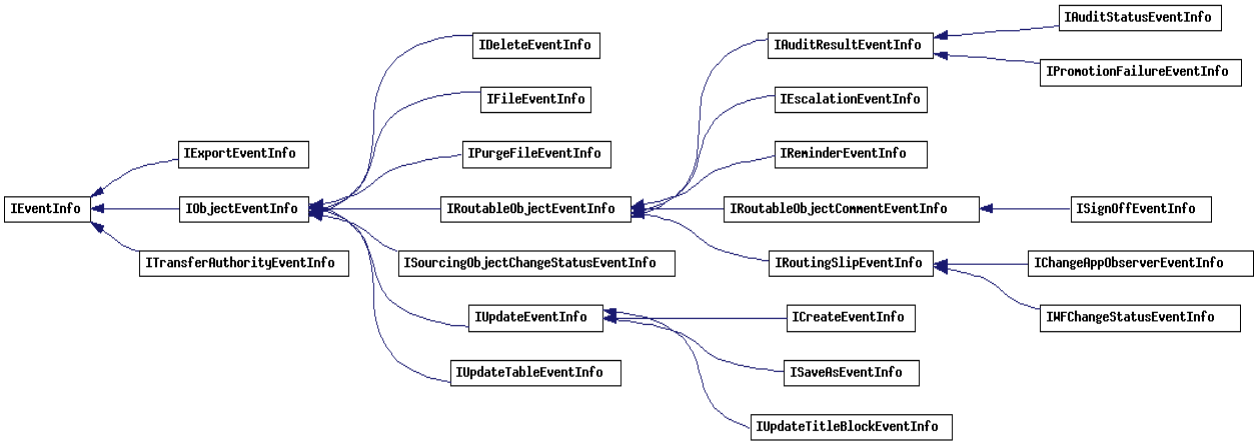
Note You must not modify Transient data in the "Post" Event triggering instance. If modified, it will throw an exception.

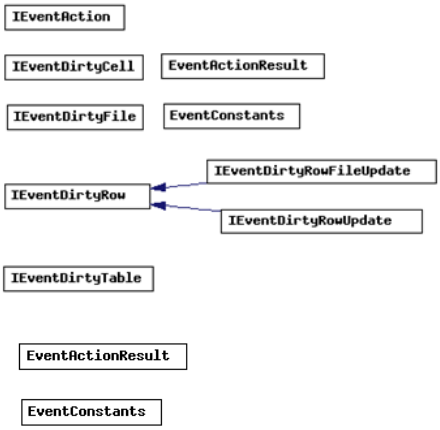
Event Information Objects

These are the interfaces for the Java PXs.

Event Type	Event Information Object
"Approve for Workflow" and "Reject for Workflow"	`ISignOffEventInfo`
"Audit for Workflow"	`IAuditStatusEventInfo`
"Change Approvers" or "Observers for Workflow"	`IChangeAppObserverEventInfo`
"Change Status for Sourcing Object"	`ISourcingObjectChangeStatusEventInfo`
"Change Status for Workflow"	`IWFChangeStatusEventInfo`
"Comment for Workflow"	`IEscalationEventInfo`
"Create Object"	`IRoutableObjectCommentEventInfo`
"Delete Object"	`IDeleteEventInfo`
"Escalation for Workflow"	`IEscalationEventInfo`
"Export Object"	`IExportEventInfo`
"Extend Tools Menu" and "Scheduled Event" and the base interface for all Event information objects	`IEventInfo`
"Get File", "Check In Files", "Check Out Files", and "Cancel Check Out Files"	`IFileEventInfo`
"Incorporate Item", "Unincorporate Item", "Extend Actions Menu", and "Compliance Rollup on Object"	`IObjectEventInfo`
"Promotion Failure for Workflow"	`IPromotionFailureEventInfo`
"Purge Version Files"	`IPurgeFileEventInfo`
"Reminder for Workflow"	`IReminderEventInfo`
"Save As Object"	`ISaveAsEventInfo`
"Transfer Authority"	`ITransferAuthorityEventInfo`
"Update Table" and "Update Relationship"	`IUpdateTableEventInfo`
"Update Title Block"	`IUpdateTitleBlockEventInfo`

Figure 30: Event information objects class hierarchy



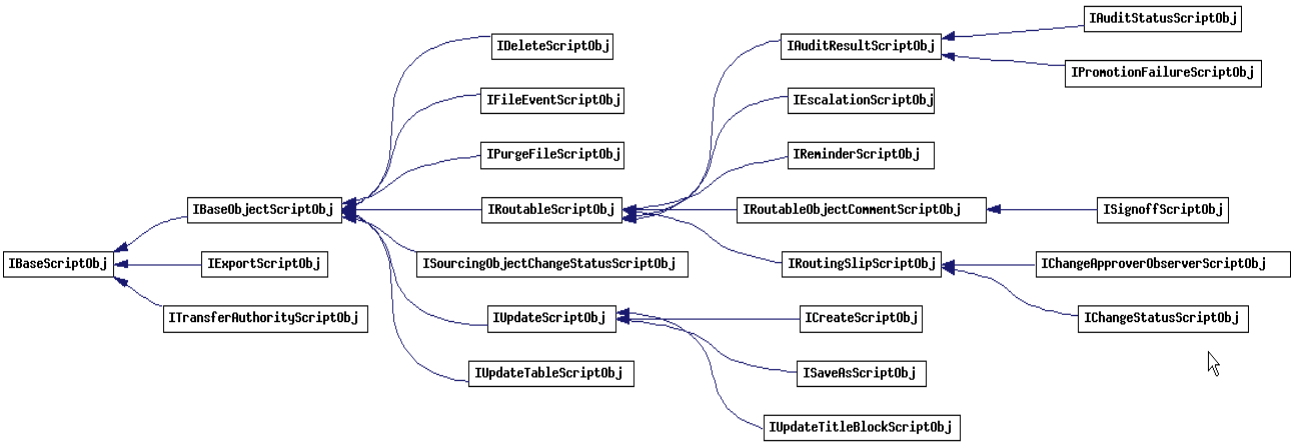


Event Script Objects

These are the interfaces for the Script PXs.

Event Type	Script Object Interface
"Approve for Workflow" and "Reject for Workflow"	ISignoffScriptObj
"Audit for Workflow"	IAuditStatusScriptObj
"Change Approvers or Observers for Workflow"	IChangeApproverObserverScriptObj
"Change Status for Sourcing Object"	ISourcingObjectChangeStatusScriptObj
"Change Status for Workflow"	IChangeStatusScriptObj
"Comment for Workflow"	IRoutableObjectCommentScriptObj
"Create Object"	ICreateScriptObj
"Delete Object"	IDeleteScriptObj
"Escalation for Workflow"	IEscalationScriptObj
"Export Object"	IExportScriptObj
"Extend Tools Menu" and "Scheduled Event "Event Script objects, also the base interface for all Event Script objects	IBaseScriptObj
"Get File', "Check In Files", 'Check Out Files' and "Cancel Check Out Files"	IFileEventScriptObj
"Incorporate Item", "Unincorporate Item", "Extend Actions Menu", and "Compliance Rollup on Object"	IBaseObjectScriptObj
"Promotion Failure for Workflow"	IPromotionFailureScriptObj
"Purge Version File Folder"	IPurgeFileScriptObj
"Reminder for Workflow"	IReminderScriptObj
"Save As Object"	ISaveAsScriptObj
"Transfer Authority"	ITransferAuthorityScriptObj
"Update Table" and "Update Relationship"	IUpdateTableScriptObj
"Update Title Block"	IUpdateTitleBlockScriptObj

Figure 31: Event Script Objects Class Hierarchy



Note In Script PX, you can invoke any SDK or Java call. You can also access any third party Java or Groovy Libraries if they are deployed on the SDK extensions directory.

Working with Event Information and Event Script Objects

Information provided in this section uses the Event Information objects and Event Script Objects to develop PX Handlers for Event-related actions, for example, general object actions such as Create Object, Delete Object; Workflow actions such as Change Status for Workflow.

Descriptions and samples for Event Information objects and Event Script objects appear in the following paragraphs. Other information includes code samples using these objects and guidelines to ensure proper handling of special instances of Events in Agile PLM.

Working with Base Event Actions

Base Event Information objects are Java interfaces that are used by all other Event Information objects. See the illustration in Event Information Objects Class Hierarchy .

Base Event Information Object - Java PX

The Base Event Information object is `IEventInfo`.

- Purpose and function – This is the interface for *Extend Tools Menu* and *Scheduled Event* Event Information objects. In addition, `IEventInfo` is the inherited interface for all Event Information objects. It maintains information on Event type, Event trigger type, Event name, Event Subscriber name, Event Handler name, and user defined Maps.

  User defined Maps serve as a place holder for any user defined data and provide a communication channel between subscribers. They are set in the Synchronous Java PXs and read by subsequent Synchronous Java PXs and Asynchronous Java PXs. Maps set inside Asynchronous PXs cannot be used by other Java PXs. If the Java PX fails, the Map is still accepted by the Agile PLM and is passed to the next Java PX but all other changes in the Event Context objects are discarded.

`IObjectEventInfo` is the base event information object for object-related events.

- Purpose and function – `IObjectEventInfo` contains Agile object for which the Event is triggered. Also, it is the interface for Incorporate Item, Unincorporate Item, Extend Actions Menu, and Compliance Rollup on object Event Information objects and inherits from `IEventInfo`.
- Inherited interface: `IEventInfo`

The following examples show using `IEventInfo` and `IObjectEventInfo`.

Example: Using IEventInfo

```
private void testIEventInfo(IEventInfo req)throws APIException{
    // getEventType()
    int evttype = req.getEventType();

    // getEventTriggerType()
    int evtTriggerType = req.getEventTriggerType();

    // getEventName()
    String eventName = req.getEventName();

    // getEventSubscriberName()
    String subscriberName = req.getEventSubscriberName();
    // getEventHandlerHame()
    String handlerName = req.getEventHandlerName();
    // setUserDefinedMap()
    Map map = new HashMap();
    map.put("METHOD", "setUserDefinedMap()");
    req.setUserDefinedMap(map);

    // getUserDefinedMap()
    Map map2 = new HashMap();
    map2 = req.getUserDefinedMap();
    String mapValue = map2.get("METHOD").toString();
}
```

Example: Call made by IObjectEventInfo

```
private void testIObjectEventInfo(IEventInfo req)throws APIException{
    String objNumber = "";
    IObjectEventInfo info = (IObjectEventInfo)req;

    // getDataObject()
    IDataObject obj = info.getDataObject();
    if (obj==null)
              objNumber = "NULL";
    else
              objNumber = obj.getName();
    }
```

Base Event Script Objects - Script PX

Base Event Script objects is `IBaseScriptObj`.

- Purpose and function – This is the interface for *Extend Tools Menu* and *Scheduled Event* Event Script objects. `IBaseScriptObj` is the inherited interface for all Event Script objects. `IBaseScriptObj` maintains information on Event type, Event trigger type, Event name, Event Subscriber name, Event Handler name, and User Defined Maps.

  Note This interface provides a Script method for sending Agile PLM Notifications.

`IBaseObjectScriptObj` is the base event information object for object-related events.

- Purpose and function – `IBaseObjectScriptObj` contains Agile object for which the Event is triggered. Also, it is the interface for Incorporate Item, Unincorporate Item, Extend Actions Menu, and Compliance Rollup on object Event Information objects and inherits from `IBaseScriptObj`.
- Inherited interface: `IBaseScriptObj`

The following examples show using `IBaseScriptObj` and `IBaseObjectScriptObj`.

Example: Using IBaseScriptObj

```
This example opens a SDK session and retrieves information pertaining
to the Event.
void invokeScript(IBaseScriptObj obj)
{
    // getEventType()
    int evttype = obj.getEventType();

    // getEventTriggerType()
    int evtTriggerType = obj.getEventTriggerType();

    // getEventName()
    String eventName = obj.getEventName();

    // getEventSubscriberName()
    String subscriberName = obj.getEventSubscriberName();
    // getEventHandlerHame()
    String handlerName = obj.getEventHandlerName();

    //logMonitor ()
    obj.logMonitor("Status is Passed");

    //getAgileSDKSession()
    IAgileSession session = obj.getAgileSDKSession();

    //getPXEventInfo ()
    IEventInfo req = obj.getPXEventInfo();

    //sendNotification()
    obj.sendNotification("Test", true, [ "admin"],  " passed from
    BaseScriptObj" + eventName);
    // setUserDefinedMap()
    obj.setUserDefinedMap ( [ 'Agile 93' :  'PLM Product',
    'Scripting':'Is Fun Tool']);
    // getUserDefinedMap()
```

```
    Map myMap = obj.getUserDefinedMap ();

}
```

Example: Using IBaseObjectScriptObj

```
IBaseObjectScriptObj retrieves Class ID and object number and sets P1,
P2, and P3 attribute values of the object
void testIBaseObjectScriptObj(IBaseScriptObj obj)
{
    String objNumber = "";
    //Disable all the warning exceptions raise
    obj.disableAllWarnings();

    //get class ID
    int classID = obj.getClassId();

    //get object number
    objNumber = obj.getObjectNumber();

    //set attributes
    obj.setValueByAttId(CommonConstants.ATT_PAGE_TWO_TEXT01,"Text
    Value");
    // setValueByAttId one from each attribute type
    /*
    * Date:      Page Two.Date01
    * Text:      page Two.Text01
    * MultiText: Page Three.MultiText10
    * List:      Page Two.List01
    * MultiList: Page Three.MultiList01
    * Numeric:   Page Two.Numeric 01
    * Money:     Page Three.Money01
    */

    // get attribute value
    obj.getValueByAttId(CommonConstants.ATT_PAGE_TWO_TEXT01);
    //log information to handler monitor
    obj.logMonitor("Object Number:" + objNumber);
    obj.logMonitor("Class Id :" +  classID);
}
```

Working with General Object Actions

General Object actions are actions such as Create, Delete, Save As, and Update Title block. Information objects and Script Event objects for these Events are grouped and described according to their inherited interfaces.

General Object Actions - Java PX

Create Object

The Information object for this Event is `ICreateEventInfo`.

- Purpose and function – `ICreateEventInfo` retrieves the number and Subclass identifier of the requested new object. It can also overwrite the number and Subclass set by the clients.
- Inherited interfaces – `IUpdateEventInfo` , `IObjectEventInfo`, `IEventInfo`

- Inherited interface purpose and function:
  - `IObjectEventInfo`
  - `IUpdateEventInfo` retrieves the Array of Dirty attributes and users can overwrite or set new attributes and values.

The following examples show using `ICreateEventInfo` and `IUpdateEventInfo`.

Example: Using ICreateEventInfo

```
private void testICreateEventInfo(IAgileSession session, IEventInfo
req) throws APIException {
       ICreateEventInfo info = (ICreateEventInfo)req;
       String number = "";
       Integer subclass = null;
       String newNumber = getNewNumber(info);  // user defined method
       Integer newSubclassId = getNewSubclassId(session, req); // user
       defined method
       Integer newSubclass = null;
       // getNewNumber()
       number = info.getNewNumber();

       // getNewSubclassId()
       subclass = info.getNewSubclassId()

       // setNewNumber()
       info.setNewNumber(newNumber);

       // setNewSubclassId()
       info.setNewSubclassId(newSubclassId);
       newSubclass = info.getNewSubclassId();
}
```

Example: Using IUpdateEventInfo

```
private void testIUpdateEventInfo(IAgileSession session, IEventInfo
req) throws Exception {
    // Interface methods
    IUpdateEventInfo info = (IUpdateEventInfo)req;


    //getCells()
    IEventDirtyCell[] cells = info.getCells();
    // getAttributeIds()
    Integer[] attrs = info.getAttributeIds();

    // setCell()
    // Get class specific P1 attribute
    Integer p1attrId = getP1Attribute(session, info);

    // Override client value
    info.setCell(p1attrId, "set desc from CO");


    // Add new Dirty value
    info.setCeldirtyl(CommonConstants.ATT_PAGE_TWO_TEXT02, "setCell()");
    String value2 =
    info.getValue(CommonConstants.ATT_PAGE_TWO_TEXT02).toString();
    // setCell() one from each attribute type
    * Date:      Page Two.Date01
    * Text:         already cover
```

```
    * MultiText: Page Three.MultiText10
    * List:      Page Two.List01
    * MultiList: Page Three.MultiList01
    * Numeric:   Page Two.Numeric 01
    * Money:     Page Three.Money01
    */

    Integer subClassId = getSubclassId(info);
    IAttribute attr1 =
       session.getAdminInstance().getAgileClass(subClassId).getAttribute
       (CommonConstants.ATT_PAGE_TWO_LIST01);
    IAttribute attr2 =
       session.getAdminInstance().getAgileClass(subClassId).getAttribute
       (CommonConstants.ATT_PAGE_THREE_MULTILIST01);
    IAgileList list1 =
       (IAgileList)attr1.getAvailableValues();
    list1.setSelection(new Object[]{"b"});
    IAgileList list2 = null;
    if (attr2!=null){
       list2 = (IAgileList)attr2.getAvailableValues();
       list2.setSelection(new Object[]{"a", "b", "e"});
    }
    SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy");
    String d = "1/31/2009";
    Date date = df.parse(d);
    info.setCell(CommonConstants.ATT_PAGE_TWO_DATE01, date);
    String multitext = "set multitext field in CO";
    info.setCell(CommonConstants.ATT_PAGE_TWO_LIST01, list1);
    info.setCell(CommonConstants.ATT_PAGE_TWO_LIST02, list1); // To test
    IEventDirtyCell
    info.setCell(CommonConstants.ATT_PAGE_TWO_NUMERIC01, 888.66);
    if (attr2!=null){
       info.setCell(CommonConstants.ATT_PAGE_THREE_MULTITEXT10,
       multitext);
       info.setCell(CommonConstants.ATT_PAGE_THREE_MULTILIST01, list2);
    }
    Money money = new Money (new Integer(100), "USD");
    // removeCell()
    info.removeCell(CommonConstants.ATT_PAGE_THREE_TEXT01);

 }
```

Update Title Block

The Information object for this Event is `IUpdateTitleBlockEventInfo`.

- Purpose and function – `IUpdateTitleBlockEventInfo` is the interface for Update Title Block Event information object. Checks whether this is a redline update or undo-redline update on the Title Block of `IItem` object which has originated from the Affected Items table of the `Change` object
- Inherited interfaces – `IUpdateEventInfo` , `IObjectEventInfo`, `IEventInfo`

Save As Object

The Information object for this Event is `ISaveAsEventInfo`.

- Purpose and function – `ISaveAsEventInfo` performs the following tasks:
  - Retrieves the number and Subclass of the newly saved object

- Overwrites the number and Subclass that are set by PLM clients

- Inherited interfaces – `IUpdateEventInfo` , `IObjectEventInfo`, `IEventInfo`

Delete Object

The Information object for this Event is `IDeleteEventInfo`.

- Purpose and function – `IDeleteEventInfo` is the interface for Delete Event information object. It performs the following tasks:
  - Retrieves the number and Subclass of the newly deleted object
  - Checks if the action performed is a *soft* or *hard* delete
- Inherited interfaces – `IObjectEventInfo`,`IEventInfo`

Export Object

The Information object for this Event is `IExportEventInfo`.

- Purpose and function – `IExportEventInfo` performs the following tasks:
  - Retrieves the format of the export file
  - Returns the array of objects that are exported
  - Returns the tables for exporting the object
- Inherited interfaces – `IEventInfo`

Working with General Base Event Script Objects

Create Object

The Script Event object for this Event is `ICreateScriptObj`.

- Purpose and function – The same as `ICreateEventInfo`
- Inherited interfaces – `IUpdateScriptObj`, `IBaseObjectScriptObj`, `IBaseScriptObj`
- Inherited interface purpose and function – The same as `IUpdateEventInfo`, `IObjectEventInfo` and `IEventInfo`

The following examples show using `ICreateScriptObj` and `IUpdateScriptObj`.

Example: Using ICreateScriptObj

```
// In the example, ICreateScriptObj modifies the number and class ID of
the new object.
void testICreateScriptObj(IBaseScriptObj obj)
{
       String origNumber = "";
       String newNumber = "";
       int newSubclassId =
          ItemConstrants.CLASS_DOCUMENT ; // new subclass ID of choice
       int newSubclass ;
       int subclass;
       // getNewNumber()
       origNumber = obj.getNewNumber();
       newNumber = origNumber +"new";
```

```
        // setNewNumber()
        obj.setNewNumber(newNumber);
        newNumber = obj.getNewNumber();

        // getNewSubclassId()
        subclass = obj.getNewSubclassId();

        // setNewSubclassId()
        obj.setNewSubclassId(newSubclassId);
        newSubclass = obj.getNewSubclassId();

        // log new object number and new subclass ID in to handler
        monitor
        obj.logMonitor("new object number is:" + newNumber);
        obj.logMonitor("new subclass ID is:" + newSubclass);

}
```

Example: Using IUpdateScriptObj

```
// In this example, IUpdateScriptObj gets the ID and value of Dirty
attributes, sets Dirty attribute values from context object, and
removes the value of Dirty attributes.
void testIUpdateScriptObj(IBaseScriptObj obj)
{
    String dirtyAttr = "";
    String dirtyValue = "";

    // get Attribute Ids
    int[] attrs = obj.getAttrIds();
    attrs.each {attr->

    // get Attribute value
    dirtyAttr = obj.getDirtyAttr(attr);

    // log attribute Id and value in to handler monitor
    obj.logMonitor("Dirty Attr Id :" + attr);
    obj.logMonitor(" dirty Attr value:" + dirtyAttr);
    }

    // Overwrite client value
    obj.setDirtyAttrValue(CommonConstants.ATT_PAGE_TWO_TEXT02, "set text
    value from CO");
    //Remove Dirty Attribute value
    obj.removeDirtyAttr(CommonConstants.ATT_PAGE_TWO_TEXT02);
    // get dirty attribute value after removing value from CO
    dirtyValue =
    obj.getDirtyAttrValue(CommonConstants.ATT_PAGE_TWO_TEXT02);
    //log original attribute value in to handler monitor
    obj.logMonitor("Attribute value after remove:" + dirtyValue);
}
```

Update Title Block

The Script Event object for this Event is `IUpdateTitleBlockScriptObj`.

- Purpose and function – `IUpdateTitleBlockScriptObj` checks whether this is a redline update or undo-redline update on the Title Block of the `IItem` object which has originated from the Affected Items table of `Change` object
- Inherited interfaces – `IUpdateScriptObj`, `IBaseObjectScriptObj`, `IBaseScriptObj`

Save As Object

Script Event object for this Event is `ISaveAsScriptObj`.

- Purpose and function – `ISaveAsScriptObj` retrieves the name or number for the new object.
- Inherited interfaces – `IUpdateScriptObj`, `IBaseObjectScriptObj`, `IBaseScriptObj`

Delete Object

The Information object for this Event is `IDeleteScriptObj`.

- Purpose and function – `IDeleteScriptObj` deletes the Event Script object.
- Inherited interfaces – `IBaseObjectScriptObj`, `IBaseScriptObj`

Export Object

Script Event object for this Event is `IExportScriptObj`.

- Purpose and function – `IExportScriptObj` retrieves the format for the export object.
- Inherited interfaces – `IBaseScriptObj`

Example: Using IExportScriptObj

```
In this example, IExportScriptObj retrieves information about the
object that is exported.
void testIExportScriptObj(IBaseScriptObj obj)
{
    int format;
    int[] selectedTables =
    String objects ="";
    String user = "";
    String objectNumber =
           "P00002" ; // the object number being exported
    // get current user
    obj.getCurrentUser();
    // get file format for the export.
    format = obj.getExtractedFormat();

    // get list of Object Names being exported
    objects = obj.getExtractedObjects();

    // get tables selected for export
    selectedTables = obj.getSelectedTables(objectNumber);
    // log current user, extracted format, object names and tables which
    exported in to handler monitor
       obj.logMonitor("current user is:" + user);
       obj.logMonitor("file format is:" + format);
```

```
        obj.logMonitor("Object Names are:" + objects);
        obj.logMonitor("Selected Tables are:" + selectedTables);
}
```

Working with Table and Relationship Actions

These actions include updating the supported object tables for specific business objects.

Table and Relationship Actions - Java PX

Update Table

The Information object for this Event is `IUpdateTableEventInfo`.

- Purpose and function – `IUpdateTableEventInfo` is the interface for *Update Table* and *Update Relationship*. It retrieves the Dirty table for the affected object.
- Inherited interfaces – `IObjectEventInfo` and `IEventInfo`. See Base Information Objects - Java PX on page 409.
- Related interfaces:
  - `IEventDirtyFile` is the interface for a Dirty file associated with `IEventDirtyRowFileUpdate` or `IFileEventInfo`. It represents a single row in a file table and retrieves the checkout date of the Dirty file.
  - `IEventDirtyTable` is the interface for a Dirty table associated with `IUpdateTableEventInfo` or `IEventDirtyRow`. The Dirty table contains a collection of modified rows. It provides access to transient table information for modified tables.
  - `IEventDirtyCell` is the interface for a Dirty cell associated with `IEventDirtyRowUpdate` or `IUpdateEventInfo`. Represents a single cell in a row It returns the attribute identifier corresponding to this Dirty cell.
  - `IEventDirtyRowFileUpdate` retrieves the Dirty file. It is the interface for a Dirty row used to perform Dirty Row Actions (Add file, Replace file).
  - `IEventDirtyRowUpdate` retrieves the Dirty row for which the update occurs. This interface is used by the update on all tables except the attachment table.

Note You can find information on "Dirty" objects in the SDK samples Documentation folder. To access this folder, see the Note in Client-Side Components on page 2.

The following examples show using `IEventDirtyTable` and `IEventDirtyRowUpdate` on Item BOM table.

Example: Using IEventDirtyTable

```
private void testIEventDirtyTable(IAgileSession session,
IEventDirtyTable table, IDataObject obj, int evtTriggerType) throws
Exception {
    //getTableId()
    String tableName = getTableName(obj, table);

    // size()
    int size = table.size();
```

```
    //iterator()
    Iterator it = table.iterator();
    while (it.hasNext()){
           IEventDirtyRow row = (IEventDirtyRow)it.next();
           if(row.getAction()!=EventConstants.DIRTY_ROW_ACTION_ADD_FILE&
           &row.getAction()!=EventConstants.DIRTY_ROW_ACTION_REPLACE_FIL
           E)
               //user defined method//
               testIEventDirtyRowUpdateCommon(session, row, obj,
               evtTriggerType);
               else
               testIEventDirtyRowFileUpdate(row);// user defined method
    }

}
```

Example: IEventDirtyRowUpdate on Item BOM table

```
private void
testIEventDirtyRowUpdate_ItemBOM_Update(IEventDirtyRowUpdate row,
IDataObject obj) throws Exception {
IEventDirtyCell[] cells1 = row.getCells();
readCells(cells1);  // user defined method

    /* setCell() - Override
    * List01 ==> c
    * MultiText30 ==> setCell() on update
    * Text01 ==> setCell()on update
    * Numeric01 ==> 888.66
    * BOM Notes ==> setCell() from CO
    */
    //List01
    IAttribute attrList =
    obj.getAgileClass().getAttribute(ItemConstants.ATT_BOM_BOM_LIST01);

    IAgileList list =
    (IAgileList)attrList.getAvailableValues();

    list.setSelection(new Object[]{"c"});
    row.setCell(ItemConstants.ATT_BOM_BOM_LIST01, list);
    //MultiText30
    row.setCell(ItemConstants.ATT_BOM_BOM_MULTITEXT30, "setCell() MT30
    update");
    //Text01
    row.setCell(ItemConstants.ATT_BOM_BOM_TEXT01, "setCell() T01
    update");
    // Numeric01 row.setCell(ItemConstants.ATT_BOM_BOM_NUMERIC01,
    888.66);
// BOM Notes row.setCell(ItemConstants.ATT_BOM_BOM_NOTES, "setCell()
Notes update");

/*
* setCell() - New
* List02 ==> a
* Text02 ==> setCell() on update
```

```
*/

    // List02
    row.setCell(ItemConstants.ATT_BOM_BOM_LIST02, list);
    //Text02
    row.setCell(ItemConstants.ATT_BOM_BOM_TEXT02, "setCell() T02
    update");
    // Qty row.setCell(ItemConstants.ATT_BOM_QTY, new Integer(2));
    //removeCell() ==> Date01 & Ref Des
    row.removeCell(ItemConstants.ATT_BOM_BOM_DATE01);
    row.removeCell(ItemConstants.ATT_BOM_FIND_NUM);

    // getCell()
    IEventDirtyCell cell =
    row.getCell(ItemConstants.ATT_BOM_BOM_LIST02);
}
```

Update Relationship

See Update Table in Table and Relationship Actions - Java PX on page 418.

Table and Relationship Actions - Script PX

Update Table

The Script Event object for this Event is `IUpdateTableScriptObj`.

- Purpose and function – `IUpdateTableScriptOb` is the interface for Update Table and Update Relationship. It retrieves the dirty table for the affected object.
- Inherited interfaces – `IBaseObjectScriptObj`. See Base Event Script Objects - Script PX on page 410 .
- Related interfaces:
  - `IEventDirtyRow` is the base interface for a dirty row associated with `IUpdateTableScriptObj`. It represents a single row in a table. There is an action associated with the row.
  - `IEventDirtyRowFileUpdate` retrieves the dirty file row. It is the interface for a dirty file row used to perform Dirty File Row Actions (Add file, Replace file).
  - `IEventDirtyRowUpdate` retrieves the Dirty row. It is the interface for a Dirty row used to perform Dirty Row Actions (Add, Delete, Update, Redline Add, Redline Delete, Redline Update, Undo Redline.)
  - `IEventDirtyFile` is the interface for a Dirty file associated with `IEventDirtyRowFileUpdate` or `IFileEventInfo`. It represents a single row in a file table and retrieves the checkout date of the Dirty file.

The following examples show using `IEventDirtyTable` and `IEventDirtyRowUpdate` on Item BOM table on Item BOM table.

Example: Using IEventDirtyTable

```
// IUpdateTableScriptObj: get table id and Dirty row ids
    void testIUpdateTableScriptObj(IBaseScriptObj obj)
{
```

```
    //getTableId()
    int table_id= obj.getTableId();
    obj.logMonitor("Table Id" + table_id);

    // getDirtyRowIds()
    rowIDs =obj.getDirtyRowIds();
    rowIDs.each{rID->   //loop through Dirty rows
    obj.logMonitor("rowID is " + rID);

    //getDirtyRow()
    row = obj.getDirtyRow(rID);

    // getAction
    action = row.getAction();
    obj.logMonitor ("Action" + action);

    if ((action == EventConstants.DIRTY_ROW_ACTION_ADD_FILE) || (action
    == EventConstants.DIRTY_ROW_ACTION_REPLACE_FILE))
       testIEventDirtyRowFileUpdate(obj,row,rID); // user method

    else
       testIEventDirtyRowUpdate(obj,row,rID); // user method
}
```

Example: Using IEventDirtyRowUpdate

```
//This Item BOM table example gets the object number and action for a
single row associate with IUpdateTableScriptObj
void testIEventDirtyRowUpdate(IBaseScriptObj obj,IEventDirtyRow row,
int rID )
{

    //getRowId()
    int rid = row.getRowId();

    // getObjectNumber()
    int rnumber = row.getObjectNumber();
    obj.logMonitor("row object" + rnumber);

    // getDirtyRow
    dirtyRow = obj.getDirtyRow(rID);

    //from client, update following attributes in Bom table
    // find number,Bom notes, Multi Text01,List01,date01, Text01,
    numeric0
    //getDirtyAttrIds()
       dirtyAttrs = dirtyRow.getDirtyAttrIds();
       def sort_attrs = dirtyAttrs as List;
       sort_attrs.sort(); //sort attribute Ids

    // getDirtyAttrValue()
       sort_attrs.each {dirtyAttr->
       attrValue =  dirtyRow.getDirtyAttrValue(dirtyAttr);
       obj.logMonitor('***'+"attribute value" +'=' + attrValue);
       }

    // array of vlues to overwrite attributes from Context object
    //[find number,Bom notes, Multi Text01,List01,date01, Text01,
    numeric01]
```

```
        set_dirty_value = ["5","Bom Note","Multi text 03","e","2009-01-30
        08:00:00","Text 01","224466"];
    //setDirtyAttrValue , overwrite attribute values
        int indexy =0;
        sort_attrs.each{att->
        dirtyRow.setDirtyAttrValue(att , set_dirty_value[indexy++]);
        }

    // getDirtyAttrValue() after overwrite in CO
        int indexB=0;
        sort_attrs.each {dirtyAttr->
        attrValue2 =  dirtyRow.getDirtyAttrValue(dirtyAttr);
        all_attrValue2[indexB++]= attrValue2;
        }

    // removetDirtyAttr() remove dirty attributes and rollback changes

        sort_attrs.each {dirtyAttr->
        dirtyRow.removetDirtyAttr(dirtyAttr);
    }
}
```

Update Relationship

See Update Table in Table and Relationship Actions - Java PX on page 418.

Working with Workflow Object Actions

These are Worflow-related actions such as, Change Status for Workflow, Change Approvers or Observers for Workflow, and the change the status of Product Cost Management's Sourcing Project.

Workflow Object Actions - Java PX

Change Status for Workflow

The Information object for this Event is `IWFChangeStatusEventInfo`.

- Purpose and function – `IWFChangeStatusEventInfo` retrieves changes in Workflow status of the object and assigned notifiers and checks whether the "auto-promote" flag is set or not set.
- Inherited interfaces – `IRoutingSlipEventInfo` , `IRoutableObjectEventInfo`, `IObjectEventInfo, IEventInfo`
- Inherited interfaces purpose and function:
  - `IRoutingSlipEventInfo` – Is the inherited interface for all Event Information objects that contain the Routing slip Object and provides methods to set/get approvers, observers, comments, and urgent flags
  - `IRoutableObjectEventInfo` – Is the inherited interface for all Event Information objects related to workflow actions and provides methods such as get/set approvers or observers

The following examples show using `IWFChangeStatusEventInfo`, `IRoutingSlipEventInfo`, and `IRoutableObjectEventInfo`.

Example: Using IWFChangeStatusEventInfo

```
private void testIWFChangeStatusEventInfo(IAgileSession session,
IEventInfo req) throws APIException {
    IWFChangeStatusEventInfo info = (IWFChangeStatusEventInfo)req;
    //getNotifiers()
    IDataObject[] notifiers = info.getNotifiers();

    // isAutoPromote()
    boolean isAutoPromote = info.isAutoPromote();
    // setNotifiers()
    IUser user = getUser(session, "yvonnec");
    IUserGroup ug = getUserGroup(session, "SOA");
    Collection col = new ArrayList();
    col.add(user);
    col.add(ug);
    info.setNotifiers(col);

    // getFromStatus()
    IStatus fromStatus = info.getFromStatus();

    // getToStatus()
    IStatus toStatus = info.getToStatus();
}
```

Example: Using IRoutingSlipEventInfo

```
private void testIRoutingSlipEventInfo(IAgileSession session,
IEventInfo req) throws APIException {
    IRoutingSlipEventInfo info = (IRoutingSlipEventInfo)req;
    Collection colAppvrs = new ArrayList();
    Collection colObsvrs = new ArrayList();
    Collection colNewAppvrs = new ArrayList();
    Collection colNewObsvrs = new ArrayList();
    String newComments = "Override in context object.";

    //getApprovers()
    IDataObject[] approvers = info.getApprovers();
    String approverList = arrayToString(approvers);

    //getObservers()
    IDataObject[] observers = info.getObservers();

    //getComments()
    String comments = info.getComments();

    //isUrgent()
    boolean isUrgent = info.isUrgent();

    //setApprovers()
    IUser user1 = getUser(session, "badriv");
    IUser user2 = getUser(session, "albertl");
    IUser user3 = getUser(session, "brucec");
    IUserGroup ug1 = getUserGroup(session, "SOA");

    colAppvrs.add(user1);
    colAppvrs.add(user2);
    colObsvrs.add(user3);
    colObsvrs.add(ug1);
```

```
    info.setApprovers(colAppvrs);
    IDataObject[] newApprovers = info.getApprovers();

    //setObservers()
    info.setObservers(colObsvrs);
    IDataObject[] newObservers = info.getObservers();

    //setComments()
    info.setComments(newComments);
    String latestComments = info.getComments();

    //setUrgent()
    boolean newUrgent = getOppositeBoolean(isUrgent);

    info.setUrgent(newUrgent);
    boolean latestUrgent = info.isUrgent();
}
```

Example: Using IRoutableObjectEventInfo

```
private void testIRoutableObjectEventInfo(IEventInfo req) throws
APIException {
       IRoutableObjectEventInfo info = (IRoutableObjectEventInfo)req;
       //getWorkFlow()
       IWorkflow wf = info.getWorkFlow();
}
```

Approve for Workflow

The Information object for this Event is `ISignOffEventInfo`.

- Purpose and function – `ISignOffEventInfo` is the interface for Approve for Workflow and Reject for Workflow. It returns and overwrites information about status, approvers (users), approver groups, and checks the status of the signoff flag.
- Inherited interfaces – `IRoutableObjectCommentEventInfo`, `IRoutableObjectEventInfo`, `IObjectEventInfo, IEventInfo`
- Inherited interfaces purpose and function – `IRoutableObjectCommentEventInfo` is the interface for Comment for Workflow. It retrieves and overwrites data entered for Comments for Workflow, Approve for Workflow, and Reject for Workflow Events.

Reject for Workflow

See Approve for Workflow.

Escalation for Workflow

The Information object for this Event is `IEscalationEventInfo`.

- Purpose and function – `IEscalationEventInfo` retrieves the following information about the Escalation for Workflow Event:
  - Sign-off user, escalated to users, escalation period, and status of the workflow
- Inherited interfaces – `IRoutableObjectEventInfo`, `IObjectEventInfo, IEventInfo`
- Inherited interfaces purpose and function – See Approve for Workflow.

Reminder for Workflow

The Information object for this Event is `IReminderEventInfo`.

- Purpose and function – `IReminderEventInfo` returns notifiers assigned to the Workflow reminder.
- Inherited interfaces – `IRoutableObjectEventInfo`, `IObjectEventInfo, IEventInfo`

Audit for Workflow

The Information object for this Event is `IAuditStatusEventInfo.`

- Purpose and function – `IAuditStatusEventInfo` retrieves the type of Workflow Audit Action that is performed.
- Inherited interfaces – `IAuditResultEventInfo, IRoutableObjectEventInfo, IObjectEventInfo, IEventInfo`
- Inherited interfaces purpose and function – `IAuditResultEventInfo` is the interface for Audit for Workflow and Promotion Failure for Workflow. It retrieves error messages for the audit or promotion failure.

Promotion Failure for Workflow

See Audit for Workflow.

Comment for Workflow

The Information object for this Event is `IRoutableObjectCommentEventInfo`. See Approve for Workflow.

Change Approvers or Observers for Workflow

The Information object for this Event is `IChangeAppObserverEventInfo`.

- Purpose and function – `IChangeAppObserverEventInfo` retrieves the action type, status, and applied to status of the Change Approvers or Observers for the Workflow Event.
- change action by the approver or observer on the Workflow state.
- Inherited interfaces – See `IRoutingSlipEventInfo`.

Workflow Object Actions - Script PX

Change Status for Workflow

The Information object for this Event is `IChangeStatusScriptObj`.

- Purpose and function – `IChangeStatusScriptObj` retrieves changes in Workflow status of the object, the assigned notifiers, and checks whether the "auto-promote" flag is set or not set.
- Inherited interfaces – `IRoutingSlipScriptObj` , `IRoutableScriptObj`, `IBaseObjectScriptObj, IBaseScriptObj`
- Inherited interfaces purpose and function:

- `IRoutingSlipScriptObj` – Inherited interface for all Event Script objects that contain the Routing slip Object
- `IRoutablecriptObj` – Inherited interface for all Event Script objects related to workflow actions

The following examples show using `IChangeStatusScriptObj`, `IRoutingSlipScriptObj`, and `IRoutableObjectScriptObj`.

Example: Using IChangeStatusScriptObj

```
IChangeStatusScriptObj: Change Status for Workflow Event script object.
void testIChangeStatusScriptObj(IBaseScriptObj obj)
{
    // getFromStatus()
    String current_status = obj.getFromStatus();

    // getToStatus()
    String next_status = obj.getToStatus();

    // getNotifiers()
    String notifiers =obj.getNotifiers();

    // setNotifiers()
     obj.setNotifiers(["admin", "demo1","weiz", "demo5"]);
     // isAutoPromote()
    boolean  auto = obj.isAutoPromote();
}
```

Example: Using IRoutingSlipScriptObj

```
void testIRoutingSlipScriptObj(IBaseScriptObj obj)
{
    // getCurrentUser()
    String user= obj.getCurrentUser();

    // getObservers()
    def observers = obj.getObservers();

    // getApprovers()
    def approvers = obj.getApprovers();

    // getComments()
    String  comments = obj.getComments();

    // isUrgent()
    boolean  flag = obj.isUrgent();

     // setApprovers()
     obj.setApprovers(["admin", "demo1"]);

     // setObservers()
     obj.setObservers(["demo6", "demo5"]);

     // setComments()
     obj.setComments(" new_comments");
     boolean new_urgent= true;

     // setUrgent()
     obj.setUrgent(new_urgent);
}
```

Example: Using IRoutableScriptObj

```
IRoutableScriptObj : get workflow
void testIRoutableScriptObj(IBaseScriptObj obj)
{

// getWorkflow()
String wf = obj.getWorkflow();
}
```

Approve for Workflow

The Information object for this Event is `ISignOffScriptObj`.

- Purpose and function – `ISignOffScriptObj` is the interface for Approve for Workflow and Reject for Workflow. It returns information about status, approvers (users), approver groups, and checks the status of the signoff flag.
- Inherited interfaces – `IRoutableObjectCommentScriptObj`, `IRoutableScriptObj`, `IBaseObjectScriptObj, IBaseScriptObj`
- Inherited interfaces purpose and function – `IRoutableObjectCommentScriptObj` – Interface for Comment for Workflow. It retrieves comments entered for the Routable object.

Reject for Workflow

See Approve for Workflow.

Escalation for Workflow

The Information object for this Event is `IEscalationScriptObj`.

- Purpose and function – `IEscalationScriptObj` retrieves names of users to whom the Workflow is escalated to.
- Inherited interfaces – `IRoutableScriptObj`, `IBaseObjectScriptObj, IBaseScriptObj`
- Inherited interfaces purpose and function – See Approve for Workflow

Reminder for Workflow

The Information object for this Event is `IReminderScriptObj`.

- Purpose and function – `IReminderScriptObj` returns notifiers assigned to the reminder period and Workflow reminder.
- Inherited interfaces – `IRoutableScriptObj, IBaseObjectScriptObj, IBaseScriptObj`

Audit for Workflow

The Information object for this Event is `IAuditStatusScriptObj.`

- Purpose and function – `IAuditStatusScriptObj` retrieves the type of Workflow Audit Action that is performed.
- Inherited interfaces – `IAuditResultScriptObj, IRoutableScriptObj, IObjectScriptObj, IBaseScriptObj`

- Inherited interfaces purpose and function – `IAuditResultEventInfo` is the interface for Audit for Workflow and Promotion Failure for Workflow. It retrieves error or warning messages and Workflow status information for the audit or promotion failure.

Promotion Failure for Workflow

See Audit for Workflow.

Comment for Workflow

The Information object for this Event is `IRoutableObjectCommentScriptObj`. See Approve for Workflow.

Change Approvers or Observers for Workflow

The Information object for this Event is `IChangeAppObserverScriptObj`.

- Purpose and function – `IChangeAppObserverScriptObj` retrieves the action type, status, and applied to status of the Change Approvers or Observers for the Workflow Event.
- Inherited interfaces – See `IRoutingSlipScriptObj`.

Working with Specific Object-Based Actions

These actions support incorporating and unincorprating the Item object in PLM's Production Collaboration solution and change status for Sourcing projects.

Specific Object-Based Actions - Java PX

Incorporate Item and Unincorporate Item

The Information object for this Event is `IObjectEventInfo`. This interface and applicable inherited interfaces are documented in General Object Actions - Java PX on page 412.

Change Status for Sourcing Project

The Information object for this Event is `ISourcingObjectChangeStatusEventInfo`.

- Purpose and function – `ISourcingObjectChangeStatusEventInfo` retrieves the type of action that is performed on the Sourcing object.
- Inherited interfaces – `IObjectEventInfo` and `IEventInfo`

Specific Object-Based Actions - Script PX

Incorporate Item and Unincorporate Item

The Script Event object for this Event is `IBaseObjectScriptObj`. This interface and applicable inherited interfaces are documented in General Object Actions - Script PX on page 415.

Change Status for Sourcing Project

The Information object for this Event is `ISourcingObjectChangeStatusScriptObj`.

- Purpose and function – `ISourcingObjectChangeStatusScriptObj` retrieves the type of action that is performed on the Sourcing object.
- Inherited interfaces – `IBaseObjectScriptObj` and `IBaseScriptObj`

Working with Files and Attachments Objects Actions

These are the file-related actions such as check out file, check in file, and purging version files.

Files and Attachments Objects Actions - Java PX

Get File, Check Out Files, Check In Files, Cancel Check Out Files

The Information object for these Events is `IFileEventInfo`.

- Purpose and function – `IFileEventInfo` retrieves the Dirty files for the selected file attachments.
- Inherited interfaces – `IObjectEventInfo` and `IEventInfo`
- Related interfaces – `IEventDirtyFile` represents a single row in a file table, the interface for `IEventDirtyRowFileUpdate` or `IFileEventIno`

The following example show using `IFileEventInfo`.

Example: Using IFileEventInfo

```
private void testIFileEventInfo(IEventInfo req) throws APIException {
    IFileEventInfo info = (IFileEventInfo)req;
    StringBuffer buffer = new StringBuffer();

    // getFiles()
    IEventDirtyFile[] files = info.getFiles();
    if(files!=null){
        int length = files.length;
        if (length!=0){
            for (int i=0;i<length;i++){
            String fileName = files[i].getFilename();
            String msg = i + ": " + fileName;
            System.out.println(msg);
            }
    }else {//rows == null
        throw new APIException (new Exception("getFiles() returns
        null"));
    }

    // Test IEventDirtyFile
    testIEventDirtyFile(files);
}
```

Purge File Version

The Information object for this Event is `IPurgeFileEventInfo`.

- Purpose and function – `IPurgeFileEventInfo` retrieves the version for the purged file.
- Inherited interfaces – `IObjectEventInfo` and `IEventInfo`

Files and Attachments Objects Actions - Script PX

Get File, Check Out Files, Check In Files, Cancel Check Out Files

The Event Script object for these Events is `IFileEventScriptObj`.

- Purpose and function – `IFileEventScriptObj` retrieves the Dirty files for the selected file attachments.
- Inherited interfaces – `IBaseObjectScriptObj` and `IBaseScriptObj`

Purge File Version

The Event Script object for this Event is `IPurgeFileScriptObj`.

- Purpose and function – `IPurgeFileScriptObj` retrieves the version for the purged file
- Inherited interfaces – `IBaseObjectScriptObj` and `IBaseScriptObj`

The following example show using `IFileEventScriptObj`.

Example: Using IFileEventScriptObj

```
// FileEventScriptObj: get file for file actions related events
void testIFileEventScriptObj(IBaseScriptObj obj,IEventDirtyRow row)
{
    // getFile()
    file =  row.getFile();
    testIEventDirtyFile(obj,file);
}
```

Example: Using IEventDirtyFile

```
IEventDirtyFile: get file information for Dirty file associated with
IEventDirtyRowFileUpdate
void testIEventDirtyFile(IBaseScriptObj obj,IEventDirtyfile file)
{

    // getFilename()
    String file_name = file.getFilename();

    // getSize()
    int file_size = file.getSize();

    // getType()
    file_type = file.getType();

    // getFileFolderNumber()
    file_folder_num = file.getFileFolderNumber();
```

```
// getFileFolderVersionNumber()
file_folder_ver = file.getFileFolderVersionNumber();
//getCheckoutUser()
user = file.getCheckoutUser();

// getCheckoutDate()
date = file.getCheckoutDate();

    obj.logMonitor("file name is:" + file_name);
    obj.logMonitor("file size is:" + file_size);
    obj.logMonitor("file type is:" + file_type);
    obj.logMonitor("file folder number is:" + file_folder_num);
    obj.logMonitor("file folder version number is:" +
    file_folder_ver);
    obj.logMonitor("checkout user is:" + user);
    obj.logMonitor("checkout date is:" + date);
```

Working with Product Governance and Compliance Actions

This action supports checking compliance of PG&C objects.

Product Governance and Compliance Actions - Java PX

Compliance Rollup On Object

The Information object for this Event is `IObjectEventInfo`. This interface and applicable inherited interfaces are documented in General Object Actions - Java PX on page 412.

Product Governance and Compliance Actions - Script PX

Compliance Rollup On Object

The Information object for this Event is `IBaseScriptObj`. This interface and applicable inherited interfaces are documented in General Object Actions - Script PX on page 415.

Working with Miscellaneous Object Actions

These are the file-related actions such as check out file, check in file, and purging version files

Miscellaneous Object Actions - Java PX

Transfer Authority

The Information object for this Event is `ITransferAuthorityEventInfo`.

- Purpose and function – `ITransferAuthorityEventInfo` contains the data belonging to Transfer Authority action and retrieves and overwrites data set by the PLM client for the Transfer Authority action.
- Inherited interfaces – See `IExportEventInfo`.

Miscellaneous Object Actions - Script PX

Transfer Authority

The Information object for this Event is `ITransferAuthorityScriptObj`.

- Purpose and function – `ITransferAuthorityScriptObj` retrieves and overwrites data set by the PLM client for the Transfer Authority action.
- Inherited interfaces – `IBaseScriptObj`

Working with Event Integration Points in PLM Clients

Events can be invoked from Agile PLM clients' Extend Actions Menu, Extend Tools Menu, and Scheduled Event described below.

Event Integration Points - Java PX

Extend Actions Menu

The Information object for this Event is `IObjectEventInfo`. This interface and applicable inherited interfaces are documented in General Object Actions - Java PX on page 412.

Extend Tools Menu

The Information object for this Event is `IEventInfo`. This interface and applicable inherited interfaces are documented in General Object Actions - Java PX on page 412.

Scheduled Event

The Information object for this Event is `IEventInfo`. This interface and applicable inherited interfaces are documented in General Object Actions - Java PX on page 412.

Event Integration Points - Script PX

Extend Actions Menu

The Information object for this Event is `IBaseScriptObj`. This interface and applicable inherited interfaces are documented in General Object Actions - Script PX on page 415.

Extend Tools Menu

The Information object for this Event is `IBaseObjectScriptObj`. This interface and applicable inherited interfaces are documented in General Object Actions - Script PX on page 415.

Scheduled Event

The Information object for this Event is `IBaseScriptObj`. This interface and applicable inherited interfaces are documented in General Object Actions - Script PX on page 415.

Guidelines for Java PX and Script PX Handlers

These comments and recommendations are intended to ensure the proper development and implementation of your Java PXs and Script PXs. They include:

- Information that you need from the Agile PLM Administrator to code the handler and information that you must convey to the Administrator to configure and implement the Event
- Information to ensure proper handling of Events
- Information to test your PXs

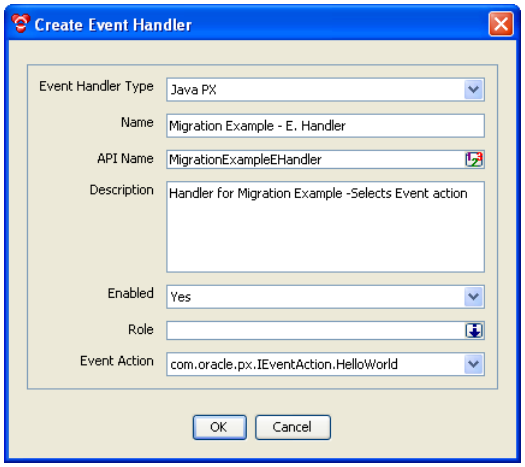
Working with Agile PLM Administrator

The Agile PLM Administrator will convey the necessary information to determine the Event type, execution mode and trigger type. Once you have developed the Java PX or Script PX handlers, you must inform the Administrator about the Event subscriber. This information includes Event type,execution mode, trigger type, order, and applicable error handling rule.

In addition to information about Event subscriber, the Administrator needs the following specific information to configure the Event Handler.

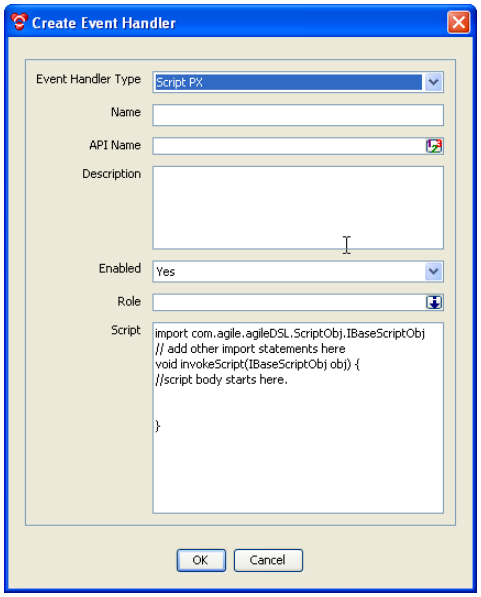
- Java PX Handlers – Java PXs are deployed on the server, for procedures, see Packaging and Deploying a Custom Action on page 349. Once deployed, the Administrator locates it in Java Client by selecting Admin > Settings > System Settings > Event Management > Event Handlers > New. The Create Event Handler dialog box appear. In Event Handler Type field select > Java PX > Event Action. The Administrator needs to know the name that appears in the Event Action field and proceed to configure the Handler.

Figure 32: Create event Handler dialog



- Script PX Handlers – The script PX code is a text file. You must send the script text file to the PLM Administrator. The Administrator will then paste it in the Script field of Create Event Handler dialog and proceed to configure the Subscriber. This field is accessed from Java Client by selecting Administrator > Event Management > Event Handlers > Create Event Handler > Script PX > Script.

Figure 33: Create Script PX



Testing Event Java PX and Event Script PX

If the Agile PLM Administrator uses your Handlers to configure the Event Subscriber, you should coordinate testing the PXs with the Administrator. On the other hand, if you do the configurations, then use the following information for this purpose.

Invoke the new Java PX or Script PX to ensure the action specified in the Handler (code) executes properly.

If the PX is configured to be invoked by a user action from the Tools menu, you can test it in Web Client or Java Client as follows:

- In Web Client tool bar select the button > <Event_name>.
- In Java Client tool bar select the Process Extensions button or Tools > Process Extensions > <Event_name>.

Triggering Guidelines for Java PX, Script PX, and Notification Handlers

Use the following guidelines when developing Handlers to ensure proper triggering of the special instances of these Event types.

General Object Actions

Create Object Event and SaveAs Event

- Web Client and Java Client behavior – This action will trigger one Create Object Event (or SaveAs Event) and its Context object will contain all Dirty attributes for Page 1, Page 2 and Page 3.
- SDK behavior – This action will trigger one Create object Event (or SaveAs Event). In addition, depending on table attributes that are updated (Page 1, Page 2 and Page 3), one to three Update Title Block Events are triggered. The context object will contain all Dirty attributes belonging to that table.

Update Title Block Event

Web Client, Java Client, and SDK behavior – Depending on table attributes that are updated (Page 1, Page 2 and Page 3), one to three Update Title Block Events are triggered, one for each table. The context object will contain all Dirty attributes belonging to that table.

Update Table Event

When you add, modify, or delete more than one row, this action has the following Event triggering behavior:

- One Event is triggered and the Context object will contain all applicable rows. The only exceptions are in Java Client:
  - Relationships Table– One Event is triggered for each applicable row and the Context Object will contain the applicable row
  - Attachment Table – For Update and Delete Actions, one Event is triggered for each applicable row and the Context Object will contain the applicable row

Workflow Actions

Promotion Failure for Workflow Event

This Event is triggered by the following actions:

- Failure of Change Status triggered through Relationship
- Autopromotion Failure – Autopromotion Failure is invoked by the following object actions when autopromotion conditions are not met after the completion of these actions. In addition, each triggered Event is tracked in the object History.
  - Change Status
  - Sign off the change
  - Remove Approvers
  - Update Cover Page, Page 2, and Page 3 attributes
  - Edit Affected Item Table, Relationship, File Attachment Tab

    Note Only editing will trigger "autopromote."
  - Cancel checkout attachment
  - Checkin attachment

Create Automatic Transfer Object Action (ATO)

ATO creation is enabled when ECO status is changed from Pending to Submitted. The sequence of Events that are triggered when the ATO is created is:

1. Pre and Post Event Subscribers are created for:
   - ECO Change status
   - ATO Create
   - ATO Change Status
2. An ECO is created
3. Workflow is assigned and status is changed to submitted

Files and Attachments Actions

Check In File Event

This action will trigger the following Events:

- Check In File Event on File Folder objects – This Event is triggered when the Check In file action is performed on File Folder objects
- Check In File Event and Update Table Event on Business objects – These Event are triggered when the Check In file action is performed on Business objects

> **Note** Update Table Event on Business objects is triggered after Check In File Event is triggered. Also, The Event is only triggered if the folder version of the attachment is not set to [LATEST]. In Web Client one Event is triggered for all selected rows. In Java Client, one Event is triggered for each selected row.

Check Out File Action

This action will trigger the following Events:

- Check Out Files Event and Get File Event on File Folder objects – These Events are triggered when the Check Out file action is performed on File Folder objects
- Check Out Files Event and Get File Event on Business objects – These Events are triggered when the Check Out file action is performed on Business objects in Java Client,
  - If Download files in one ZIP file option is selected, one Get File Event is triggered and each row triggered a Checkout Event
  - If Download files in one ZIP file option is not selected, each row triggers one checkout Event and each file triggers a Get File event

> **Note** In Web Client one Event is triggered for all selected rows. In Java Client, one Event is triggered for each selected row.

Cancel Check Out File Event

This action will trigger the following Events:

- Cancel Check Out Files Event on File Folder objects – This Event is triggered when the Cancel Check Out file action is performed on File Folder objects
- Cancel Check Out Files Event on Business objects – This Event is triggered when the Cancel Check Out file action is performed on Business objects

> **Note** In Web Client one Event is triggered for all selected rows. In Java Client, one Event is triggered for each selected row.

Get File Event

This Event is triggered by:

- Clicking the Get button on the Attachments table of Business objects or Files table of File Folder objects
- Clicking Filename on the Attachments table of the object

> **Note** If the file is viewable, Agile Viewer is launched and no event will be triggered.

- Check Out file action

> **Note** In Java Client, if users choose to download files individually instead of downloading the .ZIP file, one Event is triggered for each selected File.

Event FAQs

This section answers common questions about the Event framework and Java and Script process extensions.

What are the differences between Custom process extensions and Java process extensions (Java PX)?

Similar to Custom process extensions, Java PXs also extend the functionality of Agile PLM clients through custom actions. This is done by implementing the `IEventAction` interface in `com.agile.px` in Event framework . Process extensions can be used to connect the Agile PLM server and Agile PLM users to external systems. In addition, Java PXs contain Event COntext objects which provide more information than Custom PXs.

Can I use Agile's Java API within a Java process extension program?

Yes. You can use Agile's Java API and other external Java APIs as you did with Custom PXs.

Do Java PXs have special security requirements?

No, similar to Custom PXs.

How are roles and privileges defined for Java PXs/Script PXs?

By default, a custom action (a Handler) uses the roles and privileges of the current user. However, if you want to configure a custom action to have expanded privileges, you can specify the roles required for the Handler in the Java Client. When the Handler is executed, the roles and privileges specified for the Handler override those of the current user. Once the Handler is completed, the client reverts to the user's original roles and privileges.

Do user assigned roles override roles assigned to a Java or Script PX during configuration?

No. Roles assigned to a Java or Script PX override the user's original roles. Thus all actions that occurred inside the PX Handler are subject to privileges based on these roles. However, the access of the event context object doesn't need the privilege check, including the `getXXX` and `setXXX` method calls.

How do I configure and deploy a Java process extension?

Similar to Custom PXs, place the JAR file(s) for a process extension in the `agile_home/integration/sdk/extensions` folder on the application server. Included with the JAR file(s) should be a file named `com.agile.px.IEventAction` in the META-INF/services directory. The contents of these files are the fully qualified Java class names, one class per line, for an Event action.

After deploying a Java PX on the application server, how do I enable it?

Once Java PX code is deployed, you can configure them for use within Agile PLM in Java Client by selecting Admin > Settings > System Settings > Event Management > Event Handlers > New > Create Event Handler > Java PX > Event Action.

After I've deployed JAR file(s) for a Java PX Handler, do I need to update the application server classpath?

No. The classpath is updated automatically by a special-purpose classloader. The classloader extends the application server classpath with any classes located in agile_home/integration/sdk/extensions (or the location specified for the sdk.extensions property in the agile.properties file).

Which Custom PXs can I migrate to Event framework?

The Event framework supports migrating only Custom Action PXs. These are PXs that are initiated from the Actions Menu, Tools Menu, and Change Status for Workflow.

The corresponding Event types are: Extend Actions Menu. Extend Tools Menu, and Change Status for Workflow.

What error handling rules the developer/user must specify?

For Synchronous Handler, the user must specify the error handling rule to determine how the system reacts if it encounters an error while processing this subscription. The error handling rule only applies to synchronous Handlers. It supports the following two choices, Stop and Continue.

Agile PLM stops any further event processing, and then returns to the originator who raises the event. All remaining synchronous subscribers are not called.

In the case of pre-event, upon receiving the error from the subscriber, the originator simply throws the error to the Client that initiates the action and the original action is not performed. The system

may also rollback changes made by Handlers. However, whether the transaction can be rolled back or not depends on the Handler type. If it is a Java PX Handler, no transaction rollback is performed since the Handler is a SDK program which has its own transaction.

Do I need to deploy script PXs?

No, Scripts are pasted to the editor in the Script Handler and Agile PLM will store the code in the database. Consequently, programs are delivered in plain text files and not in object code.

Can I send Notifications using Event Handler?

Yes, you can send a Notification from a Java PX and a Script PX.

When should I use scripting?

Use Scripts for prototyping, simple operations, and test driven development.

Do I need to compile my Script code?

No, Script code is validated for syntax errors when you save it in the Handler and will be compiled when the Event is triggered.

What are Dirty files and related methods and interfaces?

They are documented in Javadoc generated SDK documentation in SDK samples folder on Oracle® E-Delivery Web site (http://edelivery.oracle.com/) under `IEventDirtyRowFileUpdate`.

Can a single action trigger multiple Events?

Yes, for some actions such as Update Multiple Attachment Rows, a single action will trigger multiple Events. In addition, if there are Subscriber for Pre and Post trigger types, then the order in which the Subscribers are invoked can vary depending on the PLM Client, the action, and the object type. For example, if you are deleting three rows from an Attachment Table with one Subscriber for the Pre trigger and one Subscriber for the Post trigger, the behavior in PLM Clients are as follows:

- Web Client – A single Update Table Event is triggered
- Java Client – Three Update Table Events are triggered and one Event for each deleted row as shown below:
  - For Changes, Items, TransferOrders, MFR Parts, Suppliers, Sites, Customers, and Package objects, the sequence of the Event Subscribers that are invoked is:
    - Pre (handler from the pre trigger subscriber - for the first row)
    - Pre (handler from the pre trigger subscriber - for the second row)
    - Pre (handler from the pre trigger subscriber - for the third row)
    - Post (handler from the post trigger subscriber - for the first row)
    - Post (handler from the post trigger subscriber - for the second row)
    - Post (handler from the post trigger subscriber - for the third row)
  - For Product Service Requests, Quality Change Requests, MFRs, Users, User Groups, the sequence is: Pre - Post - Pre - Post - Pre - Post

Best practices?

- What should we avoid in Handlers for Pre-Event?

  Use Pre-Event mainly for validation. Although you can modify the Context object, you should avoid direct object updates using SDK calls. (Sometimes, using SDK calls in Pre-Event may cause object version mismatch)

- Subscriber Ordering?

  Sometimes, for the same event type, you can have the Event Subscribers at Base Class level, Class level, and Subclass level. For example, you may have a number of Event Subscribers for Create Object Event Type: one at Item Base Class level, one at the Part Class level, and one at the Part Subclass level. If you prefer to execute the Subscribers based on Class hierarchy, it is recommended to allocate an order range for the Base Class, each Class, and each Subclass. For example, you can assign the following range for different Hierarchy levels:

  Base Class 0 – 99; Class 100 – 199; Subclass 200 – 299

- Can I mix Agile SDK calls and Script PX within Script PX?

  You can mix the two as long as you don't use both SDK calls and script PX calls to update Agile objects in the same Handler code. If you want to update the Agile object, you can either use SDK calls or update APIs that are supported by Script Context object, but not both. For procedures, see Accessing SDK with Scripts on page 463.

Appendix A

Mapping Agile PLM Client Features to Agile API

This Appendix includes the following:

- Login Features 441
- General Features 442
- Search Features 442
- Attachment Features 443
- Workflow Features 443
- Manufacturing Site Features 444
- Folder Features 444
- Projects Features 445
- Administrative Features 445

Login Features

The following table lists general features for logging in to the Agile Application Server.

Feature	Equivalent Method(s)
Get an instance of the Agile Application Server session	`AgileSessionFactory.getInstance()`
Create a session and log in to the Agile Application Server	`AgileSessionFactory.createSession()`
Close a session and disconnect from the Agile Application Server	`IAgileSession.close()`

General Features

The following table lists the General features that apply to all Agile PLM business objects.

Feature	Equivalent Method(s)
Create a new object	`IAgileSession.createObject()`
Load an existing object	`IAgileSession.getObject()`
Save an object as another object	`IDataObject.saveAs()`

Delete an object	`IDataObject.delete()`
	`IFolder.delete()`
	`IQuery.delete()`

Undelete an object	`IDataObject.undelete()`
Get a cell value for an object	`IDataObject.getValue()`
Set an cell value for an object	`IDataObject.setValue()`
Get a table for an object	`IDataObject.getTable()`
Add a row to a table	`ITable.createRow()`
Remove a row from a table	`ITable.removeRow()`
Get subscriptions for an object.	`ISubscribable.getSubscriptions()`
Enable a subscription event.	`ISubscription.enable()`
Modify subscriptions for an object.	`ISubscribable.modifySubscriptions()`

Search Features

Table below lists the supported Search (Query) features.

Feature	Equivalent Method(s)
Set the name of a search	`IQuery.setName()`
Make the search public or private	`IQuery.setQueryType()`
Set the search type for a query (object search or Where Used search)	`IQuery.setSearchType()`
Set and get search criteria	`IQuery.setCriteria()` `IQuery.getCriteria()`
Run a search	`IQuery.execute()`
Make a search case-sensitive	`IQuery.setCaseSensitive()`
Delete a search	`IQuery.delete()`
Save a search as another search	`IQuery.saveAs()`

Attachment Features

Table below lists features for working with attachments and file folders.

Feature	Equivalent Method(s)
Download all files contained in a file folder	`IFileFolder.getFile()`
Download a single file listed on the Attachments tab	`IAttachmentFile.getFile()`
Check out a file folder	`IFileFolder.checkOut()`
Check in a file folder	`IFileFolder.checkIn()`
Cancel checkout	`IFileFolder.cancelCheckOut()`
Incorporate or unincorporate an item, thereby locking or unlocking its attachments	`IAttachmentContainer.setIncorporated()`

Workflow Features

Table below lists Workflow features for routable Agile PLM objects.

Feature	Equivalent Method(s)
Audit a routable object	`IRoutable.audit()`
Change the status of a routable object	`IRoutable.changeStatus()`
Send an object to another Agile PLM user(s)	`IDataObject.send()`
Approve a routable object	`IRoutable.approve()`
Reject a routable object	`IRoutable.reject()`
Comment on a routable object	`IRoutable.comment()`
Add or remove approvers and observers for a routable object	`IRoutable.addApprovers()` `IRoutable.removeApprovers()`

Manufacturing Site Features

The table below lists features for working with manufacturing sites.

Feature	Equivalent Method(s)
Get the current manufacturing site selected for an item	`IManufacturingSiteSelectable.getManufacturingSite()`
Get all manufacturing sites for an item	`IManufacturingSiteSelectable.getManufacturingSites()`
Set an item to use all manufacturing sites.	`IManufacturingSiteSelectable.setManufacturingSite( ManufacturingSiteConstants.ALL_SITES)`
Specify that an item is not site-specific and is common to all sites.	`IManufacturingSiteSelectable.setManufacturingSite( ManufacturingSiteConstants.COMMON_SITE)`
Set an item to use a specific manufacturing site.	`IManufacturingSiteSelectable.setManufacturingSite(site)`

Folder Features

The following table lists the Folder features for working with folders.

Feature	Equivalent Method(s)
Add an item (such as a query) to the folder	`IFolder.addChild()`
Set the type of folder (public or private)	`IFolder.setFolderType()`
Set the folder name	`IFolder.setName()`
Get a folder of the current user	`IUser.getFolder()`
Remove an item from the folder	`IFolder.removeChild()`
Clear all objects from the folder	`IFolder.clear()`
Delete a folder	`IFolder.delete()`

Projects Features

The following table lists features for working with Projects.

Feature	Equivalent Method(s)
Save a Projects as another Projects or template	`IProgram.saveAs()`
Reschedule a Projects	`IProgram.reschedule()`
Assign users from a resource pool	`IProgram.assignUsersFromPool()`
Delegate ownership of a Projects to another user	`IProgram.delegateOwnership()`
Substitute Projects resources	`IProgram.substituteResource()`
Create a baseline	`IProgram.createBaseline()`
Select a baseline view of the Projects	`IProgram.selectBaseline()`
Lock or unlock a Projects	`IProgram.setLock()`
Reply to a discussion	`IMessage.reply()`

Administrative Features

The following table provides the list of features for working with Admin nodes and properties in Agile Java Client.

Feature	Equivalent Method(s)
Get an administrative node	`IAdmin.getNode()`
Get all subnodes (children) of an administrative node	`ITreeNode.getChildNodes()`
Get all properties of an administrative node	`INode.getProperties()`
Get the value for an administrative node's property	`IProperty.getValue()`
Get the possible values for a list field	`IProperty.getAvailableValues()`
Get all Agile PLM classes	`IAdmin.getAgileClasses(ALL)`
Get all top-level Agile PLM classes	`IAdmin.getAgileClasses(TOP)`
Get all Agile PLM classes that can be instantiated	`IAdmin.getAgileClasses(CONCRETE)`
Get the list of subclasses for a specific class	`IAgileClass.getSubclasses()`
Get the Autonumber sources for a subclass	`IAgileClass.getAutoNumberSources()`
Get an array of attributes for a table	`IAgileClass.getTableAttributes()`
Get the metadata for a table	`IAgileClass.getTableDescriptor()`
Get the Agile PLM list library	`IAdmin.getListLibrary()`

Feature	Equivalent Method(s)
Create a new Agile PLM list	`IListLibrary.createAdminList()`
Get an Agile PLM list	`IListLibrary.getAdminList()`
Get all Agile PLM users	Create a query of users
Get all Agile PLM user groups	Create a query of user groups
Create a user or user group	`IAgileSession.createObject()`
Set properties of a user or user group	`IProperty.setValue()`
Change user passwords	`IUser.changeApprovalPassword()` `IUser.changeLoginPassword()`

Appendix B

Migrating Release 9.2.1 and Older Table Constants to Release 9.2.2 or Later

This Appendix includes the following:

- Mapped Pre-Release 9.2.2 Table Constants to 9.2.2 Table Constants 447
- Removed Pre-Release 9.2.2 Table Constants 450

Information about merging and replacing the Relationship tables first appeared in Accessing the New and Merged Relationships Tables on page 69. Tables in this appendix list the Release 9.2.2 table constants and table constants that are either merged and mapped into a single table constant, or mapped into a new table constant.

Mapped Pre-Release 9.2.2 Table Constants to 9.2.2 Table Constants

This table lists the pre-release 9.2.2 table constants and the new table constants that they are either merged and mapped into Release 9.2.2, or mapped to in later releases of the SDK.

Pre 9.2.2 Table Constants	9.2.2 Table Constants
▫ TABLE_RELATIONSHIPSAFFECTEDBY ▫ TABLE_RELATIONSHIPSAFFECTS ▫ TABLE_REFERENCES	TABLE_RELATIONSHIPS
▫ ATT_RELATIONSHIPS_AFFECTED_BY_CRITERIA_MET ▫ ATT_RELATIONSHIPS_AFFECTS_CRITERIA_MET	ATT_RELATIONSHIPS_CRITERIA_MET
▫ ATT_RELATIONSHIPS_AFFECTED_BY_CURRENT_STATUS ▫ ATT_RELATIONSHIPS_AFFECTS_CURRENT_STATUS	ATT_RELATIONSHIPS_CURRENT_STATUS
▫ ATT_RELATIONSHIPS_AFFECTED_BY_DATE01 ▫ ATT_RELATIONSHIPS_AFFECTS_DATE01	ATT_RELATIONSHIPS_DATE01
▫ ATT_RELATIONSHIPS_AFFECTED_BY_DATE02 ▫ ATT_RELATIONSHIPS_AFFECTS_DATE02	ATT_RELATIONSHIPS_DATE02
▫ ATT_RELATIONSHIPS_AFFECTED_BY_DATE03 ▫ ATT_RELATIONSHIPS_AFFECTS_DATE03	ATT_RELATIONSHIPS_DATE03
▫ ATT_RELATIONSHIPS_AFFECTED_BY_DATE04 ▫ ATT_RELATIONSHIPS_AFFECTS_DATE04	ATT_RELATIONSHIPS_DATE04
▫ ATT_RELATIONSHIPS_AFFECTED_BY_DATE05 ▫ ATT_RELATIONSHIPS_AFFECTS_DATE05	ATT_RELATIONSHIPS_DATE05

▫ ATT_REFERENCES_DATE01	ATT_RELATIONSHIPS_DATE06
▫ ATT_REFERENCES_DATE02	ATT_RELATIONSHIPS_DATE07
▫ ATT_REFERENCES_DATE03	ATT_RELATIONSHIPS_DATE08
▫ ATT_REFERENCES_DATE04	ATT_RELATIONSHIPS_DATE09
▫ ATT_REFERENCES_DATE05	ATT_RELATIONSHIPS_DATE10
▫ ATT_RELATIONSHIPS_AFFECTED_BY_DESCRIPTION ▫ ATT_RELATIONSHIPS_AFFECTS_DESCRIPTION ▫ ATT_REFERENCES_DESCRIPTION	ATT_RELATIONSHIPS_DESCRIPTION
▫ ATT_RELATIONSHIPS_AFFECTED_BY_LIST01, ATT_RELATIONSHIPS_AFFECTS_LIST01	ATT_RELATIONSHIPS_LIST01
▫ ATT_RELATIONSHIPS_AFFECTED_BY_LIST02, ATT_RELATIONSHIPS_AFFECTS_LIST02	ATT_RELATIONSHIPS_LIST02
▫ ATT_RELATIONSHIPS_AFFECTED_BY_LIST03 ▫ ATT_RELATIONSHIPS_AFFECTS_LIST03	ATT_RELATIONSHIPS_LIST03
▫ ATT_RELATIONSHIPS_AFFECTED_BY_LIST04 ▫ ATT_RELATIONSHIPS_AFFECTS_LIST04	ATT_RELATIONSHIPS_LIST04
▫ ATT_RELATIONSHIPS_AFFECTED_BY_LIST05 ▫ ATT_RELATIONSHIPS_AFFECTS_LIST05	ATT_RELATIONSHIPS_LIST05
▫ ATT_REFERENCES_LIST01	ATT_RELATIONSHIPS_LIST06
▫ ATT_REFERENCES_LIST02	ATT_RELATIONSHIPS_LIST07
▫ ATT_REFERENCES_LIST03	ATT_RELATIONSHIPS_LIST08
▫ ATT_REFERENCES_LIST04	ATT_RELATIONSHIPS_LIST09
▫ ATT_REFERENCES_LIST05	ATT_RELATIONSHIPS_LIST10
▫ ATT_RELATIONSHIPS_AFFECTED_BY_MULTITEXT01 ▫ ATT_RELATIONSHIPS_AFFECTS_MULTITEXT01	ATT_RELATIONSHIPS_MULTITEXT01
▫ ATT_RELATIONSHIPS_AFFECTED_BY_MULTITEXT02 ▫ ATT_RELATIONSHIPS_AFFECTS_MULTITEXT02	ATT_RELATIONSHIPS_MULTITEXT02
▫ ATT_RELATIONSHIPS_AFFECTED_BY_MULTITEXT03 ▫ ATT_RELATIONSHIPS_AFFECTS_MULTITEXT03	ATT_RELATIONSHIPS_MULTITEXT03
▫ ATT_RELATIONSHIPS_AFFECTED_BY_MULTITEXT04 ▫ ATT_RELATIONSHIPS_AFFECTS_MULTITEXT04	ATT_RELATIONSHIPS_MULTITEXT04
▫ ATT_RELATIONSHIPS_AFFECTED_BY_MULTITEXT05 ▫ ATT_RELATIONSHIPS_AFFECTS_MULTITEXT05	ATT_RELATIONSHIPS_MULTITEXT05
▫ ATT_REFERENCES_MULTITEXT01	ATT_RELATIONSHIPS_MULTITEXT06
▫ ATT_REFERENCES_MULTITEXT02	ATT_RELATIONSHIPS_MULTITEXT07
▫ ATT_REFERENCES_MULTITEXT03	ATT_RELATIONSHIPS_MULTITEXT08

▫ ATT_REFERENCES_MULTITEXT04	ATT_RELATIONSHIPS_MULTITEXT09
▫ ATT_REFERENCES_MULTITEXT05	ATT_RELATIONSHIPS_MULTITEXT10
▫ ATT_RELATIONSHIPS_AFFECTED_BY_TEXT01 ▫ ATT_RELATIONSHIPS_AFFECTS_TEXT01	ATT_RELATIONSHIPS_TEXT01
▫ ATT_RELATIONSHIPS_AFFECTED_BY_TEXT02 ▫ ATT_RELATIONSHIPS_AFFECTS_TEXT02	ATT_RELATIONSHIPS_TEXT02
▫ ATT_RELATIONSHIPS_AFFECTED_BY_TEXT03, ATT_RELATIONSHIPS_AFFECTS_TEXT03	ATT_RELATIONSHIPS_TEXT03
▫ ATT_RELATIONSHIPS_AFFECTED_BY_TEXT04, ATT_RELATIONSHIPS_AFFECTS_TEXT04	ATT_RELATIONSHIPS_TEXT04
▫ ATT_RELATIONSHIPS_AFFECTED_BY_TEXT05 ▫ ATT_RELATIONSHIPS_AFFECTS_TEXT05	ATT_RELATIONSHIPS_TEXT05
▫ ATT_REFERENCES_TEXT01	ATT_RELATIONSHIPS_TEXT06
▫ ATT_REFERENCES_TEXT02	ATT_RELATIONSHIPS_TEXT07
▫ ATT_REFERENCES_TEXT03	ATT_RELATIONSHIPS_TEXT08
▫ ATT_REFERENCES_TEXT04	ATT_RELATIONSHIPS_TEXT09
▫ ATT_REFERENCES_TEXT05	ATT_RELATIONSHIPS_TEXT10
▫ ATT_RELATIONSHIPS_AFFECTED_BY_NOTES ▫ ATT_RELATIONSHIPS_AFFECTS_NOTES	ATT_RELATIONSHIPS_NOTES1
▫ ATT_REFERENCES_NOTES	ATT_RELATIONSHIPS_NOTES2
▫ ATT_RELATIONSHIPS_AFFECTED_BY_NUMBER ▫ ATT_RELATIONSHIPS_AFFECTS_NUMBER ▫ ATT_REFERENCES_NUMBER	ATT_RELATIONSHIPS_NAME

Removed Pre-Release 9.2.2 Table Constants

The following pre-release 9.2.2 table constants are no longer available and should not be used in later releases of the SDK:

- ATT_RELATIONSHIPS_AFFECTED_BY_EVENT
- ATTRELATIONSHIPS_AFFECTS_TRIGGER_EVENT
- ATT_RELATIONSHIPS_AFFECTED_BY_TRIGGER_EVENT
- ATT__RELATIONSHIPS_AFFECTS_EVENT
- ATT_RELATIONSHIPS_AFFECTS_RESULT
- MaterialDeclarationConstants.TABLE_RELATIONSHIPSAFFECTEDBY
- MaterialDeclarationConstants.TABLE_RELATIONSHIPSAFFECTS
- MaterialDeclarationConstants.TABLE_REFERENCES

Appendix C

Migrating Custom Process Extensions to Event Framework

This Appendix includes the following:

- About this Appendix 451
- Understanding Custom PXs and Java PXs 451
- Migration Task List 452

About this Appendix

This appendix provides information to:

1. Modify the Custom PX Java code created in Developing Process Extensions for use in Event framework.
2. Configure the modified code to function as a Java PX in the Event framework.

Understanding Custom PXs and Java PXs

The following paragraphs describe the difference between these PXs and lists the Custom PXs that you can migrate to Event framework and configure as Java PXs in Working with Events on page 399.

Custom PXs in PX Framework

Custom PXs defined and configured in Developing Process Extensions are a Java class deployed on the Agile Application Server, or a link to a URL. These PXs run in the PX framework. See the illustration in SDK Architecture on page 2. The Java class Custom PXs includes the following types:

- Custom action PXs that implemented the server-side Java API `ICustomAction` interface in `com.agile.px` package
- Custom autonumber source PXs that implemented the server-side Java API `ICustomAutoNumber` interface in `com.agile.px` package

Process Extensions in Event Framework

PXs configured in the Working with Events on page 399 are one of the following types:

- Java process extensions (Java PX) that implemented the server-side API `IEventAction` interface in `com.agile.px` package
- Script process extensions (Script PX) that implemented the server-side API `invokeScript(IBaseScriptObj obj)` interface in `com.agile.agile.DSLObj` package

Note To ensure proper operation of all Java process extensions, Oracle recommends recompiling existing Java-based PXs with the version of the JDK that is shipped with your application server.

Custom PXs You Can Migrate to Event Framework

The Event framework supports migrating only Custom Action PXs. These are PXs that are initiated from the Actions Menu, Tools Menu, and Workflow State (Status). See Using the Process Extension Library on page 350.

The corresponding Event types are:

Custom PX	Java PX
Actions Menu	Extend Actions Menu
Tools Menu	Extend Tools Menu
Workflow State (Status)	Change Status for Workflow

Migration Task List

Complete the following tasks to properly migrate a Custom PX to the Event framework and ensure its proper operation as a Java PX in this environment.

Task - 1: Modify the Custom PX Code

The following code samples show how to change an existing Custom PX code to a Java PX code in Event framework. The principal difference between a Java PX and a Custom PX is the interface the PX must implement. Java PXs implement `IEventAction` and Custom PXs implement `ICustomAction`.

Custom PX Code

This is an example of an existing Custom PX that appeared in Defining a Custom Action. To migrate Custom PXs, you must modify the code as shown in Java PX Code on page 453.

Example: A Custom PX

```
public class HelloWorld implements ICustomAction
{
    public ActionResult doAction(IAgileSession session, INode
    actionNode, IDataObject affectedObject)
    {
                ...

            return new ActionResult(ActionResult.STRING, "Hello World");
    }
}
```

Java PX Code

The following example is the modified Custom PX to run in the Event framework. Code modifications that enabled this migration appear in the **bold** font.

Example: The Custom PX code after modification

```
public class HelloWorld implements IEventAction
{
    public EventActionResult doAction(IAgileSession session, INode
    actionNode, IEventInfo request)
    {
        IObjectEventInfo objectEventInfo = (IObjectEventInfo) request;
        IDataObject affectedObject = objectEventInfo.getDataObject();

              ...

        ActionResult actionResult = new ActionResult(ActionResult.STRING,
        "Hello World");
        return new EventActionResult(request, actionResult);
    }
}
```

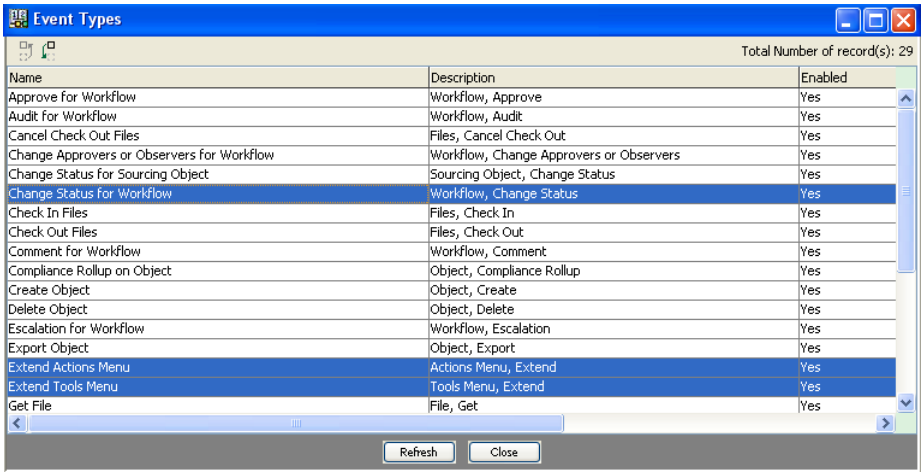
Note Java PXs get their `IDataObject` values from `IEventInfo`.

Task - 2: Package and Deploy the Modified Code

Create JAR files to package and deploy the modified Java code for use in the Event framework. For procedures, see Packaging and Deploying a Custom Action on page 349. This package is the "action" that you will use to complete Task - 3: Configure Event in Event Framework on page 454.

Task - 3: Configure Event in Event Framework

The Event Types dialog lists the supported Event types in Event framework. The supported types for Custom PXs you are migrating are Change Status for Workflow, Extend Actions Menu, and Extend Tools Menu. Procedures to create and configure these Events appear next. You can find additional information in Working with Events on page 399 and in the *Agile PLM Administrator Guide*.



Create Event

To create the Event:

1. Log in to Java Client with Administrator privileges.
2. Select Admin > System Settings > Event Management > Events > New button . The Create Event dialog appears.

3. In Event Type, point to the drop-down arrow select your Event from the list.

Figure 34: Create an Extend Tools Menu Event

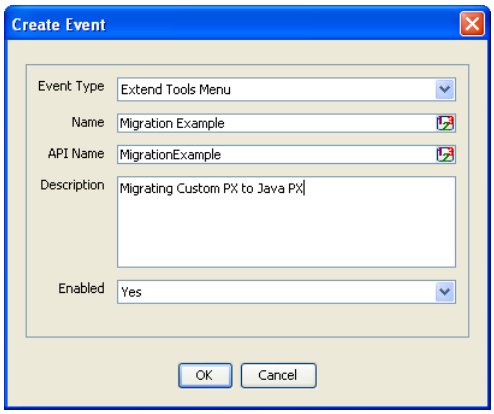


Figure 35: Change Status for Workflow

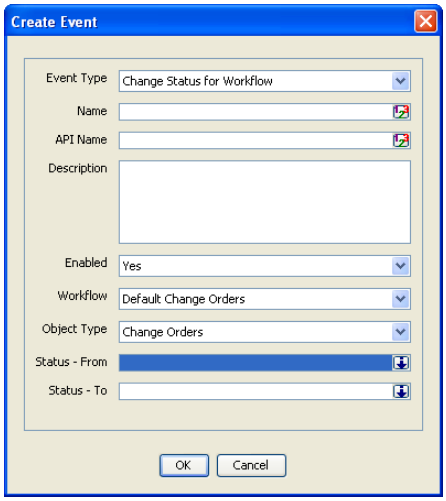
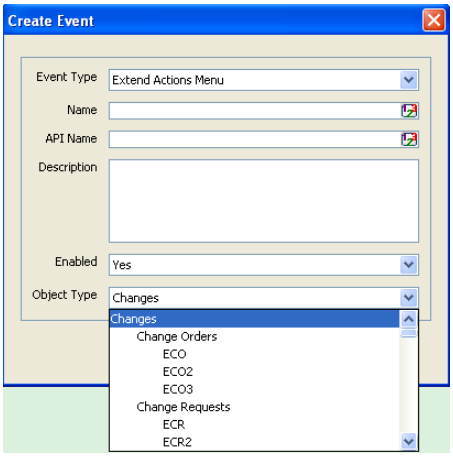


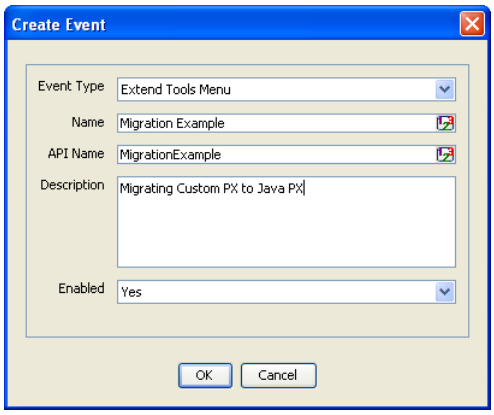
Figure 36: Extend Actions Menu



Note Fields in the Create Event dialog are not the same for all Event Types. For example, in Update Title Block, you are assigning the object's class, but in Approve for Workflow, you select a status for the Workflow. For information on assigning object classes and Workflow status, see Assigning Process Extensions to Classes Assigning Process Extensions to Classes and Assigning Process Extensions to Workflow Statuses.

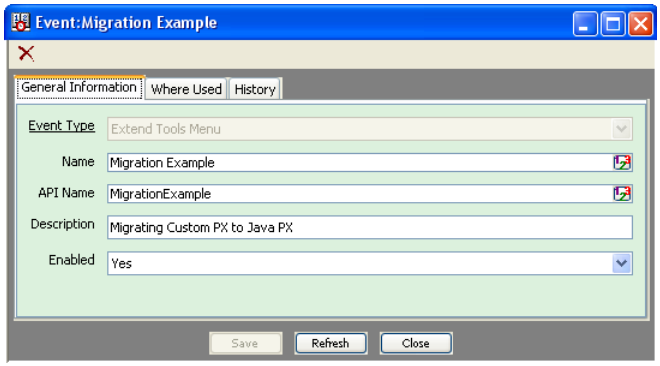
4. Select your Event Type and provide the required information, for example, as shown below. For more information on completing this dialog, refer to *Agile PLM Administrator Guide*.

Figure 37: Create an Extend Tools Menu Event



5. Click OK. The Event:<Event _Name> page appears.

Figure 38: Event General Information page



You can modify the Event from this dialog. When you make a change, the Save button is enabled. Also, this Event is listed in the Events view. To view, select Event Management > Events. The next task is to create the handler for this Event.

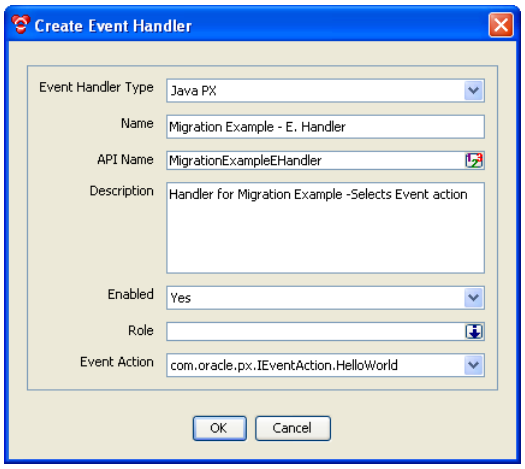
Create Event Handler

The Event Handler enables executing the compiled Java code. The following procedure guides you through this step.

To create the Event Handler:

1. In Java Client with Administrator privileges and select Admin > Systems Settings > Event Management > Event Handlers > New button . The Create Event Handler dialog opens.

Figure 39: Create event Handler dialog



2. In Create Event Handler dialog, do as follows:
   - In Event Handler drop-down list, select Java PX.
   - In Event Action field, select the one you created in Task - 2: Package and Deploy the Modified Code on page 454.
   - In Role field, if left blank, Event Handler will use the roles and privileges of the current user by default. However, you can configure a custom action to have override privileges. Refer to *Agile PLM Administrator Guide* for information and procedures.
   - Complete the remaining fields and click OK.

Create Event Subscriber

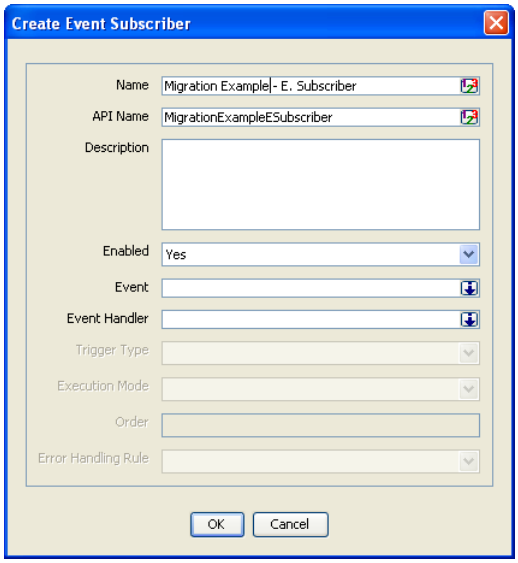
This is a process that binds a Java PX to a specific event. This is done using the Create Event Subscriber dialog to:

- Bind the Event with the Event Handler
- Set the triggering order (sequence)
- Set the execution mode

To create Event Subscriber:

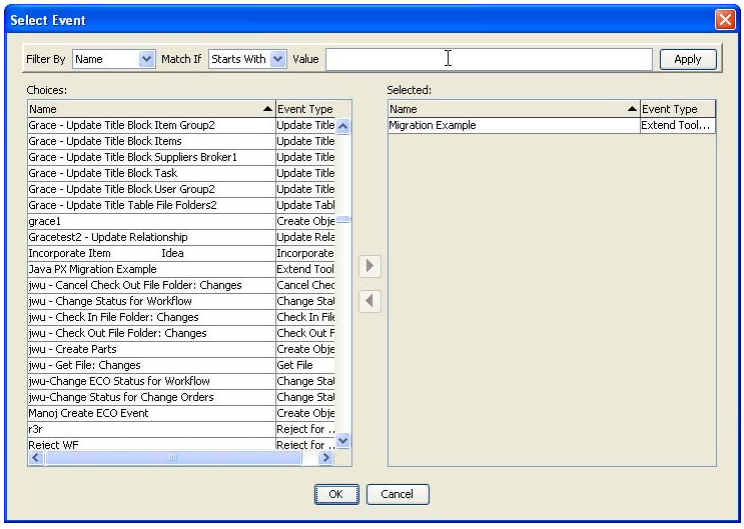
1. Log in to Java Client with Administrator privileges and select Admin > Systems Settings > Event Management > Event Subscribers > New button . The Create Event Subscriber dialog opens.

Figure 40: Create Event Subscriber dialog



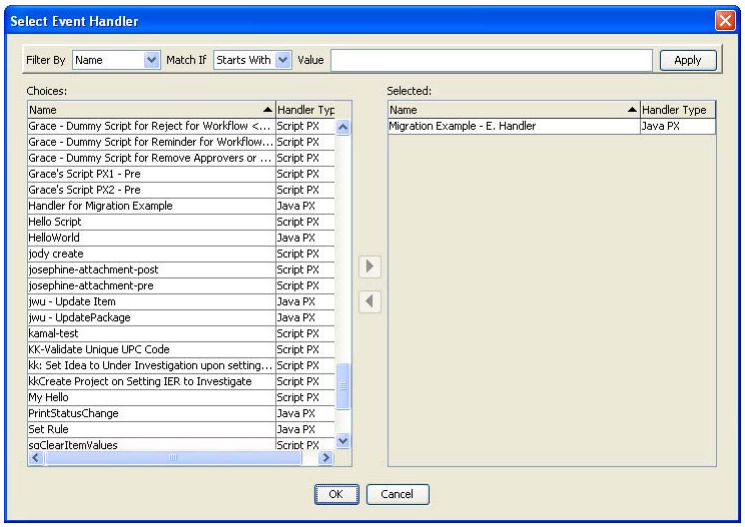
2. In the Create Event Subscriber dialog, do as follows:
   - Select Event for this Event Subscriber: Click the drop-down list in the Event field. The Select Event dialog opens.
   - In Select Event dialog locate and select the applicable Event and move it to the Selected column.

Figure 41: Select Event for Event Subscriber



- Select Event Handler for this Event Subscriber: Click the drop-down list in the Event Handler field. The Select Event Handler dialog opens. In Select Event Handler dialog locate and select the applicable Event Handler and move it to the Selected column.

Figure 42: Select Event Handler for Subscriber



Note After completing this step, some of the four fields in the Create Event Subscriber dialog that are grayed out can be configured now. For example, Trigger Type and Execution Mode.

Configure Trigger Type, Execution Mode, Order, and Error Handling Rule

Event trigger types and Event execution mode plus the order in which the Event is invoked and error handling rule options that you must set are defined below.

Trigger Type Field

This field has two options as follows:

- Pre – This trigger type signals a point prior to the occurrence of an action. The Pre trigger is commonly used for events that require data or other preparations for the upcoming action. Event subscribers configured with this trigger type are executed in the Synchronous Execution Mode only.
- Post – This trigger type signals a point immediately after the occurrence of an action. This trigger is used for events that perform auditing tasks based on the prior action. You can execute Event subscribers configured with this trigger type in either Synchronous or Asynchronous Execution Modes.

Note For migrated Custom PXs, always select Post. Custom PXs always run after the action has occurred.

Task - 4: Test the Migrated PX in Event Framework

Invoke the new Java PX to ensure the action specified in the Handler (code) occurs. Depending on the Event type, the Java PX can be invoked by a user from the Extend Actions or Extend Tools menu, or by a Change Status in Workflow. Make sure the migrated PX's behavior in Event framework is the same as its behavior in PX framework.

If the PX is configured to be invoked by a user action from the Tools menu, you can test it in Web Client or Java Client as follows:

- In Web Client tool bar select the button > <Event_name>.
- In Java Client tool bar select the Process Extensions button or Tools > Process Extensions > <Event_name>.

Task - 5: Remove Custom PX from Process Extension Library

It is a good practice to delete the Custom PX that you have migrated from the Agile Process Extensions Library. This will prevent any duplicate execution of the Custom PX and Java PX.

To remove the Custom PX from PX library:

1. Delete all references to the Custom PX and because you cannot remove the Custom PX if it is in use.
2. Open Java Client's Process Extensions Library and select and delete the PX.

Task - 6: Inform PLM Administrator

As shown in the Task - 3: Configure Java in Event Framework Task - 3: Configure Event in Event Framework on page 454, Event configuration is a UI-based Admin function performed in the Java Client by an Admin user. Depending on your role as the SDK developer, be sure to inform the PLM Admin as follows:

- If Event Subscriber creation and configuration is a PLM Admin function, be sure to inform the cognizant administrator after deploying the modified Custom PX code in Task - 2: Package and Deploy the Modified Code on page 454. This is to inform the PLM Admin of the new Event Handler and its specifics to use and complete the remaining tasks.
- If on the other hand you perform the UI-based configurations in Java Client, inform your PLM Admin of the new Java PX, its intended purpose and function and the necessary information to use the new Java PX in PLM clients.

Appendix D

Groovy Implementation in Event Framework

This Appendix includes the following:

- About this Appendix 461
- What is Groovy? 461
- Event Framework Implementation 462

About this Appendix

This appendix provides a description of the Groovy scripting language and sources of information about this tool, Other topics addressed include how to start a script, access the SDK using scripts, and sample use cases

What is Groovy?

Groovy is an object-oriented programming language that can be used as a scripting language for the Java Platform.

Sources of Information

World Wide Web provides links to many sites that offer information about Groovy. Publishers and vendors of the print media also offer information on this tool. A few are listed below.

- From World Wide Web:
  - Groovy Home – Provides links to documentation, downloads, tutorial, user guide, Eclipse plugin examples, advanced usage guide, and other sites maintained by Groovy Home (http://groovy.codehaus.org/)
- From publishers and vendors:
  - Publisher: Manning Publications – Groovy in Action by Dierk Koenig, Andrew Glover, Paul King, and Guillaume Laforge
  - Publisher: Morgan Kaufmann – Groovy Programming: An Introduction for Java Developers by Kenneth Barclay and John Savage

Script PX or Java PX?

Scripts are suitable for rapid development and deployment of applications with simple business logics. They empower Agile PLM Admins and power users to develop extensions unique to their requirements and make rapid modifications when necessary. Scripts are not suitable for developing complex applications with performance critical data structures.

Use scripts to:

- Automate functions with simple business logic such as data validation, notification, or defaulting field values
- Implement unique customization for existing applications
- Build extensions to existing systems
- Rapid prototyping
- Write test use cases

Event Framework Implementation

Event framework implementation requires running the scripting engine. Key inplementation considerations are summarized below.

Key implementation considerations

- The scripting engine runs inside the Java 2 Platform Enterprise Edition (J2EE) on the Agile PLM server and is based on the Groovy language.
- Groovy is fully embedded in Event framework.
- Groovy is the only supported scripting language.
- Script codes are currently stored in the CLOB field in Agile PLM database.
- Event Script objects (Event Script PX handlers) that you develop are text files that are deployed from the Event Management Node by selecting Event Management > Event Handlers in Java Client.
- Scripts can call Script API and Agile SDK.
- The *Agile PLM Administrator Guide* provides both background information and sample procedures to understand and manage Events.

Starting a Script

1. Use `void invokeScript(IBaseScriptObj obj)` to start your script. `InvokeScript` is script's starting point of the execution and `IBaseScriptObj` is the base interface for all Event Script Objects.
2. The run time type of the `obj` is dynamically resolved based on the type of Event that invoked this script. For example, if the script is invoked on a Create Event, then `ICreateScriptObj` is dynamically resolved as the type of `obj` at run time. You can invoke any method defined on `ICreateScriptObj` and its super-interfaces on the instance of this `obj`.
3. Use `IAgileSession getAgileSDKSession()` to access SDK session and invoke SDK functions.
4. Use `AgileDSLException` to return "exception" information from scripts.

Accessing SDK with Scripts

You can access an SDK session and Java PX Event object to write SDK programs in script.

```
/**
    * Returns agile SDK session.
    * @return agile SDK session
    * @throws AgileDSLException
    *       if the method fails
    * @since \@Agile93@
*/
public IAgileSession getAgileSDKSession() throws AgileDSLException;

/**
    * Returns PX eventInfo.
    * @return PX eventInfo
    * @throws AgileDSLException
    *       if the method fails
    * @since \@Agile93@
*/
public IEventInfo getPXEventInfo() throws AgileDSLException;
```

Use Cases

You can find Script PX Handler examples for most Event types in Client-Side Components on page 2. These handlers utilize the Pre/Post triggers and Synchronous/Asynchronous executions and cause different actions when they are invoked.