



Agile SDK Developer Guide

Release 9.2.1

Part No. E11138-01

Make sure you check for updates to this manual at the
[Oracle Technology Network \(OTN\) Web site](#)

Copyrights and Trademarks

Copyright © 1995, 2007, Oracle. All rights reserved.

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

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

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

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

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

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

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

September 12, 2007

REVISIONS

Revision	Date	Pages Affected	Description
A	9/12/07	All	Initial release of the manual for 9.2.1

CONTENTS

Preface

Agile PLM Documentation	xv
Readme	xv
Agile Training	xv
Developer Documentation and Samples	xv

Chapter 1 Introduction

What is the Agile SDK?	1-1
Client-Side Components	1-1
Server-Side Components	1-2
Architecture	1-2
Agile XML (also known as aXML)	1-2
What's New for this Release	1-3
System Requirements	1-4
Java Requirements	1-4
Agile SDK Installation Folders	1-4
Checking Your Agile PLM System	1-4
Agile PLM Business Objects	1-4
Licensing	1-5

Part I, Using the Agile API

Chapter 2 Getting Started with the Agile API

Agile API Overview	2-1
Types of Agile API Classes and Interfaces	2-1
Network Class Loading	2-2
Single-Threaded versus Multithreaded Applications	2-3
Packaging Your Agile API Programs	2-3
Agile API Files You Are Allowed to Distribute	2-3
Agile API Files You Are Not Allowed to Distribute	2-3
Sample Programs	2-3
Starting an Agile API Program	2-4
Setting the Class Path for the Agile API Library	2-4
Importing Agile API Classes	2-4
Creating a Session and Logging In	2-4
Creating a Session by Accessing a Password-Protected URL	2-5
Creating a Session from an Agile Web Service	2-6
Loading and Creating Agile PLM Objects	2-7
Loading Objects	2-7
Specifying Object Types	2-7
Specifying Object Parameters	2-7
Loading Different Types of Objects	2-8
Creating Objects	2-10
Working with Agile PLM Classes	2-11
Creating Objects of User-Defined Subclasses	2-12
Using AutoNumbers	2-12

Handling Required Fields	2-14
Creating Different Types of Objects	2-15
Propagating Values to Related Objects	2-18
Saving an Object to a New Object	2-18
Sharing an Object	2-19
Deleting and Undeleting Objects	2-19
Closing a Session	2-21

Chapter 3 Creating and Loading Queries

About Queries	3-1
Creating a Query	3-1
Saving a Query to a Folder	3-2
Creating a Parameterized Query	3-2
Specifying Query Attributes when Creating a Query	3-3
Specifying Search Criteria	3-4
Search Conditions	3-4
Query Language Keywords	3-5
Specifying Search Attributes	3-5
Retrieving Searchable Attributes	3-6
Using Relational Operators	3-6
Using Unicode Escape Sequences	3-7
Using Between, Not Between, In, and Not In Operators	3-7
Using Any, All, None Of, and Not All Relational Operator Modifiers	3-7
Using Nested Criteria to Search for Values in Object Lists	3-8
Searching for Words or Phrases Contained in Attachments	3-8
Formatting Dates in Query Criteria	3-9
Using Logical Operators	3-9
Using Wildcard Characters with the Like Operator	3-10
Using SQL Syntax for Search Criteria	3-10
Using SQL Wildcards	3-12
Sorting Query Results Using SQL Syntax	3-12
Setting Result Attributes for a Query	3-12
Duplicate Results for Site-Related Objects and AMLs	3-17
Working with Query Results	3-17
Sorting Query Results	3-17
Datatype of Query Results	3-17
Managing Large Query Results	3-18
Query Performance	3-18
Creating a Where-Used Query	3-18
Loading a Query	3-19
Deleting a Query	3-20
Simple Query Example	3-20

Chapter 4 Working with Tables

About Tables	4-2
Retrieving a Table	4-2
Working with Read-only Tables	4-11
Retrieving the Metadata of a Table	4-11
Adding Table Rows	4-11
Adding an Item to the BOM Table	4-12
Adding an Attachment to the Attachments Table	4-12
Adding a Manufacturer Part to the Manufacturers Table	4-12

Adding an Item to the Affected Items Table.....	4-13
Adding a Task to the Schedule Table.....	4-13
Adding and Updating Multiple Table Rows.....	4-13
Adding Multiple Items to the BOM Table	4-14
Updating Multiple BOM Rows	4-14
Iterating Over Table Rows	4-15
Sorting Tables	4-16
Removing Table Rows	4-17
Retrieving the Referenced Object for a Row	4-18
Checking Status Flags of a Row.....	4-21
Working with Page One, Page Two, and Page Three.....	4-22
Redlining.....	4-22
Removing Redlines.....	4-24

Chapter 5 Working with Data Cells

About Data Cells	5-1
Data Types	5-1
Checking Discovery Privilege	5-2
Checking Whether a Cell is Read-Only	5-2
Getting Values.....	5-3
Setting Values	5-4
Catching Exceptions for Locked Objects	5-5
Getting and Setting List Values.....	5-5
Getting and Setting Values for SingleList Cells	5-6
Getting and Setting Values for MultiList Cells.....	5-6
Getting and Setting Values for Cascading Lists	5-7
Using Reference Designator Cells.....	5-8

Chapter 6 Working with Folders

About Folders	6-1
Loading a Folder.....	6-2
Creating a Folder	6-3
Setting the Folder Type	6-3
Adding and Removing Folder Elements	6-3
Getting Folder Elements	6-4
Deleting a Folder	6-6

Chapter 7 Working with Items, BOMs, and AMLs

About Items	7-1
Getting and Setting the Revision of an Item	7-1
Changing the Incorporated Status of a Revision.....	7-3
Working with BOMs	7-4
Adding an Item to a BOM	7-4
Expanding a BOM	7-5
Copying one BOM into another BOM.....	7-5
Redlining a BOM	7-6
Getting a Released Assembly Item.....	7-6
Creating a Change Order	7-6
Adding an Item to the Affected Items tab of a Change Order	7-7
Modifying the Redline BOM Table.....	7-7
Working with AMLs.....	7-8

Adding an Approved Manufacturer to the Manufacturers Table	7-8
Redlining an AML	7-9

Chapter 8 Working with Lists

About Lists	8-1
List Library	8-1
SingleList Lists	8-2
MultiList Lists	8-3
Cascading Lists	8-3
Methods that Use IAgileList	8-4
Selecting a List Value	8-4
Working with Dynamic Lists	8-6
Working with Lifecycle Phase Cells	8-7
Selecting a List from the List Library	8-7
Creating Custom Lists	8-9
Creating a Simple List	8-9
Creating a New List Automatically by Modifying an Existing List	8-10
Creating a Cascading List	8-11
Checking the Data Type of a List	8-12
Modifying a List	8-13
Adding a Value to a List	8-13
Making List Values Obsolete	8-14
Setting the List Name and Description	8-14
Setting Level Names for a Cascading List	8-14
Enabling or Disabling a List	8-15
Deleting a List	8-15
Printing the Contents of an IAgileList Object	8-15

Chapter 9 Managing Manufacturing Sites

About Manufacturing Sites	9-1
Controlling Access to Sites	9-1
Creating a Manufacturing Site	9-2
Loading a Manufacturing Site	9-2
Retrieving the Sites Table for an Item	9-2
Adding a Manufacturing Site to the Sites Table	9-3
Selecting the Current Manufacturing Site for an Item	9-3
Disabling a Site	9-5

Chapter 10 Working with Attachments and File Folders

About Attachments	10-1
Working with File Folders	10-2
File Folder Tables	10-2
Working with the Files Table of a File Folder	10-3
Using the IAttachmentFile Interface	10-3
Working with the Attachments Table	10-4
Using the ICheckoutable Interface	10-5
Specifying the Revision of the Item	10-6
Checking Whether the Revision is Incorporated	10-6
Checking Out a File Folder	10-6
Canceling Checkout of a File Folder	10-6

Adding Files and URLs to the Attachments Table.	10-7
Deep Cloning Attachments and Files from One Object to Another	10-8
Specifying the File Folder Subclass When Adding Attachments	10-9
Setting the Version of File Folder Files	10-10
Checking In a File Folder.	10-10
Replacing a File.	10-11
Retrieving Attachment Files	10-12
Deleting File Folders and Attachments	10-13

Chapter 11 Managing Workflow

About Workflow.	11-1
The Change Control Process	11-1
Dynamics of Workflow Functionality	11-2
How the Status of a Change Affects Workflow Functionality	11-2
How User Privileges Affect Workflow Functionality	11-2
Selecting a Workflow	11-3
Adding and Removing Approvers.	11-4
Approving or Rejecting a Change	11-5
Commenting a Change	11-6
Auditing a Change	11-6
Changing the Workflow Status of an Object	11-7
Sending an Agile Object to Selected Users	11-8
Sending an Agile Object to User Groups	11-9

Chapter 12 Product Cost Management

About Product Sourcing	12-1
Managing Pricing	12-2
Creating a Price Object	12-2
Loading a Price Object	12-3
Adding Price Lines	12-3
Creating a Price Change Order	12-5
Working with Suppliers	12-5
Loading a Supplier	12-5
Modifying Supplier Data	12-6
Working with Sourcing Projects	12-6
Loading a Project	12-7
Setting Cover Page Values for a Project	12-7
Working with RFQs	12-8
Loading an RFQ	12-8
Working with RFQ Responses	12-8
Loading an RFQ Response	12-9

Chapter 13 Managing and Tracking Quality

About Quality Control.	13-1
Quality-Related API Objects	13-1
Quality-Related Roles and Privileges	13-2
Working with Customers	13-2
About Customers	13-2
Creating a Customer	13-2
Loading a Customer	13-2
Saving a Customer as Another Customer	13-3

Working with Product Service Requests	13-3
About Problem Reports	13-3
About Nonconformance Reports	13-3
Creating a Product Service Request	13-4
Assigning a Product Service Request to a Quality Analyst	13-4
Adding Affected Items to a Product Service Request	13-4
Adding Related PSRs to a Product Service Request	13-5
Working with Quality Change Requests	13-6
Creating a Quality Change Request	13-6
Assigning a Quality Change Request to a Quality Administrator	13-6
Saving a Quality Change Request as a Change	13-7
Using Workflow Features with PSRs and QCRs	13-7
Selecting a Workflow	13-8

Chapter 14 Creating and Managing Programs

About Programs	14-1
Differences in Behavior of Program Objects	14-1
Creating a Program	14-2
Loading a Program	14-3
Using Program Templates	14-3
Scheduling a Program	14-5
Working with Program Baselines	14-7
Delegating Ownership of a Program to Another User	14-8
Adding Resources to a Program's Team	14-8
Substituting Program Resources	14-10
Locking or Unlocking a Program	14-11
Working with Discussions	14-11
Creating a Discussion	14-12
Replying to a Discussion	14-13
Joining a Discussion	14-15
Creating an Action Item	14-15

Chapter 15 Product Governance & Compliance

About Agile Product Governance and Compliance	15-1
Agile PG&C Interfaces and Classes	15-2
Agile PG&C Roles	15-2
Creating Declarations, Specifications, and Substances	15-3
Creating Declarations	15-3
Creating Specifications	15-4
Creating Substances	15-4
Creating a Subpart	15-5
Creating a Substance Group	15-5
Creating a Material	15-5
Creating a Substance	15-6
Adding Items, Manufacturer Parts, and Part Groups to a Declaration	15-6
Adding Substances to a Declaration	15-7
Structure of Bill of Substances	15-8
Rules for Adding Substances	15-8
Adding Subparts and Materials that Do Not Exist	15-9
Example: Adding Substances to the Manufacturer Part Composition Table of a Homogeneous Material Declaration	15-10
Example: Adding Substances to the Manufacturer Part Composition Table of a Substance Declaration	15-11

Adding Substances to a Specification	15-12
Adding Specifications to a Declaration	15-12
Rules for Adding Specifications	15-13
Routing Declarations	15-13
Filling Out a Declaration	15-15
Submitting a Declaration to the Compliance Manager	15-15
Publishing a Declaration	15-16
Getting and Setting Weight Values	15-16
Adding Substance Compositions for a Manufacturer Part	15-17

Chapter 16 Subscribing to Agile PLM Objects

About User Subscriptions	16-1
Subscription Events	16-1
Subscribe Privilege	16-2
Subscription Notifications	16-2
Deleting Subscribed Objects	16-2
Getting the Subscriptions for an Object	16-2
Modifying the Subscriptions for an Object	16-3
Making Attributes Available for Subscription	16-4
Parent and Child Attributes	16-5
Working with the Subscription Table	16-5

Chapter 17 Creating Multithreaded Applications

About Multithreading	17-1
When to Use Threads	17-1
Creating a Thread	17-2
Thread Class Methods	17-2
Using the Multithreaded Sample Application	17-2
How the Multithreaded Sample Works	17-2
Running Queries as Independent Threads	17-3
Processing Queries using MtsQuery	17-4
Implementing the QueryThread Class	17-5
For More Information About Threads	17-6

Chapter 18 Performing Administrative Tasks

About Agile PLM Administration	18-1
Privileges Required to Administer Agile PLM	18-2
Administrative Interfaces	18-2
Getting an IAdmin Instance	18-2
Working with Nodes	18-3
Working with the Classes Node	18-6
Managing Agile PLM Classes	18-6
Concrete and Abstract Classes	18-8
Referencing Classes	18-9
Identifying the Target Type of a Class	18-9
Working with Attributes	18-10
Referencing Attributes	18-10
Retrieving Attributes	18-11
Retrieving Individual Attributes	18-12
Editing the Property of an Attribute	18-12

Working with User-Defined Attributes	18-12
Working with Properties of Administrative Nodes.....	18-13
Managing Users	18-13
Getting All Users.....	18-14
Creating a User	18-14
Creating a Supplier User	18-15
Saving a User to a New User	18-15
Checking for Expired Passwords.....	18-16
Configuring User Settings	18-16
Deleting a User	18-17
Managing User Groups	18-17
Getting All Users Groups.....	18-18
Creating a User Group	18-18
Listing Users in a User Group	18-19

Chapter 19 Handling Exceptions

About Exceptions.....	19-1
Exception Constants	19-1
Getting Error Codes.....	19-2
Getting Error Messages	19-2
Disabling and Enabling Warning Messages.....	19-2
Checking if an APIException is a Warning and not an Error.....	19-3
Delete Warnings Disabled Automatically by the Agile API	19-4
Saving and Restoring the State of Enabled and Disabled Warnings	19-4

Part II, Developing Agile Extensions

Chapter 20 Developing Process Extensions

About Process Extensions	20-1
Developing Custom Autonumber Sources	20-2
Defining a Custom Autonumber Source	20-2
Packaging and Deploying a Custom Autonumber Source	20-3
Configuring Custom Autonumber Sources in the Agile Java Client	20-3
Assigning Autonumber Sources to a Subclass	20-4
Developing Custom Actions	20-5
Defining a Custom Action.....	20-5
Custom Actions and User Sessions.....	20-5
Packaging and Deploying a Custom Action.....	20-6
Roles and Privileges for Custom Actions	20-6
User Privileges for Configuring Process Extensions	20-6
Configuring Custom Actions in the Agile Java Client.....	20-6
Using the Process Extension Library.....	20-7
Assigning Process Extensions to Classes.....	20-8
Assigning Process Extensions to Workflow Statuses	20-8
Defining URL-Based Process Extensions.....	20-9
Passing Encoded Agile PLM Information to Other Applications	20-10
Creating an Agile PLM Session from the Target System.....	20-11
Creating an Agile Session with AuthorizationHandler	20-12
Security Handler Operation.....	20-12
Using the AIS Authorization Handler.....	20-12
Retrieving an Agile PLM Object from an HTTP Request.....	20-12

Identifying Attributes for Agile PLM Classes.	20-13
Creating an External Report	20-14
Deploying Process Extensions in a Clustered Environment.	20-15
Process Extensions FAQ.	20-15

Chapter 21 Developing Web Service Extensions

About Web Service Extensions	21-1
Key Features	21-2
WSX Architecture.	21-2
About Web Services	21-3
Web Services Architecture	21-3
Security	21-4
Tools.	21-4
Further Information about Web Services.	21-5
Developing and Deploying a Web Service	21-5
About Deployment Descriptors	21-5
Reserved Web Service Names	21-6
Using a Web Service.	21-6
Defining a Web Service Entry Point	21-6
Authenticating Users	21-6
Building and Deploying the MyFirstWebService Sample.	21-7
Downloading Tools to Build the Sample	21-7
Installing the Java SDK.	21-7
Installing Ant	21-8
Building the MyFirstWebService Sample.	21-8
About Web Service Clients	21-9
Client Programming Languages	21-9
Accessing a Web Service.	21-10
Creating MyFirstClient	21-10
Generating the SOAP Request.	21-10
Submitting the SOAP Request.	21-11
Processing the SOAP Response	21-11
Running MyFirstClient	21-12
Microsoft .NET Interoperability.	21-12
Web Service Extensions FAQ.	21-12

Part III, Reference

Chapter 22 Mapping Agile PLM Client Features to the Agile API

Login Features	22-1
General Features.	22-1
Search Features.	22-2
Attachment Features	22-2
Workflow Features	22-3
Manufacturing Site Features	22-3
Folder Features	22-3
Program Features.	22-4
Administrative Features.	22-4

PREFACE

The Agile documentation set includes Adobe® Acrobat™ PDF files. The [Oracle Technology Network \(OTN\) Web site](#) contains the latest versions of the Oracle|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 Oracle|Agile Documentation folder available on your network from which you can access the Oracle|Agile documentation (PDF) files.

To read the PDF files, you must use the free Adobe Acrobat Reader™ version 7.0 or later. This program can be downloaded from the www.adobe.com.

The [Oracle Technology Network \(OTN\) Web site](#) can be accessed through **Help > Manuals** in both the Agile Web Client and the Agile Java Client. If applicable, earlier versions of Oracle|Agile PLM documentation can be found on the www.agile.com/support.

If you need additional assistance or information, please contact support@agile.com or phone (408) 284-3900 for assistance.

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

Readme

Any last-minute information about Oracle|Agile PLM can be found in the Readme file on the [Oracle Technology Network \(OTN\) Web site](#).

Agile Training Aids

Go to the [Agile Training Web page](#) for more information on Agile Training offerings.

CHAPTER 1

Introduction

This chapter introduces you to the Agile SDK's features. It contains the following topics:

- ❑ *What is the Agile SDK?*
 - ❑ *What's New for this Release*
 - ❑ *System Requirements*
 - ❑ *Java Requirements*
 - ❑ *Agile SDK Installation Folders*
 - ❑ *Checking Your Agile PLM System*
 - ❑ *Agile PLM Business Objects*
 - ❑ *Licensing*
-

What is the Agile SDK?

The Agile SDK is a software development kit that contains a collection of tools, application programming interfaces (APIs), sample applications, and documentation. It has the following three modules:

- ❑ **Agile API** — A Java API with interfaces that expose Agile business objects. The Agile API can be used to create additional Agile PLM clients, or it can be used as part of an extension developed using WSX or PX.
- ❑ **Process extensions (PX)** — A framework that allows Agile PLM customers to extend the functionality of Agile PLM Clients by adding external reports, user-driven and workflow-driven custom actions, custom tools, and custom autonumber sources.
- ❑ **Web service extensions (WSX)** — A framework that allows Agile PLM customers to extend the functionality of the Agile PLM server and expose customer-specific solutions using web services.

The Agile SDK lets you build custom applications that access Agile Application Server functionality. By using the Agile SDK, you can create programs that perform tasks automatically against the Agile PLM system.

The Agile SDK lets you do the following:

- ❑ Integrate the Agile PLM system with ERP applications or other custom applications.
- ❑ Extend the functionality of the Agile PLM system.
- ❑ Develop applications to process product data.
- ❑ Perform batch operations against the Agile Application Server.

Client-Side Components

The Agile SDK contains the following client-side components:

- ❑ *Agile SDK Developer Guide* (this manual)

- ❑ Agile API HTML reference
- ❑ Sample applications
- ❑ Agile API library (AgileAPI.jar)
- ❑ Process Extensions API library (pxapi.jar)
- ❑ Apache Axis library (axis.jar)

Server-Side Components

The Agile Application Server contains the following Agile SDK server-side components:

- ❑ Agile API implementation classes
- ❑ Process extensions framework
- ❑ Web service extensions framework

Architecture

The Agile SDK facilitates the development of many types of programs that can 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. The web server used for WSX can be located in the company's DMZ or not. If you are using PX to develop process extensions to Agile PLM clients, you may connect to the server directly or through a proxy web server. 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 and PX extensions can also use the Agile API. It's a tool available to all Agile SDK development projects. Of course, you can also develop extensions using APIs provided by other companies.

Agile XML (also known as aXML)

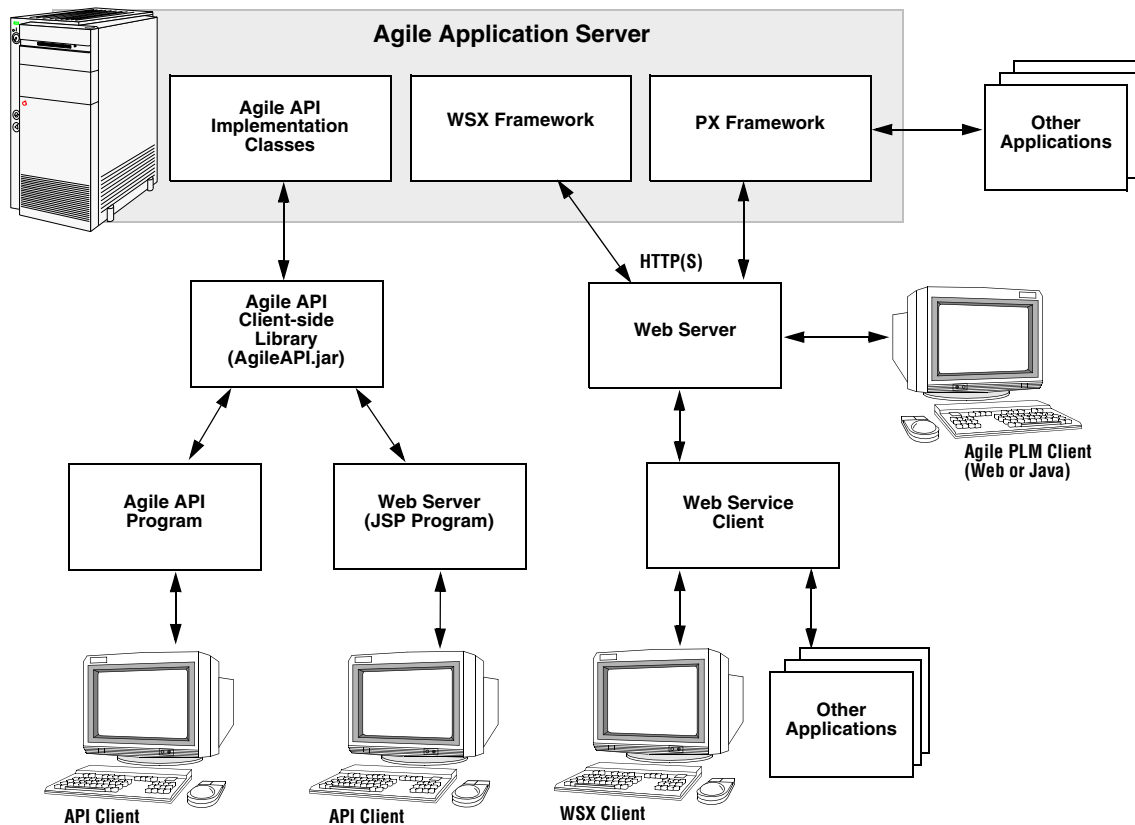
Agile XML format is an XML representation of Agile's business schema. aXML contains all product content managed in Agile including items, change details, manufacturer information, cost, drawings and other files. As a representation of schema elements across all Agile products, aXML will evolve with Agile's business schema over time.

For the latest aXML schema, please see the following Web page:

<http://support.agile.com/misc/axml/2006/03/>

Note There were changes in the aXML schema from Agile 9.2 to Agile 9.2.1. For details, see the Agile 9.2.1 Readme.

Figure 1-1: 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.

What's New for this Release

The Agile SDK supports the following new features:

- ❑ **Support for new IPC Declaration classes** – The following constants were added to **DeclarationConstants**:
 - **CLASS_IPC_1752_1_DECLARATIONS_CLASS**
 - **CLASS_IPC_1752_1_DECLARATION**
 - **CLASS_IPC_1752_2_DECLARATIONS_CLASS**
 - **CLASS_IPC_1752_2_DECLARATION**
- ❑ **Adding compositions and composition substances into a Manufacturer Part** – This feature allows you to modify the Compositions and Substances tables of a Manufacturer Part directly without submitting a declaration. See “Adding Substance Compositions for a Manufacturer Part” on page 15-17.
- ❑ **Support for file folder subclasses** – When you add a file folder to the Attachments table of a business object, you can specify which file folder subclass to use. See “Specifying the File Folder Subclass When Adding Attachments” on page 10-9.
- ❑ **Deep cloning of attachments and files** – You can use the `CommonConstants.MAKE_DEEP_COPY` virtual attribute with `ITable.createRow()` to clone an attachment row and its file from one object to another, storing a new copy of the file in the Agile File Manager vault. See “Deep Cloning Attachments and Files from One Object to Another” on page 10-8.

- ❑ **New URL to access web services** – To access an Agile PLM web service, use the following URL:
<http://<hostname>:<port>/<virtualPath>/integration/ws/<WebServiceName>>

System Requirements

For Agile SDK system requirements, please see the *Agile PLM Capacity Planning and Deployment Guide*.

Java Requirements

The Agile API is dependent on the version of Java that the application server supports. To avoid problems, an Agile API client should use the same version of Java as the application server to which it connects. Both Oracle Application Server 10g and BEA WebLogic Server 8.1 use Sun Java Runtime Environment (JRE) 1.4.2.

Agile SDK Installation Folders

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

- ❑ **lib** — The \<agile_home>\integration\sdk\lib folder contains the following libraries:
- **AgileAPI.jar** – Agile API library, which contains Agile API classes and interfaces.
 - **pxapi.jar** – PX API library, which contains interfaces used to develop custom autonumber sources and custom actions.
 - **axis.jar** – An Agile-modified version of the Apache Axis library, which is needed for web service clients.

Note When you install this manual, you can also install the Agile API HTML reference and sample programs that demonstrate how to use the APIs.

Checking Your Agile PLM System

Before trying to run Agile SDK clients on your Agile PLM system, make sure the system is configured properly and working. In particular, make sure the HTTP ports for your application server are set correctly. For more information, see the Agile PLM installation guide.

Agile PLM Business Objects

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

Table 1-2: Agile PLM business objects

Object	Related Agile API Interface
Change	IChange
Customer	ICustomer
Declaration	IDeclaration
Discussion	IDiscussion
File Folder	IFileFolder
Item	IItem
Manufacturer	IManufacturer
Manufacturer part	IManufacturerPart
Package	IPackage
Part Group (Commodity or Part Family)	ICommodity
Price	IPrice
Product Service Request	IServiceRequest

Table 1-2: Agile PLM business objects (continued)

Object	Related Agile API Interface
Program	IProgram
Project	IProject
Quality Change Request	IQualityChangeRequest
Request for Quote (RFQ)	IRequestForQuote
RFQ Response	ISupplierResponse
Site	IManufacturingSite
Specification	ISpecification
Substance	ISubstance
Supplier	ISupplier
Transfer Order	ITransferOrder

Agile PLM administrators work with additional administrative objects, such as users, user groups, administrative nodes, and Agile PLM classes.

The business objects you can view and the actions you can perform are determined by the server components installed on the Agile Application Server and the licenses, roles, and privileges you are granted to access those components. Privileges can vary from field to field.

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

Licensing

Agile Software Corporation (Agile) requires any company or individual writing code to the Agile SDK to legally obtain an Agile SDK license. The Agile SDK license grants the licensee the right to use the Agile SDK in a design environment and to freely distribute the Agile SDK libraries (such as AgileAPI.jar) with any application written by the licensee that makes calls to the Agile SDK. The Agile SDK license prohibits the distribution of the Agile SDK documentation, sample code, and source code to any other party that has not legally obtained an Agile SDK license. It also explicitly prohibits the development of competing applications.

PART I

Using the Agile API

Part I provides information about how to use the Agile API. It includes the following chapters:

- ❑ *Chapter 2, Getting Started with the Agile API*
 - ❑ *Chapter 3, Creating and Loading Queries*
 - ❑ *Chapter 4, Working with Tables*
 - ❑ *Chapter 5, Working with Data Cells*
 - ❑ *Chapter 6, Working with Folders*
 - ❑ *Chapter 7, Working with Items, BOMs, and AMLs*
 - ❑ *Chapter 8, Working with Lists*
 - ❑ *Chapter 9, Managing Manufacturing Sites*
 - ❑ *Chapter 10, Working with Attachments and File Folders*
 - ❑ *Chapter 11, Managing Workflow*
 - ❑ *Chapter 12, Product Cost Management*
 - ❑ *Chapter 13, Managing and Tracking Quality*
 - ❑ *Chapter 14, Creating and Managing Programs*
 - ❑ *Chapter 15, Product Governance & Compliance*
 - ❑ *Chapter 16, Subscribing to Agile PLM Objects*
 - ❑ *Chapter 17, Creating Multithreaded Applications*
 - ❑ *Chapter 18, Performing Administrative Tasks*
 - ❑ *Chapter 19, Handling Exceptions*
-

CHAPTER 2

Getting Started with the Agile API

This chapter shows how to get started creating an Agile API program. It contains the following topics:

- ❑ *Agile API Overview*
 - ❑ *Starting an Agile API Program*
 - ❑ *Loading and Creating Agile PLM Objects*
 - ❑ *Saving an Object to a New Object*
 - ❑ *Sharing an Object*
 - ❑ *Deleting and Undeleting Objects*
 - ❑ *Closing a Session*
-

Agile API Overview

This section provides an overview of the functionality provided by the Agile API. It describes

- ❑ Types of Agile API classes interfaces
- ❑ How classes are loaded
- ❑ How the Agile API is thread-safe
- ❑ How you should package your Agile API applications
- ❑ Where to find sample programs

Types of Agile API Classes and Interfaces

The Agile API contains many different classes and interfaces in the AgileAPI.jar library. To understand them better, it helps to further classify the files into the following groups:

- ❑ **Aggregate interfaces** — These interfaces aggregate the relevant functional interfaces for a particular object type. For example, the `IItem` interface extends `IDataObject`, `IRevised`, `IManufacturingSiteSelectable`, `IAttachmentContainer`, `IHistoryManager`, and `IReferenced`. Most of the SDK functionality falls 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; several other interfaces, 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 can be routed 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 like `IAgileClass`, `INode`, `IRoutableDesc`, `ITableDesc`, and `IWorkflow`.

- ❑ **Factory classes** — `AgileSessionFactory` is a factory class that you use 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

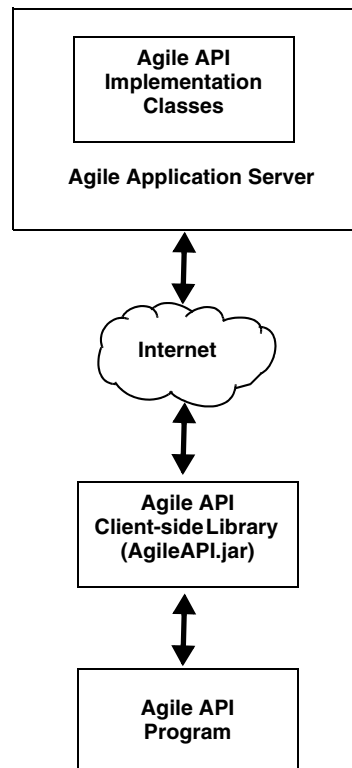
The Agile API has two main software components:

- ❑ the 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 2-1: 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 Multithreaded Applications

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

For more information about multithreading, see Chapter 17, “Creating Multithreaded Applications.”

Packaging Your Agile API Programs

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.

Agile API Files You Are Allowed to Distribute

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 might require you to distribute other class files or libraries with your program. Check the documentation for your development environment to see which runtime files should be distributed with your program. Consult the manufacturer’s license agreement for each of the files you plan to distribute to determine whether you have the right to distribute the file with your application.

Agile API Files You Are Not Allowed to Distribute

Agile requires that any company or individual writing code to the Agile API must obtain an Agile SDK license. The Agile SDK license grants the right to use the Agile API in a design environment and to freely distribute the `AgileAPI.jar` with any application that makes calls to the API. The Agile SDK license explicitly prohibits distribution of the following files to any other party that has not legally obtained an Agile SDK license:

- ☐ Agile SDK documentation
- ☐ Sample code provided with the Agile SDK
- ☐ Source code

Note The above list is not intended to be a complete list of Agile SDK files you are not allowed to distribute. For complete information, consult your Agile software license agreement.

Sample Programs

The Agile SDK provides several sample programs that demonstrate how to use its APIs. When you install this manual, the sample programs are also installed into the `\AgileDocsHome\integration\sdk\samples` folder.

Each sample program has its own Readme.txt file. Make sure you read the Readme.txt file 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 won't be 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 full package name, for example:

```
com.agile.api.IItem source =  
(com.agile.api.IItem)m_session.getObject(com.agile.api.IItem.OBJECT_TYPE, "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.

If you don't import the `com.agile.api` package or reference 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

To start an Agile API program, you must do 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 virtual path is *agile*. The virtual path is specified when the Agile PLM system is installed. For more information, see 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 2-1: 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);
}
```

Note Your Agile PLM license key determines the maximum number of concurrent sessions you can have open to the Agile Application Server per user account. If you try to exceed the maximum number of sessions, the server prevents you from logging in. Therefore, it's 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 Server, 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 `AgileSessionFactory.getInstance()` first to obtain a server instance. The `createSessionEx()` method obtains the server instance and creates the session in one call.

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

Example 2-2: Creating a session using a password-protected server 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);  
}
```

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 Chapter 21, "Developing Web Service Extensions."

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

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

Note For more information about creating queries and folders, see Chapter 3, “Creating and Loading Queries” and Chapter 6, “Working with Folders.”

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)`

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, Agile doesn’t 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 18-8.

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” on page 20-13.

Example 2-3 shows how to load part 1000-02 using a `Map` parameter that specifies an attribute (`ItemConstants.ATT_TITLE_BLOCK_NUMBER`) and a value.

Example 2-3: 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.

Example 2-4 shows how to load part 1000-02 by specifying a unique `String` identifier.

Example 2-4: 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 several different ways to load various types of Agile PLM objects.

Example 2-5: Loading different types of 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_MANUFACTURER_NAME, "World Enterprises");
params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_PART_NUMBER, "WE10023-45");
IManufacturerPart mfrPart =
    (IManufacturerPart)m_session.getObject(IManufacturerPart.OBJECT_TYPE, params);
System.out.println("ManufacturerPart : " + mfrPart.getName());

//Load a manufacturing site
IManufacturingSite siteHK = (IManufacturingSite)m_session.getObject(
    ManufacturingSiteConstants.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 a sourcing project
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)`

The `objectType` and `params` parameters are identical to those used in the `IAgileSession.getObject()` methods; for more information, see “Loading Objects” on page 2-7. 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 can’t specify an `INode` object for the `params` parameter for objects that don’t have 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 2-6: 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 2-7: 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 2-8: 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 2-9: 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.

Example 2-10: 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

Each Agile PLM class can have one or more `AutoNumber` sources. An `AutoNumber` source is a predefined number series used to automatically number objects. `AutoNumber` sources are defined in the administrative functionality of the Agile Java Client.

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

Your Agile Application Server can be configured to require the use of AutoNumbers when you create an object of a particular class. To determine if automatic numbering is required for an object, use the `IAgileClass.isAutoNumberRequired()` method. However, the method is marked as deprecated.

The `isAutoNumberRequired()` method is deprecated because the Agile API doesn't enforce automatic numbering of objects even when it's required for a particular class. This is something the client developer must handle. Consequently, if you develop a GUI program that allows users to create Agile PLM objects, you should make sure the user interface enforces automatic numbering when it's required. For an example of how a client program enforces automatic numbering, create some objects in the Agile Web Client and note how the user interface works.

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

Example 2-11: Getting the next AutoNumber

```
private void createPart(String partNumber) throw APIException {

    IAdmin admin;
    IAgileClass cls;
    IItem part;
    IAutoNumber[] numSources;
    String nextAutoNumber;

    //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 AutoNumber using the first autonumber source
        nextAutoNumber = numSources[0].getNextNumber();

        // Create the part using the AutoNumber
        part = (IItem)session.createObject(ItemConstants.CLASS_PART, nextAutoNumber);
    } 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;
}
```

Handling Required Fields

A class can be defined with several required attributes. To make a particular attribute required, the Agile PLM administrator sets the Visible and Required properties for the attribute to Yes. If you try to create an object in the Agile Java Client or the Agile Web Client without completing all required fields, the client does not allow you to save the object until you set 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 regardless if values aren't set for all required fields. If you want to enforce required fields in your client program and make it behave like the Agile Web Client and Java Client, 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 2-12: 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 2-13: 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_COMMODITY,
        "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_DECLARATION, 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_DISCUSSION, params);
System.out.println("Discussion : " + discussion.getName());

//Create a file folder
IFileFolder ff = (IFileFolder)m_session.createObject(
    FileFolderConstants.CLASS_FILE_FOLDER, "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_MANUFACTURER, 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(
    ManufacturingSiteConstants.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(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE);
I AgileList 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.CLASS_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_SUBSTANCE,
    "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_GROUP,
params);
System.out.println("UserGroup : " + group.getName());

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

Note You cannot use the Agile API to create the following product sourcing objects: Project, RequestForQuote (RFQ), SupplierResponse, and Supplier.

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 nonconformance 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`.

Example 2-14 shows how to save the current object to a new object using the next `AutoNumber` for the specified subclass.

Example 2-14: Saving an object as a new object

```
private void saveAsObject(IDataObject obj, IAgileClass sub) {

    String nextNum;

    try {
        // Get the next autonumber for the subclass
        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 2-15: 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, "albert1");
    IUser user2 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "peter1");
    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 the 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
- ❑ A released change
- ❑ A manufacturer with one or more manufacturer parts
- ❑ A manufacturer part currently used on the Manufacturers tab of another object

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 2-16: 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_WARNING);
                // 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.

Example 2-17 shows how to undelete an object that has been deleted.

Example 2-17: 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.undelele();  
    }  
}
```

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 2-18: Closing a session

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


CHAPTER 3

Creating and Loading Queries

This chapter shows how to create queries, or searches, in Agile API programs. It contains the following topics:

- ❑ *About Queries*
- ❑ *Creating a Query*
- ❑ *Specifying Search Criteria*
- ❑ *Using SQL Syntax for Search Criteria*
- ❑ *Setting Result Attributes for a Query*
- ❑ *Working with Query Results*
- ❑ *Creating a Where-Used Query*
- ❑ *Loading a Query*
- ❑ *Deleting a Query*
- ❑ *Simple Query Example*

About Queries

An `IQuery` is an object that defines how to search for Agile PLM data. It defines a search like the searches you can use in the Agile Web Client. The search can have multiple search criteria (like an Advanced Search in the 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 to use for the search. In the following example, the query class is the `Item` class.

Example 3-1: 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 3-2: 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 3-3: 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 3-4: 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 automatically deletes temporary queries after the user session that created the query is closed.

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, perhaps at runtime. Parameters provide a handy way to pass values to a query, and they can save time and extra coding. Parameterized queries can be saved and reused later.

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 3-5: 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 3-6: 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;
}
```

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 don't exist in the Agile PLM database, but that can be used to define the query at run time.

You can also specify queries based on Workflows. For details on how to specify a Workflow-related query, see "Specifying Search Criteria" on page 3-4.

Table 3-1: Query attribute constants

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 3-7: 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 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 workflows, but the nodes can also be used as ordinary search criteria.
- ❑ `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.”

Here is an example of a search condition:

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

Here 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:

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

Query language keywords are not localized. You must use English keywords, regardless of locale. You can use them in lower case or upper case.

Note In addition to keywords, Agile PLM variables such as \$USER (for current user) and \$TODAY (for today's date) can be used in Agile API queries.

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 isn't enclosed within brackets, your query will fail.

You can specify a search attribute in the following ways:

Table 3-2: Ways to specify search attributes

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.

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

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

Retrieving Searchable Attributes

The attributes that are searchable for a query depend on the query class or subclass that you specified. 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—that is, 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 with the Agile SDK. However, you must search for the Table name that corresponds with the Tab name.

Note Because you use the table name to setup the IQuery, it does not matter if an Agile administrator changes a Tab name from the name specified in the 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 3-12.

Using Relational Operators

The following table shows the relational operators the Agile API query language supports:

Table 3-3: Relational operators

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		Finds any value that includes the specified value.
does not contain		Finds any value that does not include the specified value.
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.
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 nonprintable 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'
```

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 clients such as the Agile Web Client. 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.

Table 3-4: Using between and in operators

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')
```

Using Any, All, None Of, and Not All Relational Operator Modifiers

For other than Page One, Page Two, and Page Three attributes, the 'all,' 'any,' 'none of,' and 'not all' relational operator modifiers can be used to further define a search condition. These relational operator modifiers are particularly helpful if your search uses attributes from the BOM, AML, or Attachments tables.

- ☐ any — any row in the table matches the search criteria
- ☐ all — all rows in the table match the search criteria
- ☐ none of — no rows in the table match the search criteria
- ☐ not all — not all rows in the table match the search criteria

Relational operator modifiers must be used at the start of the search condition. If you don't specify one of the four relational operator modifiers, 'any' is assumed. If you use relational operator modifiers with a Page One, Page Two, or Page Three search attribute, the search condition is ignored. The following table shows several search conditions that use 'any,' 'all,' 'none of,' and 'not all' relational operator modifiers.

Table 3-5: Search conditions that use Any, All, None Of, and Not All relational operator modifiers

Search Condition	Finds
any [Manufacturers.Mfr. Name] contains ('Motorola')	Items in which any AML row has a manufacturer name that contains "Motorola."
all [Manufacturers.Mfr. Name] contains ('Motorola')	Items in which all AML rows have a manufacturer name that contains "Motorola."
none of [Manufacturers.Mfr. Name] contains ('Motorola')	Items in which none of the AML rows has a manufacturer name that contain "Motorola."
not all [Manufacturers.Preferred Status] equal to ('Preferred')	Items in which not all AML rows have a preferred status equal to "Preferred".
any [BOM.Item Description] contains ('Power Supply')	Items in which any BOM row has an item description that contains "Power Supply".
all [BOM.Item Number] starts with ('IC')	Items in which all BOM rows have an item number that starts with "IC".
none of [BOM.Item Number] equal to ('1000-02')	Items in which none of the BOM rows have an item number equal to "1000-02".
not all [BOM.Qty] equal to ('1')	Items in which not all BOM rows have a quantity value equal to 1.
all [Title Block.Description] contains ('Motor')	No results. The search condition is ignored because it incorrectly uses a relational operator modifier with a Page One search attribute.

Note Queries that use relational operator modifiers have a high performance cost and may be noticeably slower than other queries. For that reason, they should be used infrequently.

Using 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 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 finds 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 finds 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')
```

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 uses 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.

Table 3-6: Searching 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 the Agile Web Client, choose **Settings > User Profile** and then click the **Preferences** tab.

Example 3-8: 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" } );
```

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

Example 3-9: 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 3-18.

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 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).

Here 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`

For readability, it’s recommended that SQL key words such as SELECT and FROM are entered in all 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 3-10: Query using standard Agile API query syntax

```
try {
    IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE, "Items");
    query.setCriteria("[Page Two.Numeric01] 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);
}
```

Example 3-11 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 3-11: 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 3-3.

Example 3-12: 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.

Table 3-7: ORDER BY examples

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 3-13: 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.

Table 3-8: Default query result attributes

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
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

Table 3-8: Default query result attributes (continued)

Query class	Default result attributes
Items Parts Part Documentation Document	Title Block.Item Type Title Block.Number Title Block.Description Title Block.Lifecycle Phase Title Block.Rev
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.Program 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
Programs Activities Program Phase 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

Table 3-8: Default query result attributes (continued)

Query class	Default result attributes
Projects Sourcing Projects Sourcing Project	General Info.Project Type General Info.Number General Info.Description General Info.Manufacturing Site General Info.Ship To Location General Info.Program 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
RFQs RFQs RFQ	Cover Page.RFQ Number Cover Page.RFQ Description Cover Page.MFG Site Cover Page.Ship-To Location Cover Page.Program 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

Table 3-8: Default query result attributes (continued)

Query class	Default result attributes
Suppliers	General Info.Supplier Type
Suppliers	General Info.Number
Component Manufacturer	General Info.Name
Contract Manufacturer	General Info.Description
Distributor	General Info.Status
Manufacturer Rep	
Transfer Orders	Cover Page.Transfer Order Type
Content Transfer Orders	Cover Page.Transfer Order Number
CTO	Cover Page.Description
Automated Transfer Orders	Cover Page.Status
ATO	Cover Page.Workflow

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 3-14: 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 3-15: 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(true)`, as shown in the following example.

Example 3-16: Getting the array of available result attributes

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

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, say 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 perhaps 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 4-15.

Sorting Query Results

Unlike other Agile API tables, you can’t 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 3-10.

Datatype of Query Results

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 were strings. In Agile 9.2 these values are now 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 the 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 nearly identical 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 can be defined only 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.

Table 3-9: Where-used set operators

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 3-17: 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_RELEASED);

// 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.

Note Query names are case-sensitive. Make sure you type query names in the proper case.

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

Example 3-18: 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 3-19: 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 query that has been saved, use the `IQuery.delete()` method.

Temporary queries, that is, queries that have been 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.

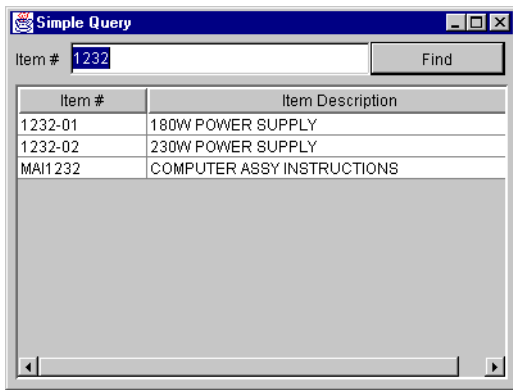
Example 3-20: Deleting a query

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

Simple Query Example

Figure 3-1 shows an example of dialog box that performs a simple query.

Figure 3-1: 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. Example 3-21 shows the code that runs the query when the user clicks the **Find** button.

Example 3-21: 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 shows how to work with Agile API tables. It contains the following topics:

- ❑ *About Tables*
 - ❑ *Retrieving a Table*
 - ❑ *Retrieving the Metadata of a Table*
 - ❑ *Adding Table Rows*
 - ❑ *Adding and Updating Multiple Table Rows*
 - ❑ *Iterating Over Table Rows*
 - ❑ *Sorting Tables*
 - ❑ *Removing Table Rows*
 - ❑ *Retrieving the Referenced Object for a Row*
 - ❑ *Checking Status Flags of a Row*
 - ❑ *Working with Page One, Page Two, and Page Three*
 - ❑ *Redlining*
 - ❑ *Removing Redlines*
-

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 the Agile Web Client, these tables are equivalent to the separate tabs in a window, such as the Manufacturers and BOM tabs.

Note In some cases, a tab in the Agile Web Client may contain multiple tables. For example, the Changes tab for an item contains the Pending Changes table and the Change History table.

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

Figure 4-1: BOM tab for an item

The screenshot shows the Agile Web Client interface with the BOM tab selected. The table displays the following data:

Item Number	Item Description	Item Rev	Qty
MA11232	Computer Assy Instructions	A 24465	REF
MT11000	Computer Test Instructions	A 24465	REF
QAT2321	Computer FCC Test Results	A 24465	REF
1543-01		B SCO_0002	1
9876-01	IDE Hard Disk Controller		1
7654-02	2.1GB IDE Hard Disk		1
6598-01	DC Power Cable	A 23450	3
6642-01	3.5" Floppy Disk Drive		1
8768-01	2MB Video Card		1
2543-01	2X Speed CD-ROM Drive		1

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`.

Table 4-1: `ITable` methods inherited from `java.util.Collection`

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

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`.

Tables vary for each Agile PLM dataobject. Tables for change objects are different from the 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. The following table lists the Agile PLM tables for each dataobject.

Table 4-2: Agile PLM tables

Class/Subclass	Tables	Table Constants	Read/Write Mode
IChange object (ChangeConstants)			
Change Orders ECO	Affected Items Affected Prices	TABLE_AFFECTEDITEMS TABLE_AFFECTEDPRICES	Read/Write Read/Write
Change Requests ECR	Attachments Cover Page	TABLE_ATTACHMENTS TABLE_COVERPAGE	Read/Write Read/Write
Deviations Deviation	History Page Three	TABLE_HISTORY TABLE_PAGETHREE	Read-only Read/Write
Manufacturer Orders MCO	Page Two Relationships Affected By	TABLE_PAGETWO TABLE_RELATIONSHIPSaffectedBY	Read/Write Read/Write
Sites Change Orders SCO	Relationships Affects Relationships References	TABLE_RELATIONSHIPSaffects TABLE_RELATIONSHIPSREFERENCES	Read/Write Read/Write
Stop Ships Stop Ship	Workflow	TABLE_WORKFLOW	Read-only
Price Change Orders PCO			
ICommodity object			
Part Groups Commodity Part Family	Attachments Compositions General Info History Page Three Page Two Parts Specifications Substances Suppliers	TABLE_ATTACHMENTS TABLE_COMPOSITIONS TABLE_GENERALINFO TABLE_HISTORY TABLE_PAGETHREE TABLE_PAGETWO TABLE_CONTENTS TABLE_COMPLIANCE TABLE_SUBSTANCES TABLE_SUPPLIERS	Read/Write Read-only Read/Write Read-only Read/Write Read/Write Read/Write Read/Write Read-only Read/Write
ICustomer object (CustomerConstants)			
Customers Customer	Attachments General Info History Page Three Page Two Related PSR	TABLE_ATTACHMENTS TABLE_GENERALINFO TABLE_HISTORY TABLE_PAGRETHREE TABLE_PAGETWO TABLE_RELATEDPSR	Read/Write Read/Write Read-only Read/Write Read/Write Read-only

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
IDeclaration object			
Declarations	Attachments	TABLE_ATTACHMENTS	Read/Write
Homogeneous Material Declaration	Cover Page	TABLE_COVERPAGE	Read/Write
IPC 1752-1 Declaration	History	TABLE_HISTORY	Read-only
IPC 1752-2 Declaration	Item Composition	TABLE_ITEMCOMPOSITION	Read/Write
JGPSSI Declaration	Items	TABLE_ITEMS	Read/Write
Part Declaration	Manufacturer Parts	TABLE_MANUFACTURERPARTS	Read/Write
Substance Declaration	Page Three	TABLE_PAGETHREE	Read/Write
Supplier Declaration of Conformance	Page Two	TABLE_PAGETWO	Read/Write
	Part Groups	TABLE_PARTGROUPS	Read/Write
	Manufacturer Part Composition	TABLE_MANUFACTURERPARTCOMPOSITION	Read/Write
	Part Group Composition	TABLE_PARTGROUPCOMPOSITION	Read/Write
	References	TABLE_REFERENCES	Read/Write
	Relationships Affected By	TABLE_RELATIONSHIPSaffectedBY	Read/Write
	Relationships Affects	TABLE_RELATIONSHIPSaffects	Read/Write
	Specifications	TABLE_SPECIFICATIONS	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only
IDiscussion object (DiscussionConstants)			
Discussions	Attachments	TABLE_ATTACHMENTS	Read/Write
Discussion	Action Items	TABLE_ACTIONITEMS	Read/Write
	Cover Page	TABLE_COVERPAGE	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two (Details)	TABLE_PAGETWO	Read/Write
	Replies	TABLE_REPLIES	Read/Write
	Where Used	TABLE_WHEREUSED (Not Supported)	Read-only
IFileFolder object (FileFolderConstants)			
File Folders	Files	TABLE_FILES	Read/Write
File Folder	History	TABLE_HISTORY	Read-only
Historical Reports	Page Three	TABLE_PAGETHREE	Read/Write
User Saved	Page Two	TABLE_PAGETWO	Read/Write
Schedule Generated	Relationships Affected By	TABLE_RELATIONSHIPSaffectedBY	Read-only
	Relationships Affects	TABLE_RELATIONSHIPSaffects	Read-only
	Relationships References	TABLE_RELATIONSHIPSREFERENCES	Read/Write
	Title Block	TABLE_TITLEBLOCK	Read/Write
	Where Used	TABLE_WHEREUSED	Read-only
	Workflow	TABLE_WORKFLOW	Read-only

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
IItem object (ItemConstants)			
Parts	Attachments	TABLE_ATTACHMENTS	Read/Write
Part	BOM	TABLE_BOM	Read/Write
Documentation Document	Change History	TABLE_CHANGEHISTORY	Read-only
	Compositions	TABLE_COMPOSITIONS	Read-only
	History	TABLE_HISTORY	Read-only
	Manufacturers	TABLE_MANUFACTURERS	Read/Write
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Pending Change Where Used	TABLE_PENDINGCHANGEWHEREUSED	Read-only
	Pending Changes	TABLE_PENDINGCHANGES	Read-only
	Prices	TABLE_PRICES	Read/Write
	QCRs	TABLE_QCRS	Read-only
	Quality	TABLE_QUALITY	Read-only
	Redline BOM	TABLE_REDLINEBOM	Read/Write
	Redline Manufacturers	TABLE_REDLINEMANUFACTURERS	Read/Write
	Sites	TABLE_SITES	Read/Write
	Specifications	TABLE_COMPLIANCE	Read/Write
	Substances	TABLE_SUBSTANCES	Read-only
	Suppliers	TABLE_SUPPLIERS	Read/Write
	Title Block	TABLE_TITLEBLOCK	Read/Write
	Where Used	TABLE_WHEREUSED	Read-only
IManufacturerPart object (ManufacturerPartConstants)			
Manufacturer Parts	Attachments	TABLE_ATTACHMENTS	Read/Write
Manufacturer Part	Compositions	TABLE_COMPOSITIONS	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Pending Change Where Used	TABLE_PENDINGCHANGEWHEREUSED	Read-only
	Prices	TABLE_PRICES	Read-only
	Specifications	TABLE_SPECIFICATIONS	Read/Write
	Substances	TABLE_SUBSTANCES	Read/Write
	Suppliers	TABLE_SUPPLIERS	Read/Write
	Where Used	TABLE_WHEREUSED	Read-only
IManufacturer object (ManufacturerConstants)			
Manufacturers	Attachments	TABLE_ATTACHMENTS	Read/Write
Manufacturer	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Where Used	TABLE_WHEREUSED	Read-only

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
IManufacturingSites object (ManufacturingSiteConstants)			
Sites Site	Attachments	TABLE_ATTACHMENTS	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
IPackage object (PackageConstants)			
Packages Package	Attachments	TABLE_ATTACHMENTS	Read/Write
	Cover Page	TABLE_COVERPAGE	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only
IPrice object (PriceConstants)			
Published Prices Contracts Published Price	Attachments	TABLE_ATTACHMENTS	Read/Write
	Change History	TABLE_CHANGEHISTORY	Read-only
	History	TABLE_HISTORY	Read-only
Quote Histories Quote History	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Pending Changes	TABLE_PENDINGCHANGES	Read-only
	Price Lines	TABLE_PRICELINES	Read/Write
	Redline Price Lines	TABLE_REDLINEPRICELINES	Read/Write
	Title Block	TABLE_TITLEBLOCK	Read/Write
IProgram (ProgramConstants)			
Activities Phase Program Task	Action Items	TABLE_ACTIONITEMS	Read/Write
	Attachments	TABLE_ATTACHMENTS	Read/Write
	Deliverables - Affected By	TABLE_RELATIONSHIPSAFFECTEDBY	Read/Write
	Deliverables - Affects	TABLE_RELATIONSHIPSAFFECTS	Read/Write
	Dependencies - Dependent Upon	TABLE_DEPENDENCIESDEPENDENTUPON	Read/Write
	Dependencies - Required For	TABLE_DEPENDENCIESREQUIREDFOR	Read-only
	Discussion	TABLE_DISCUSSION	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	News	TABLE_NEWS	Read/Write
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Links	TABLE_REFERENCES	Read/Write
	Schedule	TABLE_SCHEDULE	Read/Write
	Team	TABLE_TEAM	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
Gates Gate	Action Items	TABLE_ACTIONITEMS	Read/Write
	Approval Items	TABLE_APPROVALITEMS	Read/Write
	Attachments	TABLE_ATTACHMENTS	Read/Write
	Deliverables - Affected By	TABLE_RELATIONSHIPSAFFECTEDBY	Read/Write
	Deliverables - Affects	TABLE_RELATIONSHIPSAFFECTS	Read/Write
	Dependencies - Dependent Upon	TABLE_DEPENDENCIESDEPENDENT-UPON	Read/Write
	Dependencies - Required For	TABLE_DEPENDENCIESREQUIREDFOR	Read-only
	Discussion	TABLE_DISCUSSION	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	News	TABLE_NEWS	Read/Write
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Links	TABLE_REFERENCES	Read/Write
	Schedule	TABLE_SCHEDULE	Read/Write
	Team	TABLE_TEAM	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only
IProject object (ProjectConstants)			
Sourcing Projects Sourcing Project	AML	TABLE_AML	Read/Write
	Analysis	TABLE_ANALYSIS	Read-only
	Attachments	TABLE_ATTACHMENTS	Read/Write
	Costed BOM Analysis	TABLE_COSTEDBOMANALYSIS	Read-only
	Discussion	TABLE_DISCUSSION	Read/Write
	General Information	TABLE_GENERALINFORMATION	Read/Write
	History	TABLE_HISTORY	Read-only
	Item Added/Deleted	TABLE_ITEMADDED_DELETED	Read-only
	Item Changed	TABLE_ITEM	Read-only
	Items	TABLE_ITEMS	Read/Write
	Nonmaterials Analysis	TABLE_NONMATERIALSANALYSIS	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Responses	TABLE_RESPONSES	Read/Write
	RFQs	TABLE_RFQS	Read/Write
	Unapplied Changes	TABLE_UNAPPLIEDCHANGES	Read-only

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
IQualityChangeRequest object (QualityChangeRequestConstants)			
Audits Audit Corrective Action/ Preventive Action CAPA	Affected Items	TABLE_AFFECTEDITEMS	Read/Write
	Attachments	TABLE_ATTACHMENTS	Read/Write
	Cover Page	TABLE_COVERPAGE	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	PSR Items	TABLE_PSRITEMS	Read/Write
	Relationships Affected By	TABLE_RELATIONSHIPSAFFECTEDBY	Read/Write
	Relationships Affects	TABLE_RELATIONSHIPSAFFECTS	Read/Write
	Relationships References	TABLE_RELATIONSHIPSREFERENCES	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only
IRequestForQuote object (RequestForQuoteConstants)			
RFQs RFQ	Attachments	TABLE_ATTACHMENTS	Read/Write
	Comment	TABLE_COMMENT	Read/Write
	Cover Page	TABLE_COVERPAGE	Read/Write
	Detail Price	TABLE_DETAILPRICE	Read/Write
	Discussion	TABLE_DISCUSSION	Read/Write
	History	TABLE_HISTORY	Read-only
	Item Change	TABLE_ITEMCHANGE	Read/Write
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Price Point	TABLE_PRICEPOINT	Read/Write
	Responses	TABLE_RESPONSES	Read/Write
	Simple Price	TABLE_SIMPLEPRICE	Read/Write
	Status	TABLE_STATUS	Read-only
IServiceRequest object (ServiceRequestConstants)			
Nonconformance Reports NCR	Affected Items	TABLE_AFFECTEDITEMS	Read/Write
	Attachments	TABLE_ATTACHMENTS	Read/Write
Problem Reports Problem Report	Cover Page	TABLE_COVERPAGE	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Related PSR	TABLE_RELATEDPSR	Read/Write
	Relationships Affected By	TABLE_RELATIONSHIPSAFFECTEDBY	Read/Write
	Relationships Affects	TABLE_RELATIONSHIPSAFFECTS	Read/Write
	Relationships References	TABLE_RELATIONSHIPSREFERENCES	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only
ISpecification object			
Specifications Specification	Attachments	TABLE_ATTACHMENTS	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Substances	TABLE_SUBSTANCES	Read/Write

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
ISubstance object			
Materials	Attachments	TABLE_ATTACHMENTS	Read/Write
Material	Composition	TABLE_COMPOSITION	Read/Write
Subparts	General Info	TABLE_GENERALINFO	Read/Write
Subpart	History	TABLE_HISTORY	Read-only
Substances Groups	Item Where Used	TABLE_ITEMWHEREUSED	Read-only
Substance Group	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Substances	TABLE_SUBSTANCES	Read/Write
Substances	Attachments	TABLE_ATTACHMENTS	Read/Write
Substance	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Item Where Used	TABLE_ITEMWHEREUSED	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Substances	TABLE_SUBSTANCES	Read/Write
ISupplierResponse object (SupplierResponseConstants)			
RFQ Responses	Attachments	TABLE_ATTACHMENTS	Read/Write
RFQ Response	Comment	TABLE_COMMENT	Read/Write
	Costed BOM	TABLE_COSTEDBOM	Read-only
	Costed BOM Price Entry	TABLE_COSTEDBOMPRICEENTRY	Read/Write
	Cover Page	TABLE_COVERPAGE	Read/Write
	Detail Price	TABLE_DETAILPRICE	Read/Write
	Discussion	TABLE_DISCUSSION	Read/Write
	History	TABLE_HISTORY	Read-only
	Item Change	TABLE_ITEMCHANGE	Read/Write
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Price Point	TABLE_PRICEPOINT	Read/Write
	Responses	TABLE_RESPONSES	Read/Write
	Simple Price	TABLE_SIMPLEPRICE	Read/Write
ISupplier object (SupplierConstants)			
Suppliers	Attachments	TABLE_ATTACHMENTS	Read/Write
Component	Commodities	TABLE_COMMODITIES	Read/Write
Manufacturer	Contact Users	TABLE_CONTACTUSERS	Read/Write
Contract Manufacturer	General Info	TABLE_GENERALINFO	Read/Write
Distributor	History	TABLE_HISTORY	Read-only
Manufacturer Rep	Manufacturers	TABLE_MANUFACTURERS	Read/Write
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	PSRs	TABLE_PSRs	Read/Write
	RFQ Routing	TABLE_RFQROUTING	Read/Write

Table 4-2: Agile PLM tables (continued)

Class/Subclass	Tables	Table Constants	Read/Write Mode
ITransferOrder object (TransferOrderConstants)			
Automated Transfer Orders ATO	Attachments	TABLE_ATTACHMENTS	Read/Write
	Cover Page	TABLE_COVERPAGE	Read/Write
Content Transfer Orders CTO	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Selected Objects	TABLE_SELECTEDOBJECTS	Read/Write
	Where Sent	TABLE_WHERESENT	Read/Write
	Workflow	TABLE_WORKFLOW	Read-only
IUser object (UserConstants)			
Users User	Attachments	TABLE_ATTACHMENTS	Read/Write
	Escalations	TABLE_ESCALATIONS	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Preferences	TABLE_PREFERENCES	Read/Write
	Share	TABLE_SHARE	Read/Write
	Subscription	TABLE_SUBSCRIPTION	Read/Write
	User Group	TABLE_USERGROUP	Read/Write
IUserGroup object (UserGroupConstants)			
User Groups User Group	Attachments	TABLE_ATTACHMENTS	Read/Write
	Escalations	TABLE_ESCALATIONS	Read/Write
	General Info	TABLE_GENERALINFO	Read/Write
	History	TABLE_HISTORY	Read-only
	Page Three	TABLE_PAGETHREE	Read/Write
	Page Two	TABLE_PAGETWO	Read/Write
	Share	TABLE_SHARE	Read/Write
	Users	TABLE_USERS	Read/Write

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

Example 4-1: 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 therefore cannot be modified. When you write code to access a table, use the `ITable.isReadOnly()` method to check if the table is read-only. Table 4-2 on page 4-3 also lists the read/write mode for each Agile PLM table.

Retrieving the Metadata of a Table

`ITableDesc` is an interface that represents the metadata of a table, 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 4-2: 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;
}
```

For more information about using the Agile API to work with metadata, see Chapter 18, "Performing Administrative Tasks."

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()` can be used 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.createRows()`. See “Adding and Updating Multiple Table Rows” on page 4-13.

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 4-3: 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 4-4: 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 4-5: 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_NUMBER, "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 4-6: 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);
}

```

Since the BOM table also references `IItem` objects, you could use code similar to Example 4-6 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 program. The code adds a row to the table and initializes it with the specified `IProgram` object.

Example 4-7: 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 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 4-8: 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 4-9: 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 an expensive operation performance-wise and therefore isn't 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 4-10: 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 Example 4-10. 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 Figure 4-2.

Figure 4-2: Navigation controls in the Agile Web Client



Sorting Tables

To sort the rows in a table by a particular attribute, use `getTableIterator(ITable.ISortBy[])` to return a sorted iterator. The `sortBy` 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 `sortBy` parameter of `getTableIterator()` must contain only one `ISortBy` object.

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

Example 4-11: 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 Program 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 3-10.

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 4-11. 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 4-12: 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 4-13: 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 4-3 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).

Table 4-3: Tables that reference other Agile PLM objects

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

Table 4-3: Tables that reference other Agile PLM objects (continued)

Object	Table	Referenced Object(s)
IDeclaration	Attachments Item Composition Items Manufacturer Part Composition Manufacturer Parts Part Group Composition Part Groups References Relationships Affected By Relationships Affects Specifications	IAttachmentFile ISubstance IItem ISubstance IManufacturerPart ISubstance ICommodity Multiple object types Multiple object types Multiple object types ISpecification
IDiscussion	Attachments Where Used	IAttachmentFile Not supported
IFileFolder	Files Relationships Affected By Relationships Affects Relationships References Where Used	IAttachmentFile Multiple object types Multiple object types 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
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

Table 4-3: Tables that reference other Agile PLM objects (continued)

Object	Table	Referenced Object(s)
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
IQualityChangeRequest	Affected Items Attachments PSR Items Relationships Affected By Relationships Affects Relationships References	IItem IAttachmentFile IItem Multiple object types Multiple object types Multiple object types
IRequestForQuote	Attachments	IAttachmentFile
IServiceRequest	Affected Items Attachments Related PSR Relationships Affected By Relationships Affects Relationships References	IItem IAttachmentFile IServiceRequest Multiple object types Multiple object types 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

Table 4-3: Tables that reference other Agile PLM objects (continued)

Object	Table	Referenced Object(s)
IUser	Attachments	IAttachmentFile
	Subscription	Multiple object types
	User Group	IUserGroup
IUserGroup	Attachments	IAttachmentFile
	Users	IUser

Example 4-14 shows how to retrieve the referenced IChange object from the Pending Changes table for an item.

Example 4-14: 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
        }
    }
}
```

Example 4-15 shows how the code in Example 4-14 can be simplified by using the ITable.getReferentIterator() method to iterate through the table's referenced objects.

Example 4-15: 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.

Example 4-16 shows how to use the isFlagSet() method to determine whether an item has attachments.

Example 4-16: 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 One, Page Two, and Page Three

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 4-17: 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

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 or Manufacturers tables.
 - 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. SCOs and MCOs do not affect an item's revision.
- 5 Modify a redline table, such as the Redline BOM, Redline Manufacturers (AML), or Redline Price Lines.

The following example shows the steps necessary for redlining the Manufacturers table (AML) of an item.

Example 4-18: 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 Preferred 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);
}
```

Removing Redlines

When you make redline changes to a table such as a BOM, 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 were 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 4-19: 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();
    }
}
```

CHAPTER 5

Working with Data Cells

This chapter provides code examples that show how to retrieve and modify Agile PLM data cells. It contains the following topics:

- ❑ *About Data Cells*
- ❑ *Data Types*
- ❑ *Checking Discovery Privilege*
- ❑ *Checking Whether a Cell is Read-Only*
- ❑ *Getting Values*
- ❑ *Setting Values*
- ❑ *Getting and Setting List Values*

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 the 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 5-1 lists the object types of cell values for `getValue()` and `setValue()` methods.

Table 5-1: Data types for cell values

DataTypeConstants	Object type associated with <code>getValue</code> and <code>setValue</code>
TYPE_DATE	Date
TYPE_DOUBLE	Double
TYPE_INTEGER	Integer
TYPE_MONEY	Money
TYPE_MULTILIST	IAgileList
TYPE_OBJECT	Object
TYPE_SINGLERLIST	IAgileList
TYPE_STRING	String
TYPE_TABLE	Table

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

Checking Discovery Privilege

The Discovery privilege is the most basic Agile PLM privilege. It allows users to learn 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 Example 5-1.

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 is different behavior from other Agile PLM clients. For example, the Agile Web Client displays the value "No Privilege" for any field you don't have privileges to see.

Example 5-1: 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 Whether a Cell is Read-Only

Roles and privileges, which are assigned to a user by person who administers the Agile PLM system, determine the access that a 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 should 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 5-2: 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.

Table 5-2: Methods for setting cell values

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

Before you can begin working with a cell's value, you must select the cell. Agile PLM cells are simply instances of attributes. To specify the attribute for a cell, you specify either the attribute's ID constant, its fully qualified name (such as "Title Block.Description"), or an `IAttribute` object. For more information about referencing attributes, see "Referencing Attributes" on page 18-10.

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

Example 5-3: 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 qualified attribute name.

Example 5-4: 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 5-5: 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 5-6: 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 5-1.

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 3-17.

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.

Setting Values

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

Table 5-3: Methods for setting cell values

Method	Description
<code>ICell.setValue()</code>	Sets a cell value.
<code>IRow.setValue()</code>	Sets a cell value contained within a row.
<code>IRow.setValues()</code>	Sets multiple cell values contained within a row.
<code>IDataObject.setValue()</code>	Sets a cell value on Page One, Page Two, or Page Three.
<code>IDataObject.setValues()</code>	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 5-7: 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 5-8: 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 5-9: 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 Chapter 19, "Handling Exceptions."

Getting and Setting List Values

There are two different datatypes for lists, `SingleList` and `MultiList`. 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 8-14.

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 5-10: 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 5-11: 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 Chapter 8, “Working with Lists.”

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 5-12: 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 Example 5-12 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 5-13: 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");
    values.setSelection(new Object[] { continent });

    // Set the country to "United States"
    IAgileList country = (IAgileList)continent.getChildNode("United States");
    continent.setSelection(new Object[] { country });

    // Set the city to "San Jose"
    IAgileList city = (IAgileList)country.getChildNode("San Jose");
    country.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.

Table 5-4: Methods for retrieving reference designators

Method	Description
<code>IReferenceDesignatorCell.getCollapsedValue()</code>	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.
<code>IReferenceDesignatorCell.getExpandedValue()</code>	Gets an expanded value of a reference designator. For example, for “A1-A3” the string, “A1, A2, A3” would be returned.
<code>IReferenceDesignatorCell.getReferenceDesignators()</code>	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 `IReferenceDesignatorCell`; 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 shows how to create and work with folders. It contains the following topics:

- ❑ *About Folders*
 - ❑ *Loading a Folder*
 - ❑ *Creating a Folder*
 - ❑ *Setting the Folder Type*
 - ❑ *Adding and Removing Folder Elements*
 - ❑ *Getting Folder Elements*
 - ❑ *Deleting a Folder*
-

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 and therefore has its own interface, `IFileFolder`. A file folder holds one or more files that can be referenced from the Attachments table of other objects. For more information about file folders, see Chapter 10, “Working with Attachments and File Folders.”

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. You cannot modify or delete the Recently Visited folder.

Note The user’s 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. Every user, however, has a Home folder. From each user's Home folder, you can construct the 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 don't set a transaction boundary for a folder, it's automatically updated as soon as you add anything to it or remove anything from it.

`IFolder` extends `java.util.Collection` and `ITreeNode` and supports all the methods provided by those superinterfaces. This means that 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.

Table 6-1: `IFolder` methods inherited from `java.util.Collection` and `ITreeNode`

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

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 could specify “/Personal Searches” or “/PERSONAL SEARCHES”.

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

Example 6-1: 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 6-2: 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 6-3: 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 6-4: 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.

Example 6-5: Adding objects to a folder

```
public void addFolderItem(IFolder folder, Object obj) {

    try {
        folder.addChild(obj);
    } catch (APIException ex) {
        System.out.println(ex);
    }
}
```

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

Example 6-6: 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 6-7: 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 6-8: 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 forwards and backwards, use the `IFolder.getFolderIterator()` method to return an `ITwoWayIterator` object. `ITwoWayIterator` provides `previous()`, `next()`, and `skip()` methods, among others.

Example 6-9: 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 6-10: Deleting a folder

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

Working with Items, BOMs, and AMLs

This chapter shows how to work with items, BOMs, and AMLs. It contains the following topics:

- ❑ *About Items*
- ❑ *Getting and Setting the Revision of an Item*
- ❑ *Changing the Incorporated Status of a Revision*
- ❑ *Working with BOMs*
- ❑ *Working with AMLs*

About 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 7-1: 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.getValue()).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 `IRevised` interface, as shown in the following example, which loads an item and then iterates through the item's revisions.

Example 7-2: 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 `IRevised.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 in 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 7-3: 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 `IRevised.getRevision()` method returns a null String and the `IRevised.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 the 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 7-4: 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.

Table 7-1: BOM tables

BOM Table	ID Constant
Current BOM table	<code>ItemConstants.TABLE_BOM</code>
Redline BOM table	<code>ItemConstants.TABLE_REDLINEBOM</code>

For an example showing how to retrieve a BOM table, see “Retrieving a Table” on page 4-2.

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 7-5: 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(((I AgileList)m_session.getCurrentUser().getValue(  
        UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0]  
    );  
    item.getTable(ItemConstants.TABLE_BOM).createRow(map);  
}
```

For more information about manufacturing sites, see Chapter 9, “Managing Manufacturing Sites.”

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 7-6: 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 manufacturing sites associated with it as the source item.

Example 7-7: 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 7-8: 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());
}
}
```

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 Redlines” on page 4-24.

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 7-9: 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 7-10 shows how to create an ECO and select a workflow for it.

Example 7-10: 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 7-11: 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 7-12: 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.

Table 7-2: Manufacturers tables

BOM Table	ID Constant
Current Manufacturers table	<code>ItemConstants.TABLE_MANUFACTURERS</code>
Redline Manufacturers table	<code>ItemConstants.TABLE_REDLINEMANUFACTURERS</code>

Adding an Approved Manufacturer to the Manufacturers Table

Like 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 7-13: 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_NUMBER, 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_NUMBER, 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().getValue(
        UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0]
    );
    item.getTable(ItemConstants.TABLE_MANUFACTURERS).createRow(mfrPart);
}

```

For more information about manufacturing sites, see Chapter 9, “Managing Manufacturing Sites.”

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 Redlines” on page 4-24.

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

Working with Lists

This chapter describes how to work with different types of Agile lists and how specifically to manage `IAgileList` objects. It contains the following topics:

- ❑ *About Lists*
 - ❑ *Selecting a List Value*
 - ❑ *Selecting a List from the List Library*
 - ❑ *Creating Custom Lists*
 - ❑ *Checking the Data Type of a List*
 - ❑ *Modifying a List*
 - ❑ *Printing the Contents of an `IAgileList` Object*
-

About Lists

Many attributes in the Agile PLM system are configured to be 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. Once you add a value to a list you cannot remove it. However, you can make a list value obsolete so that it can no longer be selected.

List Library

In the 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 the 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.

Figure 8-1: List Library

ID	Name	Description	Enabled	Editable	Is Cascading?
18414	Action Status	Action Status	Yes	Yes	No
2249	AML Preferred Status	AML Preferred Status	Yes	Yes	No
4682	AttachType List	AttachType List	Yes	Yes	No
6820	Audit Result	Audit Result	Yes	Yes	No
8934	Buyer	Buyer	Yes	Yes	No
2000000192	Category 10 List	Category 10 List	Yes	Yes	No
2000000189	Category 7 List	Category 7 List	Yes	Yes	No
2000000190	Category 8 List	Category 8 List	Yes	Yes	No
2000000191	Category 9 List	Category 9 List	Yes	Yes	No
730	Change Analysts	Change Analysts	Yes	No	No
411	Change Category	Change Category	Yes	Yes	No
331	Change Function	Change Function	Yes	Yes	No
8223	Changes	All Change Objects	Yes	No	No
12941	Commodities	Commodities for Items	Yes	No	No
750	Comp Engineers	Component Engineers	Yes	No	No
2000004949	Compliance Manager	Compliance Manager	Yes	No	No
4598	Continent	Continent	Yes	Yes	No
365	Country	Country	Yes	Yes	No
8704	Currencies	Currencies	Yes	No	No
2000000107	Customer List	Customer List	Yes	Yes	No
6505	Customers	All Customer Objects	Yes	No	No
2000002155	Declarations	All Declaration Objects	Yes	No	No
18412	Discussion Priority List	Discussion Priority List	Yes	Yes	No

Note The Agile API provides support for several internal Agile lists that are not exposed in the list library in the Agile Java Client.

SingleList Lists

A SingleList attribute or cell presents a list from which only one value can be selected. The following figure shows the Time Format field, a SingleList cell in the Agile Web Client.

Figure 8-2: SingleList cell in the Agile Web Client

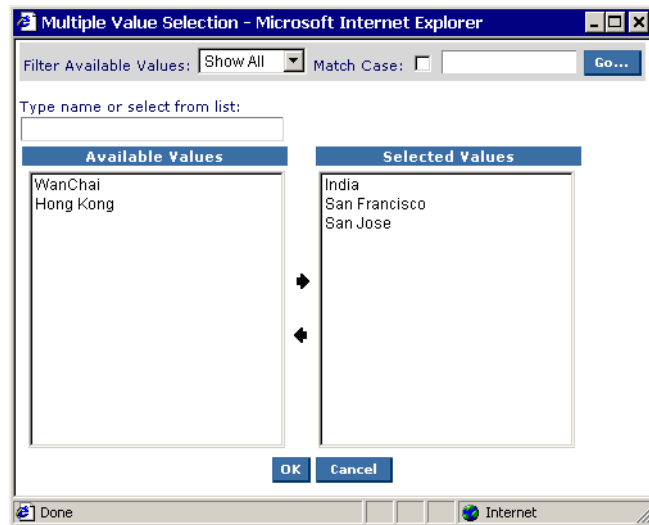
Time Format:

- hh:mm:ss aaa z (10:00:00 AM PST)
- HH:mm:ss z (17:30:00 PST)
- hh:mm:ss aaa z (10:00:00 AM PST)
- HH:mm:ss z (17:30:00 PST)

MultiList Lists

A MultiList attribute or cell presents a list from which multiple values can be selected. In the Agile Web Client, you can select values for a MultiList cell using the Multiple Value Selection window, shown in the following figure.

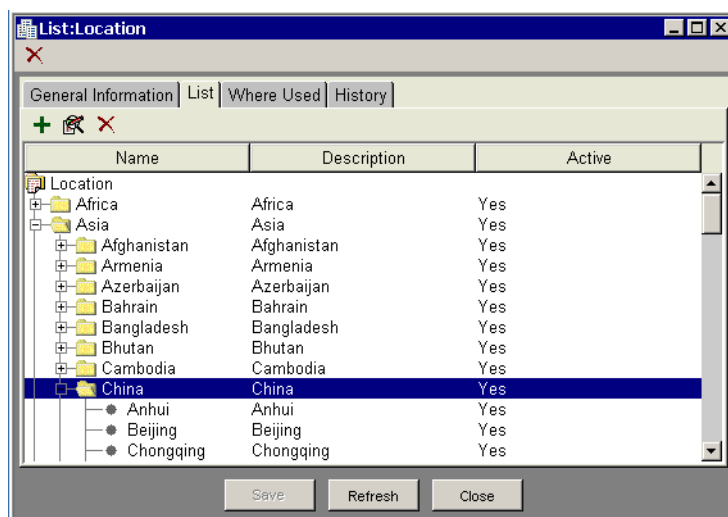
Figure 8-3: Multiple Value Selection window in the Agile Web Client



Cascading Lists

A SingleList attribute can be configured in the Agile Java Client 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 the Agile Java Client. The list has separate levels for continent, country, and city.

Figure 8-4: 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.

Methods that Use IAgileList

The `IAgileList` interface provides methods for getting and setting 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 you select a 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).

Example 8-1: Changing the `Visible` property of an attribute

```
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 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 8-2: 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 8-3: 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 Dynamic Lists

The Agile server has both static lists and dynamic lists. Static lists contain a selection of values that are unchanging at run time. Dynamic lists contain a selection of values that are updated at run time. Users with administrator privileges can modify static lists; new values can be added and current values can be made 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. The 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't enumerate the values for them.

Agile SDK object lists can be 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 the list refers to. The enumeration property for an object is hardcoded at 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 doesn't prevent you from selecting a value for the list.

Example 8-4: Checking whether values for a dynamic list can be enumerated

```
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);
        }
    }
}
}

```

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 8-5: 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);
}

```

Selecting a List from the List Library

The `IListLibrary` interface lets you work 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 ID or by one of the `AdminListConstants`, such as `LIST_DISPOSITION_SELECTION`. Before you attempt to use a list from the list library, make sure the list is enabled.

Cascading lists can only be used for SingleList attributes, not for MultiList attributes. When you select a list from the list library, use `IAdminList.isCascaded()` to check whether the list is a cascading list.

The following example shows how to configure a Page Two list attribute to use a list called Users.

Example 8-6: 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 8-7: 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 the 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()` should 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 8-8: 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 8-9: 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");
```

Creating a New List Automatically by Modifying an Existing List

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 8-10: 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 examples 8-11 and 8-12 below.

Example 8-11: 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();

    // Set level names starting with index 0 for level 1.
    list.setLevelName(0, "Field Office Country");
    list.setLevelName(1, "Field Office City");

    // 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);

} 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 the 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 Example 8-11; it shows how to add the same list values as in Example 8-11, but using fewer lines of code.

Example 8-12: Adding parent nodes and subnodes to a cascading list

```
// Get the list values
IAgileList list = listFO.getValues(); // The list is empty at this point.

// Set level names
list.setLevelName(0, "Field Office Country");
list.setLevelName(1, "Field Office City");

// 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);
```

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 8-13: 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 the list
- ☐ Make list values obsolete
- ☐ Set the list name and description
- ☐ Set level names for a cascading list
- ☐ Enable or disable the list
- ☐ Delete the 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 8-14: 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

Once a value is added to a list, it cannot be removed. If you try to remove a list value using `ITreeNode.removeChild()`, the API throws an exception. Although you can't remove a list value, you can make it obsolete, in which case the value can no longer be selected from the list.

When you mark a list value as obsolete, the server maintains obsolete list values to handle existing objects that may use the values. Consequently, if you use the `IProperty.getAvailableValues()` method, the returned `IAgileList` object may include obsolete list values.

The following example shows how to check whether a list value is obsolete and how to set it as obsolete.

Example 8-15: 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 its name and description. The following example shows how to set the name and description of a list.

Example 8-16: 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 8-11.

Note Level names for cascading lists are not displayed in the 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 8-17: 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 8-18: 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.");
    }
}
```

Printing the Contents of an IAgileList Object

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 8-19: 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 9

Managing Manufacturing Sites

This chapter shows how to manage different manufacturing sites used in your Agile PLM system. It contains the following topics:

- ❑ *About Manufacturing Sites*
 - ❑ *Controlling Access to Sites*
 - ❑ *Creating a Manufacturing Site*
 - ❑ *Loading a Manufacturing Site*
 - ❑ *Retrieving the Sites Table for an Item*
 - ❑ *Adding a Manufacturing Site to the Sites Table*
 - ❑ *Selecting the Current Manufacturing Site for an Item*
 - ❑ *Disabling a Site*
-

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, see the *Product Collaboration Guide*.

Controlling Access to Sites

The use of sites is controlled by your organization's licenses, plus users' licenses, roles, privileges, and the default site property. You can create an unlimited number of manufacturing sites, but your organization's license determines how many of those sites can be *enabled*. Your organization may have implemented the Agile PLM system in such a way that users can access only the information pertaining to certain sites.

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.

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 9-1: 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 9-2: Loading a manufacturing site

```
try {
    // Load the Hong Kong site
    IManufacturingSite siteHK =
        (IManufacturingSite)m_session.getObject(ManufacturingSiteConstants.CLASS_SITE, "Hong
Kong");
    // Load the Taipei site
    IManufacturingSite siteTaipei =
        (IManufacturingSite)m_session.getObject(IManufacturingSite.OBJECT_TYPE, "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 9-3: 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 9-3 for an example of how to use `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 9-4: 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 9-5: 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 9-6: 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 9-7: 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().getValue(
        UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0]
    );
    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 9-8: 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 10

Working with Attachments and File Folders

This chapter shows how to work with attachments in an Agile API program. It contains the following topics:

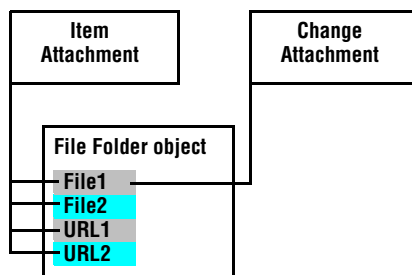
- ❑ *About Attachments*
- ❑ *Working with File Folders*
- ❑ *Working with the Attachments Table*
- ❑ *Checking Out a File Folder*
- ❑ *Canceling Checkout of a File Folder*
- ❑ *Adding Files and URLs to the Attachments Table*
- ❑ *Checking In a File Folder*
- ❑ *Replacing a File*
- ❑ *Retrieving Attachment Files*
- ❑ *Deleting File Folders and Attachments*

About Attachments

Most primary Agile API objects, such as `IItem`, `IChange`, `IManufacturerPart`, `IManufacturer`, `IPackage`, `ITransferOrder`, `IUser`, and `IUserGroup`, have an Attachments table that lists indirect references to the files or URLs contained in separate file folders. Each row in an Attachments table can refer to one file or to all files from a referenced file folder. A file folder is a business object that specifies one or more URLs (Internet addresses) or files stored in the file server vault.

The following figure illustrates an example of how files or URLs contained in a file folder can be referenced indirectly from the Attachments table of multiple business objects, in this case an item and a change.

Figure 10-1: 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 it.

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 the 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, see the *Agile PLM Administrator Guide*.

Working with File Folders

A file folder is an Agile PLM business object with 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. Currently there are two out-of-the-box file folder classes, File Folders and Historical Report File Folders, and each of these classes has subclasses. The Agile PLM administrator can define new file folder subclasses.

`IFileFolder` is the interface that corresponds to the file folder business object. The following example shows how to create a file folder.

Example 10-1: 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);
}
```

When you add a file or an 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 10-2: 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;
}
```

File Folder Tables

File folders have the following tables:

Table 10-1: `IFileFolder` tables

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
Workflow	TABLE_WORKFLOW	Read/Write
Relationships Affects	TABLE_RELATIONSHIPSAFFECTS	Read-only
Relationships Affected By	TABLE_RELATIONSHIPSAFFECTEDBY	Read-only

Table 10-1: IFileFolder tables (continued)

Table	Constant	Read/Write Mode
Relationships References	TABLE_RELATIONSHIPSREFERENCES	Read/Write
Where Used	TABLE_HISTORY	Read-only
History	TABLE_ATTACHMENTS	Read-only

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 10-3: 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();
}
```

Using the IAttachmentFile Interface

IAttachmentFile is an interface that provides generalized access to a file stored in the Agile PLM file vault. It 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.

Table 10-2: Return values for `IAttachmentFile.getFile()` and `IAttachmentFile.isSecure()`

Calling object	<code>getFile()</code> return value	<code>isSecure()</code> return value
Row from the Attachments table of any business object	Returns either a single file <code>InputStream</code> if the row refers to a specific file from the file folder or a zipped <code>InputStream</code> with all the files from the file folder.	<code>true</code> if the referenced file is not URL, or all the files are not URLs.
<code>FileFolder</code> object	Returns a zipped <code>InputStream</code> with all files from the file folder.	<code>true</code> 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 <code>InputStream</code> that refers to a specific file from the file folder.	<code>true</code> if the referenced file is not an 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 10-4: 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 the Attachments Table

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.

Example 10-5 shows how to retrieve an item, get the Attachments table for the item, and then select the first attachment.

Example 10-5: 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
    InputStream stream = ((IAttachmentFile)row).getFile();
}
else {
    JOptionPane.showMessageDialog(null, "There are no files listed.",
        "Error", JOptionPane.ERROR_MESSAGE);
}
} catch (APIException ex) {
    System.out.println(ex);
}

```

Using the ICheckoutable Interface

ICheckoutable is an interface that lets you check in and check out files associated with an object. It 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()

Example 10-6 shows how to use the ICheckoutable interface to check out and check in a file from a row of the Attachments table.

Example 10-6: 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 7-1.

Checking Whether the Revision is Incorporated

Generally, if the revision for an item has been released, it has also been incorporated. The attachments for an incorporated item are locked and cannot be checked out. You can still view the attachments, but you cannot modify them unless you submit a new change (and thereby create a new revision). For more information about checking whether a revision is incorporated, see “Changing the Incorporated Status of a Revision” on page 7-3.

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 10-7: 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 “Using the ICheckoutable Interface” on page 10-5.

Canceling Checkout of a File Folder

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.

Example 10-8 shows how to cancel a checkout of a file folder.

Example 10-8: 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 “Using the ICheckoutable Interface” on page 10-5.

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 can be either one or more physical files or an Internet address, or 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 an 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 an URL attachment, the server stores a reference to the Internet location but does not upload a file. Therefore, you cannot download an URL attachment. The Agile API validates URL strings that you attempt to check in as an attachment. If an URL is invalid, the Agile API considers the string a filename instead of an 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.

Example 10-9 shows several `addAttachment()` methods that illustrate the different ways to add rows to an Attachments table.

Example 10-9: 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 the copying of 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 10-10: 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 10-11: Deep cloning a 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 item1
    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 attachments table of item2
    IRow row2 = tblFiles2.createRow(params);

    // Check in folder1 and folder2
    folder1.checkIn();
    folder2.checkIn();
}
```

Specifying the File Folder Subclass When Adding Attachments

Your Agile PLM system can be set up 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.

To make it easier to specify which file folder subclass to use, a virtual attribute, `CommonConstants.ATT_ATTACHMENTS_FOLDERCLASS`, is provided. You can 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 10-12: Specifying the file folder subclass when adding attachments

```
IAgileClass class = 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, class);

// add file
IRow row = tab_attachment.createRow(map);
```

Setting the Version of File Folder Files

A file folder can have several versions. When you add a file folder to the Attachments table of another business object, you can specify the file version to use. If you don't specify a file version, the Agile API uses the default or latest version. If you specify a file version, the row on the Attachments table is linked to that version.

If the parent object containing the Attachments table is an item, you can incorporate the item to lock the specified versions of its attachments. For more information about incorporating an item, see “Changing the Incorporated Status of a Revision” on page 7-3.

Example 10-13: Setting the version when adding a row to the Attachments table

```
// Add a file folder to the Attachments table and use version 1 of all files
public static IRow addAttachment(ITable attTable, IFileFolder ff) throws APIException {
    ff.setCurrentVersion(new Integer(1));
    IRow row = attTable.createRow(ff);
    return row;
}

// Add a file folder to the Attachments table and use version 1 of all files.
// This method passes a hash table for the params parameter of createRow().
public static IRow addAttachment(ITable attTable, IFileFolder ff) throws APIException {
    HashMap map = new HashMap();
    map.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, ff);
    map.put(CommonConstants.ATT_ATTACHMENTS_FOLDER_VERSION, new Integer(1));
    IRow row = attTable.createRow(map);
    return row;
}

// Add a row from the Files table of a file folder to the Attachments table and use version 2 of
the file
public static IRow addAttachment(ITable attTable, IFileFolder ff) throws APIException {
    ff.setCurrentVersion(new Integer(2));
    IRow filesRow = (IRow)ff.getTable(FileFolderConstants.TABLE_FILES).iterator().next();
    IRow row = attTable.createRow(filesRow);
    return row;
}
```

Checking In a File Folder

After you finish editing a file folder that you checked out, you can check it in again. Once it is checked in, other users can check it out. You can check in a file folder from any computer, not just the computer used to check it out.

File folders can contain multiple files. When you check in a file folder, you automatically check in all files that are contained in it. You do not need to specifically list the files contained in the file folder.

Example 10-14: Checking in a file folder

```
void checkInFiles(IFileFolder ff) throws Exception {

    // Set the local file path
    String path = "d:\\files\\file1.doc";

    // Get the Files table
    ITable files = ff.getTable(FileFolderConstants.TABLE_FILES);
```

```
// Get the first row
IRow row = (IRow)files.iterator().next();

// Replace the file
row.setValue(FileFolderConstants.ATT_FILES_FILE_NAME, new File(path));

ff.checkIn();
}
```

Note You can also use `ICheckoutable.checkIn()` to check in a row of the Attachments table. See “Using the `ICheckoutable` Interface” on page 10-5.

Replacing a File

To replace a file listed on the Files table of a file folder, use the `IRow.setValue()` method. For the `cellID` parameter of `IRow.setValue()`, specify the attribute ID constant for the File Name field. For the `value` parameter of `IRow.setValue()`, specify either a `String` (the local path of the file), a `File` object, or an `InputStream` object.

Example 10-15: Replacing a file

```
// Replacing a file by specifying the path to a file
String path = "d:\\files\\file1.doc";
row.setValue(FileFolderConstants.ATT_FILES_FILE_NAME, path);

// Replacing a file by specifying a File object
String path = "d:\\files\\file1.doc";
row.setValue(FileFolderConstants.ATT_FILES_FILE_NAME, new File(path));
```

If you use `IRow.setValues()` instead of `IRow.setValue()`, you can pass a `Map` object to change the content and file name at the same time. The `Map` should contain two key parameters, `FileFolderConstants.ATT_FILES_FILE_NAME` and `FileFolderConstants.ATT_FILES_CONTENT`. Possible values for these keys are a `String`, a `File`, or an `InputStream`.

Example 10-16: Updating the file content and the filename at the same time

```
void changeContentAndFilename(IFileFolder ff, String newFilename, File newFile) throws Exception {

    // Check out the file folder
    ff.checkOutEx();

    // Get the Files table
    ITable files = ff.getTable(FileFolderConstants.TABLE_FILES);

    // Get the first row
    IRow row = (IRow)files.iterator().next();

    // Create a Map containing the new file and filename
    Map map = new HashMap();
    map.put(FileFolderConstants.ATT_FILES_CONTENT, newFile);
    map.put(FileFolderConstants.ATT_FILES_FILE_NAME, newFilename);

    // Set values for content and file name
    row.setValues(map);

    // Check in
    ff.checkIn();
}
```

If the filename hasn't changed, you can update only the file content, as shown in the following example.

Example 10-17: Updating only the file content

```
...
// Create a Map containing the new file
Map map = new HashMap();
map.put(FileFolderConstants.ATT_FILES_CONTENT, newFile);

// Set values
row.setValues(map);
...
```

To replace a file listed on the Attachments table of a business object, you can also use the `IFileFolder.setFile()` method. For the `param` parameter of `setFile()`, specify a `File`, `InputStream`, or `Map` object. If you specify a `Map` object, it must contain two key parameters represented by the following Agile API constants:

- ❑ `CommonConstants.ATT_ATTACHMENTS_FILE_NAME`
- ❑ `CommonConstants.ATT_ATTACHMENTS_CONTENT`

Possible values for the file content attribute are a `String`, a `File`, or an `InputStream`.

Example 10-18: Replacing a file for a row of the Attachments table

```
public replaceFileInRow(IRow row, String filename, String filePath) throws Exception {
    ((ICheckoutable)row).checkoutEx();
    Map map = new HashMap();
    map.put(CommonConstants.ATT_ATTACHMENTS_FILE_NAME, filename);
    map.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, new File(filePath));
    ((IAttachmentFile)row).setFile(map);
    ((ICheckoutable)row).checkIn();
}
```

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 an 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.

Example 10-19 shows how to retrieve a copy of an attached file.

Example 10-19: 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 10-20: 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 File Folders and Attachments

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 2-19.

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 4-17. 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.

CHAPTER 11

Managing Workflow

This chapter shows how to manage Agile workflow processes. It contains the following topics:

- ❑ *About Workflow*
- ❑ *Selecting a Workflow*
- ❑ *Adding and Removing Approvers*
- ❑ *Approving or Rejecting a Change*
- ❑ *Commenting a Change*
- ❑ *Auditing a Change*
- ❑ *Changing the Workflow Status of an Object*
- ❑ *Sending an Agile Object to Selected Users*
- ❑ *Sending an Agile Object to User Groups*

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. Table 11-1 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.

Table 11-1: Sequences in the change control process

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
Default CTOs	Pending → Review → Released → Complete

Table 11-1: Sequences in the change control process (continued)

Workflow	Default sequence
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 11-1: Getting the status of a change object

```
private static boolean isReleased(IChange change) throws APIException {
    return (change.getStatus().getStatusType().equals(StatusConstants.TYPE_RELEASED));
}
```

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 11-2 lists privileges needed to perform workflow actions.

Table 11-2: Privileges related to workflow actions

Privilege	Related API
Change Status	<code>IRoutable.changeStatus()</code>
Comment	<code>IRoutable.comment()</code>
Send	<code>DataObject.send()</code>

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 11-2: 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.setWorkflow()` 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 11-3: 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 11-4: Making a change unassigned

```

private void unassign(IChange change) throws APIException {
    change.setWorkflow(null);
}

```

Adding and Removing Approvers

After a change has been routed and the online approval process has begun, it may be necessary to add or remove people from the list of approvers or observers. To add or remove approvers or observers, a user must have both the Agile Product Change Server license and 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; see the API Reference for more details.

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 11-5: 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 11-6: 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);
    }
}
```

Approving or Rejecting a Change

After a change is routed to a group approvers, the online approval process begins. Users listed on 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 user license.

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; see the API Reference for more details.

The following example shows how to approve a change.

Example 11-7: 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 user's 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; see the API Reference for more details.

The following example shows how to reject a change.

Example 11-8: 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; see the API Reference for more details.

The following example shows how to comment a change.

Example 11-9: 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 11-10: 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. Generally, you release a change after it has been signed off by the CCB members. In addition to changing 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; see the API Reference 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 11-11: 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 11-12: 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 11-13: 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 11-14: 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 = (IUser[]) (userList.toArray());

    // 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 11-15: 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 12

Product Cost Management

This chapter shows how to manage product sourcing processes using the Agile API. It contains the following topics:

- ❑ *About Product Sourcing*
 - ❑ *Managing Pricing*
 - ❑ *Working with Suppliers*
 - ❑ *Working with Sourcing Projects*
-

About Product Sourcing

Agile PLM's Product Sourcing features support, enhance, and simplify the handling of total product costs throughout the product lifecycle. This lets you effectively manage and manipulate sourcing content, collaborate with suppliers to establish new sourcing content, and analyze the data. With Product Sourcing, you can

- ❑ Create a sourcing project
- ❑ Gather and prepare product content
- ❑ Leverage pricing contracts and history
- ❑ Create RFQs
- ❑ Manage supplier RFQ responses and negotiate pricing
- ❑ Conduct project analysis

The Agile API supports the following Product Sourcing objects:

- ❑ **IPrice** — Interface for the `Price` class, which handles both published prices and historical prices.
- ❑ **IChange** — Interface for the `Change` class, which includes Price Change Orders (PCOs).
- ❑ **IProject** — Interface for the `Project` class, which is the container used for product sourcing data.
- ❑ **IRequestForQuote** — Interface for the `RequestForQuote` class, which represents an RFQ for a sourcing project.
- ❑ **ISupplier** — Interface for the `Supplier` class.
- ❑ **ISupplierResponse** — Interface for the `SupplierResponse` class, which handles data sent by a supplier in response to an RFQ.

The Agile API allows you to read and modify most Product Sourcing objects, but not create them. However, you can create `IPrice` and `IChange` objects. The following table shows the create, read, and modify rights for all Product Sourcing objects.

Table 12-1: Create, Read, and Modify rights for Product Sourcing objects

Object	Create	Read	Modify
<code>IChange</code> (including <code>PCO</code>)	✓	✓	✓
<code>IPrice</code>	✓	✓	✓
<code>IProject</code>		✓	✓
<code>IRequestForQuote</code>		✓	✓
<code>ISupplier</code>	✓	✓	✓
<code>ISupplierResponse</code>		✓	✓

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 Quote and Published Price. A Historical Quote object contains price quotes from previous projects or legacy data. A Published Price contains published prices or contract prices on current items and manufacturer parts.

Here 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 aren't 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.

Example 12-1: 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, "B");
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_PROGRAM, "PROGRAM0023");
    params.put(PriceConstants.ATT_GENERAL_INFORMATION_MANUFACTURING_SITE, "Global");
    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 12-2: 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, see Table 4-2 on page 4-3.

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 12-3: 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 be modified 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 create a PCO using the `IAgileSession.createObject()` method.

Example 12-4: 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 12-5: Loading a supplier

```
public ISupplier getSupplier() throws APIException {

    ISupplier supplier = (ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE, "SUP20013");

    return supplier;
}
```

For a list of Supplier object tables, see Table 4-2 on page 4-3. 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 12-6: 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://www.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 project is where you prepare content for sourcing tasks, such as Requests for Quotes (RFQs) and sourcing analysis. The project is a centralized, collaborative solution. Multiple users can add data to a project and perform analysis of sourcing results. Because sourcing projects serve as the home for all sourcing activities, they are tightly linked with many classes of objects, including Supplier, RequestForQuote (RFQ), and SupplierResponse.

Although the Agile Web Client provides rich functionality for sourcing projects, the Agile API exposes projects for simple data extraction and updating. Therefore, you cannot use the Agile API to

- ❑ add items to a project
- ❑ validate items, commodities, or manufacturer parts
- ❑ publish items, AMLs, and manufacturer parts
- ❑ filter project tables

- ❑ modify the price scenario for a project (change quantity breaks and effectivity periods)
- ❑ create an RFQ
- ❑ create an RFQ response

Loading a Project

To load a sourcing project, use the `IAgileSession.getObject()` method. To uniquely identify a sourcing project, specify the value for the Cover Page | Number attribute.

Example 12-7: Loading a project

```
public IProject getProject() throws APIException {

    String prjnum = "PRJACME_110";
    IProject prj = (IProject)m_session.getObject(IProject.OBJECT_TYPE, prjnum);

    return prj;
}
```

For a list of sourcing project tables, see Table 4-2 on page 4-3. The Agile API does not support adding new rows to Project tables.

Setting Cover Page Values for a Project

You can read and update all read/write Project cells. The following example shows how to update cells on the Project's Cover Page.

Example 12-8: Setting values for a 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 project for Odyssey III");

    //Update Manufacturing Site (List field)
    cell = project.getCell(ProjectConstants.ATT_GENERAL_INFORMATION_MANUFACTURING_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_LOCATION);
    listvalues = cell.getAvailableValues();
    listvalues.setSelection(new Object[] { "San Jose" });
    cell.setValue(listvalues);
}
```

Working with RFQs

Requests for Quotes (RFQs) allow users to request pricing information from suppliers. The RFQ serves as the instrument for negotiating pricing and terms for items or manufacturer parts. Many RFQs can originate from one single project.

RFQs support one-to-many correspondence with suppliers. Therefore, one RFQ may generate many responses from suppliers.

Note The Agile API does not support creating new RFQs. However, you can load an existing RFQ and modify it.

Loading an RFQ

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 12-9: Loading an RFQ

```
public IRequestForQuote getRFQ() throws APIException {  
  
    IRequestForQuote rfq = (IRequestForQuote)m_session.getObject(IRequestForQuote.OBJECT_TYPE,  
        "RFQ01004");  
    return rfq;  
  
}
```

In addition to loading an RFQ using `IAgileSession.getObject()`, you can also select an RFQ from the RFQ table of a Project object.

Example 12-10: Loading an RFQ from the Project RFQs table

```
public IRequestForQuote getFirstRFQFromTable(IProject project) throws APIException {  
  
    //Load the RFQ table  
    ITable rfqs = project.getTable(ProjectConstants.TABLE_RFQS);  
  
    //Get the first row  
    ITwoWayIterator i = rfqs.getTableIterator();  
    IRow row = (IRow)i.next();  
  
    //Get the referenced RFQ  
    IRequestForQuote rfq = (IRequestForQuote)row.getReferent();  
  
    return rfq;  
}
```

For a list of RFQ tables, see Table 4-2 on page 4-3. The Agile API does not support adding new rows to RFQ tables.

Working with RFQ Responses

Suppliers respond to RFQs by sending in price quotes using a response form. The response can be entered directly in the Agile PLM system, or the supplier can choose to edit the comma-delimited file in a spreadsheet program like Excel. The Agile API exposes RFQ responses in an interface named `ISupplierResponse`.

Note The Agile API does not support creating new RFQ responses. However, you can load an existing RFQ response and modify it.

Loading an RFQ Response

To load an RFQ response, use the `IAgileSession.getObject()` method. To uniquely identify an RFQ response, specify values for the Cover Page | RFQ Number and Cover Page | Supplier attributes. If you fail to specify the Supplier, the RFQ response can't be loaded. Each RFQ can be sent to many suppliers.

Example 12-11: Loading an RFQ response

```
public ISupplierResponse getRFQResponse() throws APIException {

    ISupplierResponse resp = null;
    Map params = new HashMap();
    params.put(SupplierResponseConstants.ATT_COVERPAGE_RFQ_NUMBER, "RFQ01004");
    params.put(SupplierResponseConstants.ATT_COVERPAGE_SUPPLIER, "SUP20013");

    resp = (ISupplierResponse)m_session.getObject(ISupplierResponse.OBJECT_TYPE, params);

    return resp;
}
```

In addition to loading an RFQ response using `IAgileSession.getObject()`, you can also select an RFQ Response from the Responses table of an RFQ object.

Example 12-12: Loading an RFQ response from the RFQ Responses table

```
public ISupplierResponse getFirstResponseFromTable(IRequestForQuote rfq) throws APIException {

    //Load the Responses table
    ITable responses = rfq.getTable(RequestForQuoteConstants.TABLE_RESPONSES);

    //Get the first row
    ITwoWayIterator i = responses.getTableIterator();
    IRow row = (IRow)i.next();

    //Get the referenced Supplier Response
    ISupplierResponse resp = (ISupplierResponse)row.getReferent();

    return resp;
}
```

For a list of RFQ Response tables, see Table 4-2 on page 4-3. The Agile API does not support adding new rows to RFQ Response tables.

CHAPTER 13

Managing and Tracking Quality

This chapter shows how to manage product quality processes using the Agile API. It contains the following topics:

- ❑ *About Quality Control*
 - ❑ *Working with Customers*
 - ❑ *Working with Product Service Requests*
 - ❑ *Working with Quality Change Requests*
 - ❑ *Using Workflow Features with PSRs and QCRs*
-

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, see 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.

The `ICustomer` 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 13-1: 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 13-2: 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);
}
```


For a list of Customer tables, see Table 4-2 on page 4-3.

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 13-3: 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 13-4 shows how to create problem reports and NCRs.

Example 13-4: 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;

}
```

For a list of Product Service Request tables, see Table 4-2 on page 4-3.

Assigning a Product Service Request to a Quality Analyst

To assign a problem report or NCR to a Quality Analyst, you 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. Example 13-5 shows how to set the value for the **Cover Page.Quality Analyst** field for a problem report or NCR.

Example 13-5: 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 13-6: 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_CHANGE);
    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_SITE);
    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 13-7: 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 `I AgileSession.createObject()` method. The only required attribute value you must specify is the object's number. Example 13-8 shows how to create both CAPA and Audit QCRs.

Example 13-8: 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;

}
```

For a list of Quality Change Request tables, see Table 4-2 on page 4-3.

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 13-9: 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
I AgileList 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.

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 13-10: Saving a QCR as an ECO

```
public IChange saveQCRasECO(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();

    // Save the QCR as an ECO
    IChange eco = (IChange)qcr.saveAs(ChangeConstants.CLASS_ECO, numbers[0]);

    // 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. Table 13-1 lists the workflow commands you can use for product quality objects.

Table 13-1: Product quality workflow features

Feature	Equivalent API(s)
Audit a PSR or QCR	<code>IRoutable.audit()</code>
Change the status of a PSR or QCR	<code>IRoutable.changeStatus()</code>
Send a PSR or QCR to another user	<code>IDataObject.send()</code>
Approve a PSR or QCR	<code>IRoutable.approve()</code>
Reject a PSR or QCR	<code>IRoutable.reject()</code>

Table 13-1: Product quality workflow features (continued)

Feature	Equivalent API(s)
Comment on a PSR or QCR	<code>IRoutable.comment()</code>
Add or remove approvers for a PSR or QCR	<code>IRoutable.addApprovers()</code> <code>IRoutable.removeApprovers()</code>

For more details on how to use the Agile API to manage workflow, see Chapter 11, “Managing Workflow.”

Selecting a Workflow

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.setWorkflow()` to select a workflow, as shown in Example 13-11.

Example 13-11: 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 13-12: 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 14

Creating and Managing Programs

This chapter shows how to create and manage programs. It contains the following topics:

- ❑ *About Programs*
 - ❑ *Creating a Program*
 - ❑ *Loading a Program*
 - ❑ *Using Program Templates*
 - ❑ *Scheduling a Program*
 - ❑ *Working with Program Baselines*
 - ❑ *Delegating Ownership of a Program to Another User*
 - ❑ *Adding Resources to a Program's Team*
 - ❑ *Substituting Program Resources*
 - ❑ *Locking or Unlocking a Program*
 - ❑ *Working with Discussions*
-

About Programs

You can use the program management features of Agile Program Execution (PE) to define a program and all of the associated elements such as activity schedules, deliverables, and discussions. These capabilities let you to determine availability of required resources, assign resources to tasks, identify bottlenecks, and respond to over- and under-allocated resource conditions. You can also create and reuse program templates.

You use program objects to schedule and execute programs. Each program not only contains schedule information but also the attachments, discussions and actions items, resources and roles, and history of all activity related to the program. 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 programs. The `IProgram` interface represents all program objects, including programs, phases, tasks, and gates.

Like other Agile PLM business objects, the `IProgram` interface implements `IRoutable`, which means it uses the same `IRouteable.changeStatus()` method to change a program's workflow status and to route it to other users. For more information, see "Changing the Workflow Status of an Object" on page 11-7.

Differences in Behavior of Program Objects

The `IProgram` interface implements several interfaces commonly used by other Agile PLM objects. However, it also provides the following distinct functionality that separates it from other objects.

- ❑ The Program object is a container of other underlying program objects, such as Phases, Tasks, and Gates. The underlying program objects are related to the parent object, usually the Program, through the Schedule table.

- ❑ Programs 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.
- ❑ Programs can be archived. If you archive the root program, the entire program tree is soft-deleted from the system.
- ❑ Programs can be locked or unlocked.

Creating a Program

To create a program, use the `IAgileSession.createObject()` method. When you specify program parameters, you must specify the program subclass (for example, program, phase, task, or gate). For programs, phases, and tasks, you must also specify following required program 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 a new program and specify the required attributes.

Example 14-1: Creating a program

```
try {
    // Create a Map object to store parameters
    Map params = new HashMap();

    // Set program name
    String name = "APOLLO PROGRAM";

    // Set program start date
    Date start = new Date();
    start.setTime(1);

    // Set program end date
    Date end = new Date();
    end.setTime(1 + 2*24*60*60*1000);

    // Set program 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" });

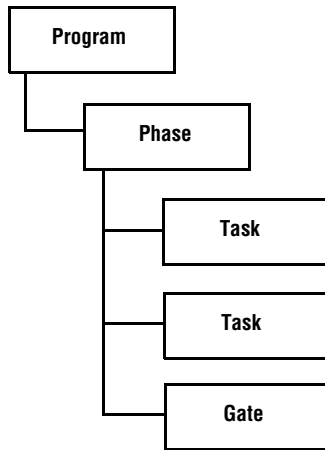
    // Initialize the params object
    params.put(ProgramConstants.ATT_GENERAL_INFO_NAME, name);
    params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE, start);
    params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE, end);
    params.put(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE, avail);

    // Create the program
    IProgram program = (IProgram)m_session.createObject(ProgramConstants.CLASS_PROGRAM, params);

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


A program contains 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 programs. The following figure shows the hierarchy of program objects.

Figure 14-1: Program hierarchy



You can use the `IAgileSession.createObject()` method to create phases, tasks, and gates in the same way that you create other program objects. Once you create these different types of activities, you can add them to the Schedule table of a program. For more information, see “Scheduling a Program” on page 14-5.

Loading a Program

To load a program, use the `IAgileSession.getObject()` method. To uniquely identify a program, specify the value for the **General Info.Number** attribute. You can also load a program 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 14-2: Loading a program

```

public IProgram loadProgram(String number) throws APIException {
    IProgram program = (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, number);
    return program;
}
  
```

For a list of Program tables, see Table 4-2 on page 4-3.

Note The News table for Programs is disabled by default. To enable it, log into the Java Client as an Administrator and make the News tab visible.

Using Program Templates

Program templates make it easy to define a new program, activity, or task. A template is simply a program with the **General Info.Template** attribute set to “Template”. You can use a template to create a new program by loading it and then using the `IProgram.saveAs()` method.

This special version of the `saveAs()` method lets you specify the program tables to copy from the original program to the new program. 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 following example. For a complete list of program tables, see Table 4-2 on page 4-3.

Example 14-3: Creating a new program from a template

```
try {

    // Get the program 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);
}
```

When you create a program, 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 program or when you save it as a new program. Existing programs cannot be changed from the "Active" or "Proposed" state to "Template". The following example shows how to open a program and save it as a template.

Example 14-4: Saving a program 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(
            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[] {"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 a Program

To schedule a program, 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 14-5: 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 program
    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 program's schedule has been defined, you can easily reschedule it using the `IProgram.reschedule()` method. The `reschedule()` method takes a couple of parameters, the `IProgram.RESCEDULE` constant and the new value for that schedule option. Here are the list of `IProgram.RESCEDULE` constants you can use:

- ❑ `STARTDATE` — Moves the schedule end date to the specified date.
- ❑ `ENDDATE` — Moves the schedule end date to the specified date.
- ❑ `BACKWARD_DAYS` — Moves the schedule backward by the specified number of days.
- ❑ `FORWARD_DAYS` — Moves the schedule forward by the specified number of days.

Example 14-6: Rescheduling a program

```
try {

    // Get a program
    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.RESCEDULE.STARTDATE, start);

        // Change the schedule end date
        program.reschedule(IProgram.RESCEDULE.ENDDATE, end);

        // Move the schedule backward three days
        program.reschedule(IProgram.RESCEDULE.BACKWARD_DAYS, new Integer(3));

        // Move the schedule forward two days
        program.reschedule(IProgram.RESCEDULE.FORWARD_DAYS, new Integer(2));
    }
} catch (Exception ex) {
    System.out.println(ex);
}
```

Working with Program Baselines

Program baselines allow you to compare actual progress with your original plans. When you create a baseline, a snapshot of your program's 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 program 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 14-7: Creating and retrieving baselines

```
try {  
  
    // Get a program  
    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 Program to Another User

The owner or program manager of a program object can assign the ownership of the program 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 Program Manager role for the delegated program object.

To delegate ownership of a program, use the `IProgram.delegateOwnership()` method. When you delegate ownership of a program, you automatically update the Delegated Owner field, which is read-only. The `delegateOwnership()` method lets you specify whether delegated ownership also applies to the program's children.

Example 14-8: Delegating ownership of a program 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 Program's Team

The Team table lets you manage the team member list for a program 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 program's Team table.

When you add a resource to the Team table, you specify what roles the user or user group has for that program object. The roles available are not the complete set of Agile PLM roles; they are roles specifically related to Program 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, see the *Agile PLM Administrator Guide*.

The Team table has two attributes that require special mention: `ProgramConstants.ATT_TEAM_NAME` and `ProgramConstants.ATT_TEAM_ROLES`. These are `SingleList` and `MultiList` attributes, respectively. To get available values for these attributes, use the `ITable.getAvailableValues()` method instead of `IAttribute.getAvailableValues()`. Otherwise, the `IAgileList` object returned from the method may contain invalid list values.

Example 14-9: Adding resources to a program's team

```
try {
```

```

// Get users
IUser user1 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "akurosawa");
IUser user2 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "creed");
IUser user3 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "dlean");
IUser user4 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "jford");

// Get a resource pool (user group)
IUserGroup pool = (IUserGroup)m_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 program
IProgram program = (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
if (program != null) {

    // Get the Team table of the program
    ITable teamTable = program.getTable(ProgramConstants.TABLE_TEAM);

    // Get Name attribute values (use ITable.getAvailableValues)
    IAgileList attrNameValues = teamTable.getAvailableValues(ProgramConstants.ATT_TEAM_NAME);

    // 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
    attrNameValues.setSelection(new Object[]{user1});
    attrRolesValues.setSelection(new Object[]{"Change Analyst", "Program Manager"});
    map.put(ProgramConstants.ATT_TEAM_NAME, attrNameValues);
    map.put(ProgramConstants.ATT_TEAM_ROLES, attrRolesValues);
    map.put(ProgramConstants.ATT_TEAM_ALLOCATION, new Integer(50));
    map.put(ProgramConstants.ATT_TEAM_ACTUAL_HOURS, new Integer(16));
    IRow row1 = table.createRow(map);

    // Add the second user to the team
    attrNameValues.setSelection(new Object[]{user2});
    attrRolesValues.setSelection(new Object[]{"Executive"});
    map.put(ProgramConstants.ATT_TEAM_NAME, attrNameValues);
    map.put(ProgramConstants.ATT_TEAM_ALLOCATION, new Integer(100));
    map.put(ProgramConstants.ATT_TEAM_ACTUAL_HOURS, new Integer(40));
    IRow row2 = table.createRow(map);

    // Add the resource pool to the team
    attrNameValues.setSelection(new Object[]{pool});
    attrRolesValues.setSelection(new Object[]{"Program Team Member"});
    map.put(ProgramConstants.ATT_TEAM_NAME, attrNameValues);
    IRow row3 = table.createRow(map);
}
} catch (APIException ex) {
    System.out.println(ex);
}

```

In the 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 program's Team table.

Example 14-10: 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 Program 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 program. To substitute program 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 program's children.

Example 14-11: Substituting program resources

```
try {

    // Get a program
    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);

    // Substitute u4 with a user group, and apply to children
    program.substituteResource(u4, ug, true);
}
} catch (APIException ex) {
    System.out.println(ex);
}

```

Locking or Unlocking a Program

The owner of a program can lock or unlock it. When a program is locked, its schedule cannot be modified. To lock or unlock a program, use the `IProgram.setLock()` method.

Note Programs are automatically locked when you use the Gantt Chart or the Microsoft Project integration functionality in the Agile Web Client.

Example 14-12: Locking a program

```

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 programs, are not routable objects. Therefore, discussions do not have workflows associated with them.

For a list of Discussion tables, see Table 4-2 on page 4-3.

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**

Of course, you should also specify data for the **Cover Page.Notify List** and **Cover Page.Message** attributes. Otherwise, the discussion won't have a notification list or a message that users can reply to.

The following example shows how to create a new discussion and add it to the Discussion table of a program.

Example 14-13: 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 program
    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 14-14: Replying to a discussion

```

private void replyToDiscussion() throws Exception {

    Iterator it;
    IDiscussion discussion;

    // Get a program
    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_MESSAGE);
        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 the Agile Web Client.

Figure 14-2: Expanded Replies table

Subject	Creator
[-] Stress testing	Dassin, Jules (jdassin)
RE: Stress testing	Hitchcock, Alfred (ahitchcock)
[-] RE: Stress testing	Reed, Carol (creed)
[-] RE: Stress testing	Huston, John (jhuston)
RE: Stress testing	Dassin, Jules (jdassin)

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 14-15: 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

The Agile Web Client allows users to join a discussion by clicking the **Discussion** tab of a program, 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 program.

Note If you aren't on the Notify List of a Discussion object, you cannot read the replies. However, anyone listed on the Team table of a program can join a discussion associated with that program.

Example 14-16: Joining a discussion

```

try {

    // Get a program
    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 program object. Make sure the map object used to initialize the row contains parameters for the Subject, Assigned To, and Due Date fields.

Example 14-17: Creating an action item

```

private void replyToDiscussion() throws Exception {

    // Get a program
    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
ITable table = program.getTable(ProgramConstants.TABLE_ACTIONITEMS);

// Add the new Action Item to table
table.createRow(map);
}
} catch (APIException ex) {
    System.out.println(ex);
}
```

Product Governance & Compliance

This chapter shows how to use the Agile Product Governance & Compliance APIs of Agile PLM to track restricted substances and materials contained in any item, manufacturer part, or part group. It contains the following topics:

- ❑ *About Agile Product Governance and Compliance*
 - ❑ *Agile PG&C Interfaces and Classes*
 - ❑ *Agile PG&C Roles*
 - ❑ *Creating Declarations, Specifications, and Substances*
 - ❑ *Adding Items, Manufacturer Parts, and Part Groups to a Declaration*
 - ❑ *Adding Substances to a Declaration*
 - ❑ *Adding Substances to a Specification*
 - ❑ *Adding Specifications to a Declaration*
 - ❑ *Routing Declarations*
 - ❑ *Filling Out a Declaration*
 - ❑ *Submitting a Declaration to the Compliance Manager*
 - ❑ *Publishing a Declaration*
 - ❑ *Getting and Setting Weight Values*
 - ❑ *Adding Substance Compositions for a Manufacturer Part*
-

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, see the separate *Product Governance & Compliance User Guide*.

Agile PG&C Interfaces and Classes

The following table lists Agile PG&C-related interfaces and classes:

Table 15-1: Agile PG&C-related interfaces and classes

Object	Interface	Constants Class
Declaration	IDeclaration	DeclarationConstants
Specification	ISpecification	SpecficationConstants
Substance	ISubstance	SubstanceConstants

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** — 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** — 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 or (Restricted) Material Provider roles. For more information about Agile PLM roles, see the *Agile PLM Administrator Guide*.

Creating Declarations, Specifications, and Substances

This section shows how to create each of the Agile 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 15-1: 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.

For a list of Specification tables, see Table 4-2 on page 4-3.

Example 15-2: 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.

As you can see, these substance subclasses comprise the hierarchy of objects that can appear on a Composition table, also known as the Bill of Substances.

For a list of Substance tables, see Table 4-2 on page 4-3.

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 15-3: 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 subClass = 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 15-4: 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 subClass = 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 15-5: 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 15-6: 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 subClass = m_session.getAdminInstance().
        getAgileClass(SubstanceConstants.CLASS_SUBSTANCE);

    // Create a new substance
    ISubstance substance = (ISubstance)m_session.createObject(subClass, params);
}
```

Adding Items, Manufacturer Parts, and Part Groups to a Declaration

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 15-7: 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_NUMBER, "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 a Declaration

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. The 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 required attributes the Map object must contain:

Table 15-2: Attributes required to create a new row on the Composition table

Composition Table	Required Attributes	Declaration Constants
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

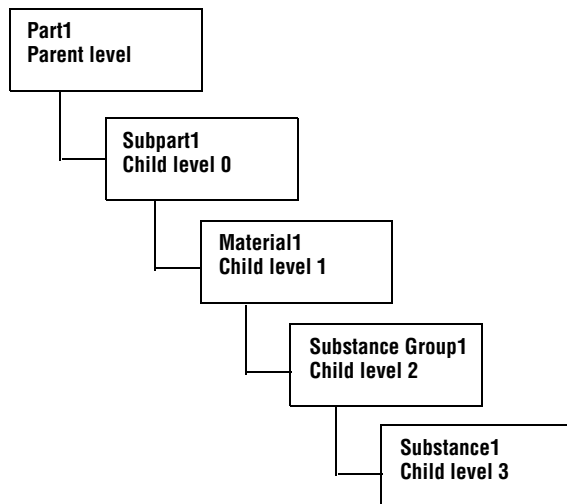
Structure of Bill of Substances

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.

Figure 15-1: Bill of Substances structure 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 15-8: 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_SUBSTANCE_NAME,
            dummyName);
        params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SUBSTANCE_TYPE,
            subtype);

        IRow dummyRow = tblMfrPartComp.createRow(params);
        return dummyRow;

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

Example: Adding Substances to the Manufacturer Part Composition Table of a Homogeneous Material Declaration

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 15-9: 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.CLASS_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 Homogeneous 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_NUMBER, "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_SUBSTANCE_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_SUBSTANCE_NAME, material);
        IRow rowMaterial = tblMfrPartComp.createRow(params);
    }
}
```



```

// Add a substance group
ISubstance sg = (ISubstance)m_session.getObject(SubstanceConstants.CLASS_SUBSTANCE_GROUP,
    "Lead Compounds");
params.clear();
params.put(DeclarationConstants.ATT_PARENT_ROW, rowMaterial);
params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SUBSTANCE_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_SUBSTANCE_NAME, sub);
IRow rowSubs = tblMfrPartComp.createRow(params);

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

```

Example: Adding Substances to the Manufacturer Part Composition Table of a Substance Declaration

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 15-10: 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.CLASS_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_NUMBER, "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_GROUP,
    "Lead Componds");
params.clear();
params.put(DeclarationConstants.ATT_PARENT_ROW, rowMfrParts);
params.put(DeclarationConstants.ATT_MANUFACTURER_PART_COMPOSITION_SUBSTANCE_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_SUBSTANCE_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 15-11: 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:

Table 15-3: Specification Rules for Declaration Types

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.

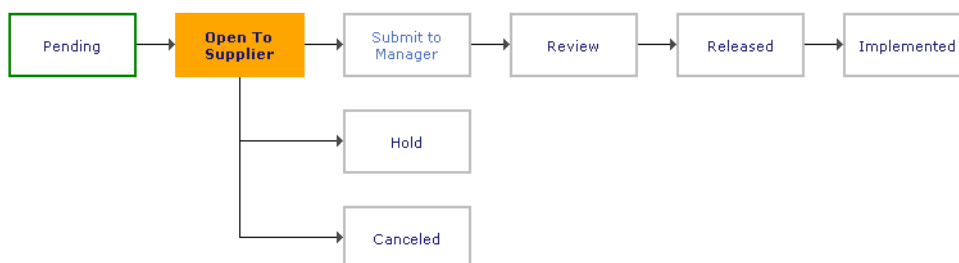
Example 15-12: Adding specifications to the Specification table

```
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 15-2: Default Declarations workflow



The following table describes each status in the Default Declarations workflow.

Table 15-4: Default Declarations workflow statuses

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 15-13: 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 15-14: 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 related to workflow processes, see Chapter 11, “Managing Workflow.”

Filling Out 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. For more information, see the next section.

Submitting a Declaration to the Compliance Manager

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 15-15: 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, the 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 15-16: 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 uommm = (IUnitOfMeasureManager)m_session.getManager(
        IUnitOfMeasureManager.class);
    uom = uommm.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 15-17: 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 a Manufacturer Part

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 15-8.

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 15-18: 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_PART_NUMBER,
            "Widget");
        params.put(ManufacturerPartConstants.ATT_GENERAL_INFO_MANUFACTURER_NAME, "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_TYPE,
            "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_GROUP,
            "Lead Compounds");
        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);
    }
}
```


Subscribing to Agile PLM Objects

This chapter describes how to use the Agile API to subscribe to an object and receive notification when the object triggers one of the selected events. It contains the following topics:

- ❑ *About User Subscriptions*
- ❑ *Getting the Subscriptions for an Object*
- ❑ *Modifying the Subscriptions for an Object*
- ❑ *Making Attributes Available for Subscription*
- ❑ *Working with the Subscription Table*

About User Subscriptions

When you load an Agile PLM business object, such as an item or change, you can subscribe to it. By subscribing to the object, you will receive a notification whenever a triggering event occurs for that object. You can specify which events trigger notification. Subscription events can be a lifecycle change, a change to attachment files, or a change to the value of any cell that has been made available for subscription.

You can subscribe to both routable and nonroutable objects. The Agile API provides an interface called `ISubscribable`, which lets you retrieve and modify 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.

Table 16-1: Subscription events

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 Program Execution objects, which aren't supported by the Agile API.

Although most routable and nonroutable objects support the seven subscription events listed in Table 16-1, 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 notifications: email and inbox. Agile PLM inbox notifications occur automatically regardless of user preferences. Email notifications are sent only if the user's Receive Email Notification preference is set to Yes.

Note The Agile API does not currently expose notification objects. However, you can use the Agile API to set the email notification preference.

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 the 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 the subscriptions for an object.

Example 16-1: 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_FILE)) {
            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 16-2: 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 16-3: 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 the Subscription Table

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 16-4: 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 16-5: 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 16-6: 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);
}
```


Creating Multithreaded Applications

This chapter shows how to develop multithreaded applications using the Agile API. It contains the following topics:

- ❑ *About Multithreading*
 - ❑ *Using the Multithreaded Sample Application*
 - ❑ *For More Information About Threads*
-

About Multithreading

Java provides built-in support for multithreading. A multithreaded program contains two or more parts that run independently but concurrently. Each part of the program is called a thread, and each thread defines a single sequential flow of control within the program.

The Agile API supports development of multithreaded applications. This means that Agile API objects have been designed and certified to be thread-safe. A thread can't write to an Agile API object at the same time that another thread is reading from it. Agile API methods are properly synchronized to prevent such conflicts.

Multithreaded applications are more efficient because they make use of the CPU's idle time. This is particularly important for Web-based applications that run over the Internet.

When you run a multithreaded application, the threads may not be apparent to the user. After all, you can't see them. However, there is a lot of work involved in programming a multithreaded application. First, you have to decide how to organize your program into multiple threads. Then, you have to decide how to implement them.

When to Use Threads

Threads are useful for the following purposes:

- ❑ **User interface processes** — If an application is busy processing, the user may wish to stop the process. For example, say your program is capable of running a process that takes several minutes to run, and the user interface has a Stop button to end the process. For the Stop button to work, it must be handled by a separate thread.
- ❑ **Multiple instances of a process** — Sometimes you want more than one copy of an object, window, or variable to be available in your program. For example, suppose you want your program to execute multiple Agile API queries concurrently. Each query process could be handled by a separate thread that runs independently.
- ❑ **Improving program speed and efficiency** — A multithreaded application shares objects and efficiently uses CPU time, resulting in improved overall performance.

Creating a Thread

A thread is an instance of an object of type `Thread`. There are two ways to create a thread in your program:

- ❑ Create a class that implements the `Runnable` interface
- ❑ Create a class that extends the `Thread` class, and then create an instance of that class.

Which method should you use? If your class must subclass another class (such as the `Applet` class), use `Runnable`. If your class needs to override any methods of the `Thread` class, then extend the `Thread` class.

Thread Class Methods

To write a multithreaded program, you should know something of the Java `Thread` class and the `Runnable` interface. The `Thread` class encapsulates a single thread of execution. The `Runnable` interface is a companion to `Thread`. You can construct a thread on any class that implements `Runnable`.

Table 17-1: Selected `Thread` class methods

Method(s)	Description
<code>getName()</code> , <code>setName()</code>	Gets or sets the name of the thread.
<code>getPriority()</code> , <code>setPriority()</code>	Gets or sets the priority of the thread.
<code>interrupt()</code>	Interrupts the thread.
<code>isAlive()</code>	Tests if the thread is alive.
<code>join()</code>	Waits for a thread to die.
<code>run()</code>	Subclasses of <code>Thread</code> should override this method. Otherwise, it calls a <code>Runnable</code> object's <code>run</code> method.
<code>sleep()</code>	Suspend a thread for a specified number of milliseconds.
<code>start()</code>	Start a thread by calling its <code>run</code> method.

For more information about the `Thread` class, see the Java API documentation.

Using the Multithreaded Sample Application

The Agile SDK includes a multithreaded sample application called MTS. The application demonstrates how to run multiple queries concurrently in separate threads. Due to the efficient use of threads, the queries run faster.

After you install Agile SDK sample programs, MTS is located in the `\<agile_home>\integration\sdk\samples` folder.

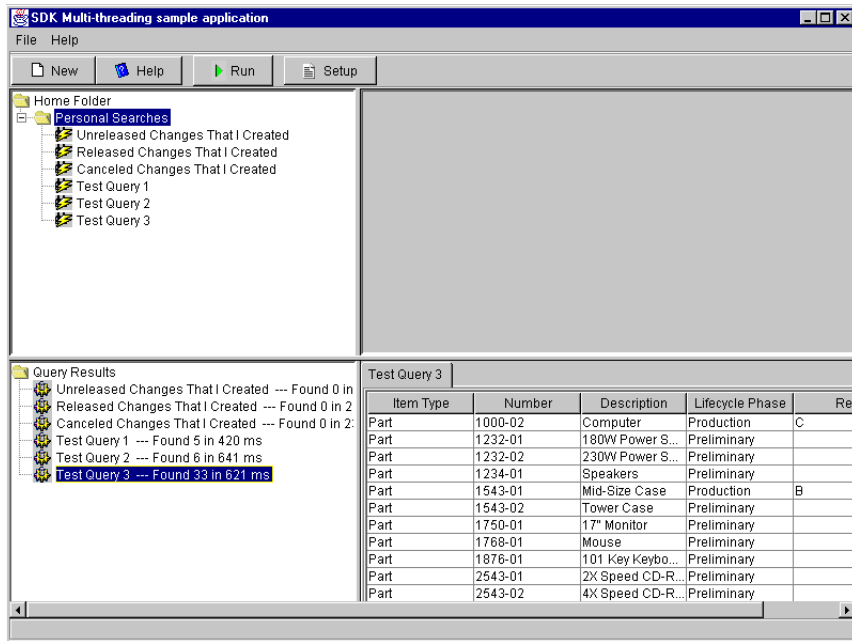
How the Multithreaded Sample Works

MTS is a Swing application. There are three folders in the MTS project:

- ❑ `app` — contains the application, or program logic, classes.
- ❑ `config` — contains a configuration file for connecting to Agile PLM servers.
- ❑ `gui` — contains user interface classes.

When you run MTS, it displays a multipane window, shown in Figure 17-1. The application also has a menu and toolbar.

Figure 17-1: MTS window



The top-left pane of MTS has a tree control that displays the personal queries for the current user. It does not display public (global) or system queries.

Use these toolbar buttons to control the MTS application:

- ❑ **New** — Adds a new query with one search condition.
- ❑ **Run** — Runs all the personal queries concurrently.
- ❑ **Setup** — Changes settings used to connect to Agile PLM servers.

The bottom-left pane has a tree control that displays a node for each query's results. When you select a node, the results data appears in the bottom-right pane.

Running Queries as Independent Threads

When you click the **Run** button, MTS calls the `executeAllQueries()` method of the `Mts` class. `executeAllQueries()` retrieves the list of personal queries and then runs each query.

Example 17-1: `executeAllQueries` method

```
/**
 * Execute all the queries as multiple threads
 */
public void executeAllQueries() {
    folderQryList = getAllPersonalQueries();
    buildThreadTreeNode(folderQryList);
    Vector ql = mtsFolder.getQueries();
    IQuery qry;
    FolderQuery fq;

    for (int i = 0; i < ql.size(); i++) {
        qry = (IQuery) ql.get(i);
        //same mtsQry is set with multiple queries
    }
}
```

```
        //single instance of mtsQry object will be executed by multiple threads
        MtsQuery newMtsQry = new MtsQuery(mtsSession);
        newMtsQry.setQuery(qry);
        fq = (FolderQuery) folderQryList.get(i);
        executeQuery(newMtsQry, fq.getQryIndex());
    }
}
```

The `executeAllQueries()` method calls the `executeQuery()` method for each query in the list. `executeQuery()` creates and starts a new `QueryThread` instance for each query.

Example 17-2: `executeQuery` method

```
/**
 * Execute each query as thread
 */
public void executeQuery(MtsQuery mtsQry, int queryIndex) {
    queryThread = new QueryThread(this, mtsQry, queryIndex);
    queryThread.start();
}
```

Processing Queries using MtsQuery

The `MtsQuery` class is used to construct, execute, retrieve, and store the results of each query MTS runs.

Example 17-3: `MtsQuery` methods

```
/**
 * Construct a new query and add it to the Agile PLM server
 */
public void addQuery(String queryClass, String attName, String relOp, String searchValue) {
    ...
}
/**
 * Return the Agile Query object
 */
public IQuery getQuery() {
    return qry;
}
/**
 * Put the query results in an array of Vectors
 */
public Vector[] getResults() {
    ...
}
/**
 * Run the query
 */
public void runQuery() {
    Vector[] rs = null;
    try {
        qryTbl = qry.execute();
        qryIterator = qry.execute().iterator();
    } catch (APIException ex) {
        ex.printStackTrace();
    }
}
/**
 * Set the Query object to this class without constructing
 * it from scratch
 */
public void setQuery(IQuery qry) {
```

```

    this.qry = qry;
}

```

Implementing the QueryThread Class

The `QueryThread` class is the most important class of MTS because it defines the thread used for each query. Although `QueryThread` extends `Thread`, it overrides only the `run()` method. It could also have been implemented as a `Runnable` class.

Example 17-4: `QueryThread` class

```

package mts.app;

import com.agile.api.*;
import java.util.*;

public class QueryThread extends Thread {

    private static Object[] results;
    //private Object[] results;
    Mts mts;
    int queryIndex;
    int qryCount = 0;
    MtsQuery mtsQry;

    /**
     * constructor: query items
     */
    public QueryThread(Mts mts, MtsQuery mtsQry, int index) {
        this.mts = mts;
        qryCount = mts.folderQryList.size();
        results = new Object[qryCount];

        this.mtsQry = mtsQry;
        queryIndex = index;
    }

    /**
     * implement thread's run method
     */
    public void run() {
        query(queryIndex);
    }

    /**
     * when the query results are put in Vector[], notify the Thread
     * tree to update the GUI tree with number of records to update
     */
    public synchronized void queryFinished(int queryIndex, int qryRecords,
    long time) {
        mts.getThreadTree().queryFinished(queryIndex, qryRecords, time);
    }

    /**
     * when the query is started, notify the Thread tree to update the
     * GUI tree with number of records to update
     */
    public synchronized void queryStarted(int queryIndex) {
        mts.getThreadTree().queryStarted(queryIndex);
    }
}

```

```
/**
 * create, execute and finish the query
 */
public synchronized void query(int queryIndex) {
    long startTime = System.currentTimeMillis();
    queryStarted(queryIndex);
    mtsQry.runQuery();
    Vector rs[] = mtsQry.getResults();
    long endTime = System.currentTimeMillis();
    long time = endTime - startTime;
    queryFinished(queryIndex, rs.length-1, time);
    storeResults(rs, queryIndex);
    rs = null;
}

/**
 * Store the result in results array.
 * This method is synchronized to avoid multiple threads
 * from accessing the results global variable.
 */
private synchronized void storeResults(Vector[] rs, int queryIndex) {
    results[queryIndex] = rs;
}

/**
 * Get the result for given query
 */
public static Vector[] getResult(int queryIndex) {
    return (Vector[]) results[queryIndex];
}
}
```

For More Information About Threads

There is a lot more you can learn about threads. Check other Java reference sources for the following information:

- ❑ How to control interactions between threads
- ❑ How to avoid deadlocks in threads
- ❑ How to set thread priorities

Sun's Java Web site contains much helpful information about threads. There is an online Java Tutorial at the following URL:

<http://java.sun.com/docs/books/tutorial/index.html>

For the section of the Java Tutorial specifically on threads, see this URL:

<http://java.sun.com/docs/books/tutorial/essential/threads/index.html>

Note The location of Sun's Java documentation pages may have changed. If so, try Sun's Java Web site, <http://java.sun.com/>, and from there go to the Documentation page.

Performing Administrative Tasks

This chapter provides code examples that show how to use the Agile API to perform administrative tasks on the Agile Application Server. It contains the following topics:

- ❑ *About Agile PLM Administration*
 - ❑ *Privileges Required to Administer Agile PLM*
 - ❑ *Administrative Interfaces*
 - ❑ *Getting an IAdmin Instance*
 - ❑ *Working with Nodes*
 - ❑ *Managing Agile PLM Classes*
 - ❑ *Working with Attributes*
 - ❑ *Working with Properties of Administrative Nodes*
 - ❑ *Managing Users*
 - ❑ *Managing User Groups*
-

About Agile PLM Administration

The 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 dataobjects; 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, please see your 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.

Table 18-1: Administrative interfaces

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 one of the Admin nodes 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. Use ITreeNode instead.
IUser	An Agile PLM user.
IUserGroup	A user group. You can use user groups to define project teams, site-related groups, departments, and global groups.
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 18-1: 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.

The following table identifies how Agile Java Client nodes map to Agile API administrative functionality.

Table 18-2: Mapping Agile Java Client nodes to the Agile API

Agile Java Client node	Agile API equivalent
Data Settings	
Classes	<code>NodeConstants.NODE_AGILE_CLASSES</code>
Character Sets	<code>NodeConstants.NODE_CHARACTER_SETS</code>
Lists	Not supported
Process Extensions	Not supported
AutoNumbers	<code>NodeConstants.NODE_AUTONUMBERS</code>
Criteria	<code>NodeConstants.NODE_CRITERIA_LIBRARY</code>
Workflow Settings	
Workflows	<code>NodeConstants.NODE_AGILE_WORKFLOWS</code>
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	<code>NodeConstants.NODE_ROLES</code>
Privileges	<code>NodeConstants.NODE_PRIVILEGES</code>
User Monitor	Not supported
Deleted Users	Not supported
Deleted User Groups	Not supported

Table 18-2: Mapping Agile Java Client nodes to the Agile API (continued)

Agile Java Client node	Agile API equivalent
System Settings	
SmartRules	NodeConstants.NODE_SMARTRULES
Viewer & Files	NodeConstants.NODE_VIEWER_AND_FILES
Notifications	NodeConstants.NODE_NOTIFICATION_TEMPLATES
Full Text Search	Not supported
UOM	Not supported
Company Profile	Not supported
Currency Exchange Rates	IAdmin.getConversionRates()
Commodities	Not supported
Product Cost Management	
Ship To Locations	Not supported
Program Execution	
Program 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	
Example Roles	Not supported
Example Privileges	Not supported
Example Criteria	Not supported

The 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.

Table 18-3: Mapping Agile Web Client administrative functionality 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 the 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 the 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 18-2: 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 18-3: 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 18-4: 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 roughly equivalent to the `IAgileClass` objects 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, we recommend 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's much easier to work with `IAgileClass` objects. For more information, see "Managing Agile PLM Classes" below.

Managing Agile PLM Classes

The `Agile Classes` node provides a framework for classifying Agile PLM objects, such as parts, changes, and packages. Using the `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 available on your `Agile PLM` system depend on the `Agile PLM` server licenses your company has purchased.

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.

Table 18-4: Agile PLM base classes, classes, and 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
	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
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
Programs	Activities	Phase
		Program
		Task
	Gates	Gate
Quality Change Requests	Audits	Audit
	Corrective and Preventive Actions	CAPA
Reports ¹	Custom Reports	Custom Report
	External Reports	External Report
	Standard Reports	Administrator Report
		Standard Report
Requests for Quote	Requests for Quote	RFQ

Table 18-4: Agile PLM base classes, classes, and subclasses (continued)

Base Class	Classes	Predefined Subclasses
RFQ Responses	RFQ Responses	RFQ Response
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:

- 1 Report objects are not supported by the Agile API.

Concrete and Abstract Classes

Agile PLM superclasses, like Item 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 18-5: 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 18-6: 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 18-7: 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 18-8: 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 has been created or opened in your program. For more information about `ICell` objects, see Chapter 5, "Working with Data Cells."

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 a suffix name "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, Agile recommends referencing attributes by ID number or constant. However, many of the examples in this manual reference attributes by name simply to make them more readable.

The following example shows how to reference an attribute ID constant.

Example 18-9: 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 on 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 18-10: 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.

Table 18-5: Retrieving class attributes

Method	Description
<code>IAgileClass.getAttribute()</code>	Retrieves the specified <i>IAttribute</i> object for a class.
<code>IAgileClass.getAttributes()</code>	Retrieves an array of <i>IAttribute</i> objects for all tables of a class.
<code>IAgileClass.getTableAttributes()</code>	Retrieves an array of <i>IAttribute</i> objects for a specified table of the class.
<code>ITable.getAttributes()</code>	Retrieves an array of <i>IAttribute</i> objects for a table.
<code>ICell.getAttribute()</code>	Retrieves the <i>IAttribute</i> object for a cell.

The following example shows how to retrieve BOM table attributes.

Example 18-11: 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 18-12: 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 18-13: 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 the 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 18-14: 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;
}
```

`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 5-5.

Managing Users

Users are dataobjects 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 18-15: 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 dataobjects 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 18-16: 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 18-16.

Creating a Supplier User

Supplier users are by default assigned the Restricted user category, which restricts 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 18-17: 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 licenses, 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 username and login password.

Example 18-18: 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

Agile PLM passwords can be set to expire over time. When a user's login password expires, the user cannot log in to the Agile Application Server. When a user's approval password expires, he is prevented from approving a change. If either one or both of the user's Agile PLM passwords has expired, your Agile API program should 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 18-19: Checking for expired passwords

```
private void login(String username, String password) {
    try {
        HashMap params = new HashMap();
        params.put(AgileSessionFactory.USERNAME, username);
        params.put(AgileSessionFactory.PASSWORD, password);
        AgileSessionFactory instance =
            AgileSessionFactory.getInstance("http://agileserver/virtualPath");
        m_session = instance.createSession(params);
    } catch (APIException ex) {
        if (ex.getErrorCode().equals(ExceptionConstants.API_MUST_CHANGE_BOTH_PWDS))
            System.out.println("Login Failed. You must change both your login and approval passwords.");
        else if (ex.getErrorCode().equals(ExceptionConstants.API_MUST_CHANGE_LOGIN_PWD))
            System.out.println("Login Failed. You must change your login password.");
        else
            System.out.println(ex.getMessage());
    }
}
```

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 18-20: 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** — 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** — Can log in to the server only if a concurrent user license is available.
- ☐ **Restricted** — Users with 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 licenses, roles, and privileges, see the *Agile PLM Administrator Guide*.

The following example shows how to set a user's User Category and Roles settings.

Example 18-21: 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" });

    //Change the cell value
    cell.setValue(roles);
}
```

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 18-22: Deleting a user

```
private void removeUser(IUser user) throws APIException {
    user.delete();
    user = null;
}
```

Note In the Agile Java Client, deleted users can be listed by choosing **Admin > User Settings > Deleted Users**.

Managing User Groups

A user group is, quite simply, 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 the 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 18-23: 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_USERS) + ", " +
            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.

Of course, when you create 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 18-24: 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);
I AgileList 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 18-25: 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
    I AgileList 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);
}

```

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 18-26: 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_NAME));
    }
}

```


CHAPTER 19

Handling Exceptions

This chapter shows how to catch Agile API exceptions. It contains the following topics:

- ❑ *About Exceptions*
- ❑ *Exception Constants*
- ❑ *Getting Error Codes*
- ❑ *Getting Error Messages*
- ❑ *Disabling and Enabling Warning Messages*

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 should

- ❑ 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`.

Table 19-1 lists the `APIException` class methods for handling exceptions:

Table 19-1: `APIException` methods

Method	Description
<code>getErrorCode()</code>	Returns the number of the error code associated with the <code>APIException</code> .
<code>getMessage()</code>	Returns the error message associated with the <code>APIException</code> .
<code>getRootCause()</code>	Returns the root cause of the <code>APIException</code> , if any.
<code>getType()</code>	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, see the Agile API HTML reference.

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 program, you can disable them. For more information, see "Disabling and Enabling Warning Messages" on page 19-2.

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 19-1: 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_WARNING))
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Warning", JOptionPane.YES_NO_OPTION);
    }
}
```

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 19-2: 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, see the Agile API HTML reference under `ExceptionConstants`.

Disabling and Enabling Warning Messages

Several Agile PLM error messages are warnings that let you 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 19-3: 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_WARNING);
                    // Release the Change
                    chgObj.changeStatus(nextStatus, false, "", true, true, null, null, null, false);
                    // Enable all warnings
                    m_session.enableWarning(ExceptionConstants.APDM_UNRESPONDEDCHANGE_WARNING);
                } catch (APIException exc) {}
            }
        }
    }
}

```

Checking if an APIException is a Warning and not an 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 the 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 19-4: 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;
    }
}

```

```
}  
}
```

Delete Warnings Disabled Automatically by the 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, see “Deleting and Undeleting Objects” on page 19-2.

Saving and Restoring the State of 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 19-5: 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_WARNING);  
  
    // 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();  
}
```

PART II

Developing Agile Extensions

Part II provides details on process extensions and web service extensions, which let you extend the functionality of the Agile server. It includes the following chapters:

- ❑ *Chapter 20, Developing Process Extensions*
 - ❑ *Chapter 21, Developing Web Service Extensions*
-

CHAPTER 20

Developing Process Extensions

This chapter contains information about developing process extensions for the Agile PLM server. It has the following sections:

- ❑ *About Process Extensions*
 - ❑ *Developing Custom Autonumber Sources*
 - ❑ *Developing Custom Actions*
 - ❑ *Defining URL-Based Process Extensions*
 - ❑ *Creating an External Report*
 - ❑ *Deploying Process Extensions in a Clustered Environment*
 - ❑ *Process Extensions FAQ*
-

About Process Extensions

Process extensions 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.

Process extensions allow 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 opens up the Agile PLM system, allowing you to tailor it to your business.

A process extension is either a Java class deployed on the Agile Application Server or a link to an URL. The URL can be a simple Web site or the location of a Web-based application.

Process extensions can be used 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 `webserviceextensionsframework`.

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 State
- ❑ 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 the 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 20-1: 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 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 the Agile Java Client

In the 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 into the 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 20-1: Define the Autonumber dialog box

6 Enter the following information:

- **Name** — Enter the name of the autonumber source.
- **Description** — Enter a brief description of the autonumber source.
- **Enabled** — Select Yes or No.
- **AutoNumber type** — Select Custom. This activates the Custom AutoNumber and TimeOut fields.
- **Where Used** — Select the subclass(es) that can use the autonumber source.
- **Custom AutoNumber** — Select a custom autonumber source from the list.
- **TimeOut** — Enter the number of seconds after which a timeout error will occur if the custom autonumber source does not respond. The default is 30 seconds. Make sure the timeout setting is long enough to ensure that the code runs to completion. Otherwise, it won't generate a number for the object.

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 into the 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 State

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 20-2: 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");
    }
}
```

Of course, the above `HelloWorld` class doesn’t perform a useful action. It simply demonstrates how to implement the class for a custom action.

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 should never directly create new Agile PLM sessions.

If a process extension makes use of web services to invoke a web service extension (WSX), that WSX will likely create a new Agile PLM session, which is distinct from the session used by the process extension. However, if you have written a 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.

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 20-11.

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 the 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 the Agile Java Client

In the 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.

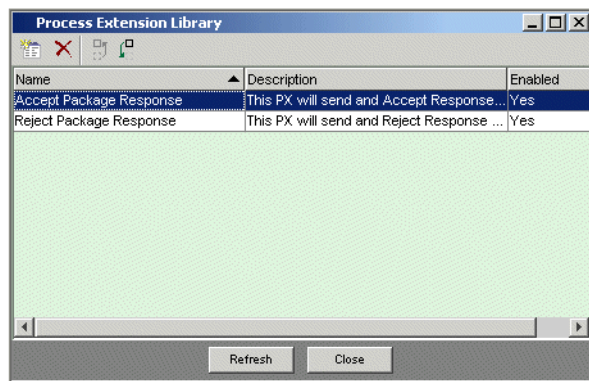
Using the Process Extension Library

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:

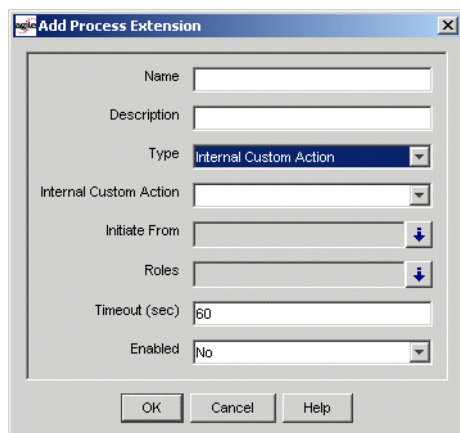
- 1 Log into the Agile Java Client as an administrator.
- 2 Click the **Admin** tab.
- 3 Open the **Data Settings** folder.
- 4 Open the **Process Extensions** node.

Figure 20-2: Process Extension Library



- 5 In the Process Extension Library window, click . The Add Process Extension dialog box appears.

Figure 20-3: Add Process Extension dialog box



- 6 Enter the following information:
 - **Name** — Enter the name of the process extension.
 - **Description** — Enter 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 state** — Invokes the custom action whenever a properly configured workflow enters a particular status. If the process extension is an URL, the Workflow State option is unavailable.

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 the Agile Web Client to create the report. If you specify that a process extension is initiated from the Tools menu, it's available at all times 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.
- **Timeout** — Enter the number of seconds the client will be blocked if the custom action does not respond. If Timeout equals -1, the client will be blocked until the custom action returns a result. The default timeout is 60 seconds. URL-based process extensions do not have a timeout.
- **Enabled** — Select Yes or No.

7 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 into the 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:

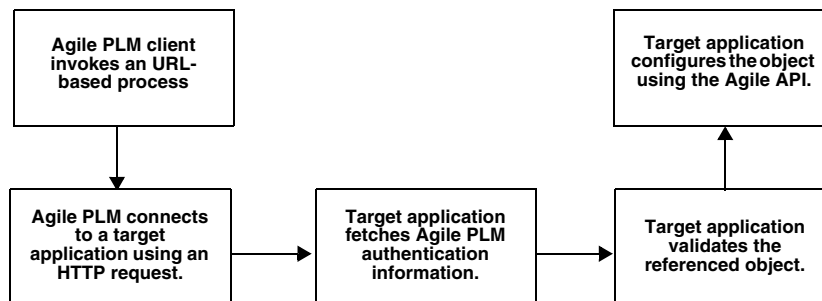
- 1 Log into the Agile Java Client as an administrator.
- 2 Click the **Admin** tab.
- 3 Open the **Workflow Settings** folder.
- 4 Open the **Workflows** node.
- 5 In the Workflows window, double-click a workflow
- 6 Click the **Status** tab.
- 7 Select a status other than Pending. The selected status 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 URL-Based Process Extensions

A process extension can be defined as a reference to either custom code deployed on the Agile Application Server or an URL or Web page. When the Agile PLM client invokes a process extension that references an 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 20-4: Process flow for a possible URL-based process extension



URL-based process extensions can also be used to reference a Web-based report engine. To create an external report that uses an URL-based process extension, choose **Create > Report > External** in the Agile Web Client. For more information, see “Creating an External Report” on page 20-14.

URL-based process extensions cannot be initiated by a change in a workflow status. This is because an Agile PLM client may not be active in triggering the change in status.

Note URL-based process extensions are not supported for sourcing projects (IProject).

To define an URL-based process extension:

- 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** — Enter the name of the process extension.
 - **Description** — Enter a brief description of the process extension.
 - **Type** — Select 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.
 - **External report** — Allows you to generate a report by accessing the Web page.
 - **Tools menu** — Allows you 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 a process extension is used to generate an external report, you can use the Agile Web Client to create the report. If you specify that a process extension is initiated from the Tools menu, it's 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 9.2.1 does not support single sign-on via password protected external application servers.

Important Single sign-on only applies if you run a URL PX in the same web container. If you want to password protect access to an external Application Server, you need to hardcode the username and password to access the external servlet into your code.

If an 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 an 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 an 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 an URL-based process extension from the Agile Web Client.
- ☐ 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 an URL-based process extension initiated from the 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 the 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 username and password. When you initiate an URL-based process extension from the Agile Web Client, the target system can use cookies to create an Agile PLM session. In effect, the Agile Web Client and the Agile API client on the target system can share a single signon.

Note The Agile Java Client, unlike the Web Client, does not create client-side cookies. Therefore, it does not support the single signon 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”, all computers ending with “.agile.agilesoft.com” can use the `j_username` and `j_password` cookies.

For more information about cookies, see the following Web site:

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 20-3: 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 20-4: 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;
}
```

Creating an Agile Session with AuthorizationHandler

The current WSX Security support allows you to create an Agile session with the `AuthorizationHandler`. If the user specifies the `AuthorizationHandler`, the implicit session is owned and managed by the application server. If the user provides basic authentication with the incoming WSX HTTP request, the user indicated by that authentication will be in effect for the implicit session.

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.

To avoid failure setting up an SDK session with username and password hardcoded within the service, use the following workaround for setting up an SDK session.

Example 20-5 shows the previous function of setting up a session within a WSX. Refer to the workaround that follows this example in order to set up a session successfully if the WSX makes use of the AIS authorization handler.

Example 20-5: Setting Up a Session Within a WSX

```
AgileSessionFactory factory = AgileSessionFactory.getInstance(null);
HashMap params = new HashMap();
params.put(AgileSessionFactory.USERNAME, ...);
params.put(AgileSessionFactory.PASSWORD, ...);
IAgileSession session = factory.createSession(params);
```

Security Handler Operation

You may expect the SDK to throw an exception if you create an `AgileSession` without the `Authorization Handler` in deployment. However, depending on which application server you use to authenticate users, the SDK may *not* throw an exception. This is because different application servers treat HTTP authentication differently. For example, Oracle application servers will always authenticate a user when presented with authentication credentials in the HTTP headers. The WAS will not do this unless you program it to do so.

Using the AIS Authorization Handler

In Example 20-5 above, the `AgileSessionFactory` did not have to be initialized with a target server because the SDK implicitly targets the hosting server. If the WSX makes use of the AIS authorization handler, you cannot use the construct in Example 20-5. Instead, change the first line in Example 20-6 below. Instead of using `AgileSessionFactory.getInstance(null)`, you should specify the URL to the system.

This causes the SDK to explicitly set up the `AgileSessionFactory`, and allow an explicit session to be created that is independent of the implicit session.

Example 20-6:

```
AgileSessionFactory factory = AgileSessionFactory.getInstance("http://...");
```

Retrieving an Agile PLM Object from an HTTP Request

If you invoke an 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 20-7: 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.

Table 20-1: Identifying attributes for Agile PLM classes

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.2000004284	Title Block.Name
Declaration	agile.classID	Class ID of selected object
	agile.agile.2000002615	Title Block.Name
Discussion	agile.classID	Class ID of selected object
	agile.18417	Cover Page.Number
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

Table 20-1: Identifying attributes for Agile PLM classes (continued)

Class	Parameter	Description
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
Report ¹	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
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:

- 1 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 the 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” above.

To create reports in the Agile Web Client, you must have the Create Reports privilege.

To create an external report:

- 1 Log into the Agile Web Client.

Note You cannot create external reports in the Agile Java Client.

- 2 Choose **Create > Report > External**. The Report Creation Wizard appears.
- 3 Type the name of the report. Click **Next**.

- 4 Enter the following General Information:
 - **Description** — Enter a brief description of the report.
 - **Process Extension** — Select a process extension. The process extension you select is associated with an 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 has not been run on a server in the application server cluster, the `/<agile_home>/integration/sdk/extensions` folder won't exist on that server. If so, you should 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 process extension JAR files in the `/<agile_home>/integration/sdk/extensions` folder on each server in the cluster.

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 currently supports two types of process extensions actions, 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 an URL. The URL can be a simple Web site or the location of a Web-based application.

What programming language does process extensions support?

If you are developing an internal process extension program, only Java is supported. However, if the process extension is simply a link to an URL, other programming languages, such as C++ and Visual Basic.Net, can be used to develop a Web-based application.

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.

Are custom actions synchronous or asynchronous?

Technically, all custom actions are asynchronous. Once a custom action is initiated, it runs to completion regardless of what the Agile PLM client does. If you initiate a custom action by selecting it from the Tools menu or Actions menu, you generally want the client to be blocked until the result is returned. The timeout parameter determines how long the client is blocked while waiting for a result. If the client times out, it proceeds as if no result was received from the custom action. Meanwhile, the custom action continues running until completion.

If a custom action is initiated by a workflow status change, you may not want to block the client for any length of time. In that case, you can set the timeout value to 0. However, if the custom action updates field values or moves the change to the next status, you may want to block the client long enough so that the user interface accurately reflects the results of the custom action. In that case, set the timeout value to an appropriate time to allow the custom action to complete.

Remember, custom autonumber sources are different from custom actions. A custom autonumber source returns a value to the invoking process so that it can be displayed in the client user interface. It doesn't write an object directly to the Agile PLM database. The custom autonumber source must complete and return its value before the timeout limit. Otherwise, the invoking process won't receive the value. Therefore, you should always make sure the timeout setting for a custom autonumber source is long enough to ensure that the code runs to completion.

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 the 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 roles and privileges of the current user. Once the custom action is completed, the client reverts to the user's roles and privileges.

How are process extensions similar to web service extensions?

Both process extensions and web service extensions can use Agile's Java API and they also run on the Agile Application Server within the server context. However, process extensions do not make use of the web services invocation mechanism. Instead, they make direct Java method calls into custom code after performing security and thread initialization. JAR files for process extensions are deployed in the same `<agile_home>/integration/sdk/extensions` folder as Agile PLM web services.

If a process extension is used to call Agile Integration Services or web service extensions, is another Agile PLM session needed?

Yes, if a process extension calls AIS, another session is created. However, if you want to call a web service extension from within a process extension, you can avoid the web service infrastructure altogether and invoke WSX Java methods directly.

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 names, 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 the 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).

Do process extensions actions run indefinitely or do they timeout eventually if no activity occurs?

Process extensions actions can be configured to run indefinitely, but most will have limits set on the length of time it takes the action to complete. Each process extension action runs in a separate thread. You can set a timeout value for custom actions. The default timeout is 60 seconds.

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 the 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 the 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 9.2.1, 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\.

Developing Web Service Extensions

This chapter contains information about developing web service extensions for the Agile PLM server. It has the following sections:

- ❑ *About Web Service Extensions*
 - ❑ *About Web Services*
 - ❑ *Developing and Deploying a Web Service*
 - ❑ *Defining a Web Service Entry Point*
 - ❑ *Authenticating Users*
 - ❑ *Building and Deploying the MyFirstWebService Sample*
 - ❑ *About Web Service Clients*
 - ❑ *Creating MyFirstClient*
 - ❑ *Microsoft .NET Interoperability*
 - ❑ *Web Service Extensions FAQ*
-

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.

WSX can be used 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), a separately licensed product, are web services built with WSX technology that provide programmatic import and export capabilities for the Agile PLM system. For more information about AIS, see the *Agile Integration Services Developer Guide*.

Key Features

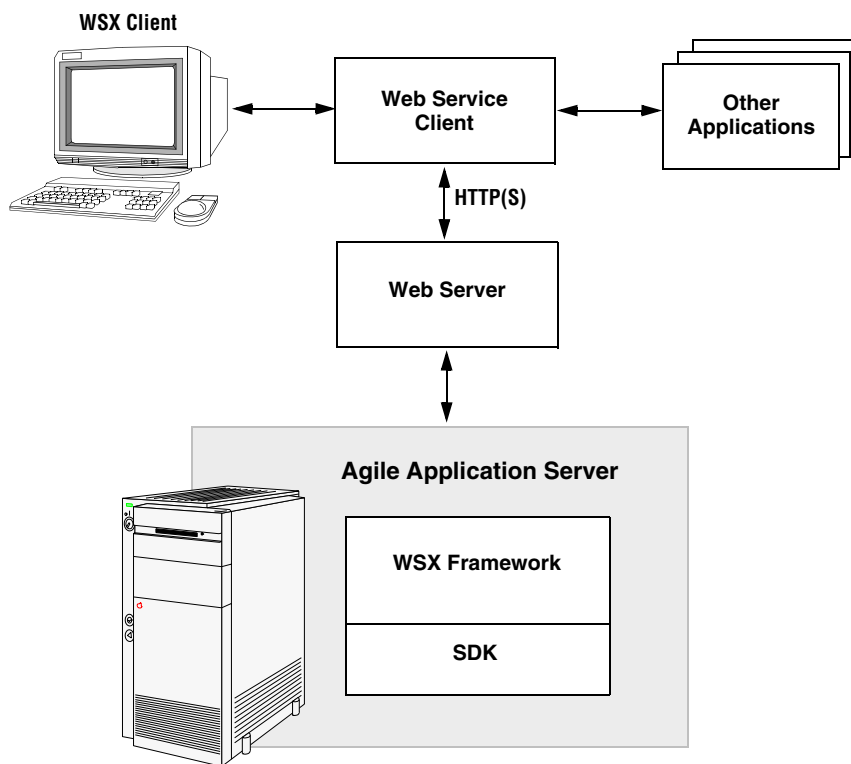
WSX includes the following key features:

- ❑ **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 21-1: 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 of 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**—provides the service by implementing it and making it available on the Internet.
 - **Service requestor**—user of the service who accesses the service by opening a network connection and sending an XML request.
 - **Service registry**—a centralized directory of services where developers can publish new services or 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 reusable 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. You can also specify an `AuthorizationHandler` in the web service deployment descriptor; for more information about how to specify an `AuthorizationHandler`, see “Authenticating Users” on page 21-6. Finally, you can pass the username and password by embedding them in the URL used to invoke the web service.

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, see the AXIS web site: <http://xml.apache.org/axis>

Further Information about Web Services

Here are some helpful 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://sunonedev.sun.com/building/development/developer_platform_overview.html
- ❑ 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, see the following Axis documentation:

- ❑ *Axis User's Guide* — <http://cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/user-guide.html>
See the sections entitled “Custom Deployment - Introducing WSDD” and “Service Styles - RPC, Document, Wrapped, and Message.”
- ❑ *Axis Reference Guide* — <http://cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/reference.html>
See the sections entitled “Deployment (WSDD) Reference.”

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

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 an 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://cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/user-guide.html#DataMapping>.

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

Web services can be accessed without requiring a username/password to be provided as part of the request. However, to provide the authorization functionality (that is, if you want your web services to be username/password protected), you can add the following declaration to your service deployment descriptor:

```
<requestFlow>
  <handler type="java:com.agile.wsx.security.AuthorizationHandler"/>
</requestFlow>
```


For example:

```
<service xmlns:sample="http://www.agile.com/ws/SampleWsx" name="MyFirstWebService"
  style="rpc" use="encoded">
  <requestFlow>
    <handler type="java:com.agile.wsx.security.AuthorizationHandler"/>
  </requestFlow>
  <parameter name="allowedMethods" value="*" />
  <parameter name="className" value="first.MyFirstWebService" />
  <parameter name="wsdlTargetNamespace" value="http://www.agile.com/ws/SampleWsx" />
  <parameter name="sendMultiRefs" value="false" />
</service>
```

By adding the `requestFlow` as above, the web service invocation is inspected to ensure that it contains username/password information and that information is used to set up a user session that can be used within the web service. If the web service uses the Agile API, then a call to `AgileSessionFactory.createSession(null)` uses the user session that was established during authorization.

If your web service uses an authorization handler, your web service client must set the username and password for each request. To set the username and password for your client, call the `setUsername()` and `setPassword()` methods available from the client-side stubs. For example, the `MyFirstClient` sample uses the following code to set the username and password:

```
// Configure the stub with the necessary authentication information
stub.setUsername(cl.getOptionValue(USER_SHRT));
stub.setPassword(cl.getOptionValue(PASSWORD_SHRT));
```

Building and Deploying the MyFirstWebService Sample

In order to illustrate the simplicity of developing a web service, a sample has been provided that highlights the development process. The sample, `MyFirstWebService`, is a relatively simple example that demonstrates how to create a web service that can use the Agile SDK in order to retrieve information about a particular item and return that as the result of the web service operation.

In order to support the desired operation, the following entry point has been defined:

```
public String getItemField(String[] args) throws RemoteException
```

A third party library, Jakarta Commons CLI, is used to parse the args as if it were a set of command line arguments. Based on those arguments, the results are returned as a `String`. You can look at the sample, located at `AGILE_HOME\integration\sdk\samples\wsx\src\first`, for more information on the implementation details. This section is concerned with the deployment process rather than implementation details.

Downloading Tools to Build the Sample

Before you can build and deploy the `MyFirstWebService` sample, you need to download the following tools:

Table 21-1: Tools to download

Tool	Download Site
Java 2 SDK SE Version 1.4.2	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 for installing the Java SDK on Windows and on Solaris. 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:\j2sdk142).

To install the Java SDK on Solaris:

- 1 Execute the distribution (for example, \$./j2sdk-1_4_2-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>/j2sdk142).
- 3 Execute your .profile or .cshrc (depending on your shell) file to reinitialize your environment settings.

Installing Ant

This section provides the instructions for installing Ant on Windows and on Solaris.

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 should be 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 the MyFirstWebService Sample

Agile provides several sample programs for the SDK, including a sample web service called MyFirstWebService. You can install SDK documentation and sample programs by using the Agile Documentation Installer, which you can download from <http://docs.agile.com>.

After installation, the samples can be found under <agiledocs_home>\integration\sdk\samples.

You can build the MyFirstWebService sample using the sample's build.xml file, located at AGILE_HOME\integration\sdk\samples\wsx\build.xml.

To invoke the sample build on Windows, enter the following command:

```
%ANT_HOME%\bin\ant -Dwsx.url=http://webserver/virtualPath/ws
```

To invoke the sample build on Solaris, enter the following command:

```
$ANT_HOME/bin/ant -Dwsx.url=http://webserver/virtualPath/ws
```

The Ant tool reads the build.xml script and builds all targets in the following sequence:

- 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:
`AGILE_HOME\integration\sdk\samples\wsx\built\src\client`
- 5 Compiles the client classes and places them in the following folder:
`AGILE_HOME\integration\sdk\samples\wsx\built\classes\client`

Once you have deployed the MyFirstWebService sample, you can point your web browser to `http://hostname:port/virtualPath/ws/MyFirstWebService?wsdl` to see the WSDL corresponding to this web service. You can also run the client using the runner script. For more information, see “Running MyFirstClient” on page 21-12.

To build the WSX sample on server machine:

- 1 Install the Agile SDK sample programs.
- 2 Copy the samples folder from: `<agiledocs_home>\integration\sdk` to `<agile_home>\integration\sdk`.
- 3 Build the MyFirstWebService sample using the sample's build.xml file:

```
D:\Agile\Agile921\integration\sdk\samples\wsx>ant -Dwsx.url=http://hostname:port/Agile/ws -
Dusername=admin -Dpassword=agile
```

To build the WSX sample on a client machine:

- 1 Install the Agile SDK sample programs.
- 2 Copy the samples folder from `<agiledocs_home>\integration\sdk` to `<agile_home>\integration\sdk`
- 3 Build the MyFirstWebService sample using the sample's build.xml file:

```
D:\Agile\Agile921\integration\sdk\samples\wsx>ant -Dwsx.url=http://hostname:port/Agile/ws -
Dusername=admin -Dpassword=agile
```

Note The build will fail because MyFirstWebService.jar has not been deployed to server yet.
- 4 Manually copy MyFirstWebService.jar from `<agile_home>\integration\sdk\extensions` on the client machine to `<agile_home>\integration\sdk\extensions` on the server.
- 5 Build the sample again with the following command:

```
D:\Agile\Agile921\integration\sdk\samples\wsx>ant -Dwsx.url=http://hostname:port/Agile/ws -
Dusername=admin -Dpassword=agile
```

About Web Service Clients

This section describes the tools you can use to develop client applications, the languages that can generate and process XML and process 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. See the following website: <http://ws.apache.org/axis/>
- ❑ **Java Web Services Developer Pack (JWSDP)** — Sun's Java implementation of the SOAP protocol. See the following website:
<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. See the following website:
<http://msdn.microsoft.com/net>
- ❑ **SOAP::Lite for Perl** — A Perl implementation of the SOAP protocol. See the following website:
<http://www.soaplite.com/>

Note For a comprehensive list of other SOAP implementations, see the following website:
<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-wsd12java output="${built.dir}/src"
                 url="${wsx.url}/MyFirstWebService?wsdl">
    <mapping namespace="http://www.agile.com/ws/SampleWsx" package="client"/>
  </axis-wsd12java>
</target>
```

The above 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 website 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`, you can see 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

When you build and deploy MyFirstWebService, the `AGILE_HOME\integration\sdk\samples\wsx` directory contains a file that can be used to run the web service client. The file contains all the necessary CLASSPATH initialization for the sample.

- ❑ On Windows, the file is `runner.bat`
- ❑ On Solaris, the file is `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 "Title Block.Description"
-e Agile -h localhost -l 80 -n 1000-02 -p agile -u jeffp
```

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 datatypes, AXIS-based web services should work fine with .Net-based web service clients. For some complex datatypes (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 FAQ

This section answers common questions about web service extensions.

What is web service extensions (WSX)?

WSX is a framework that allows 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.

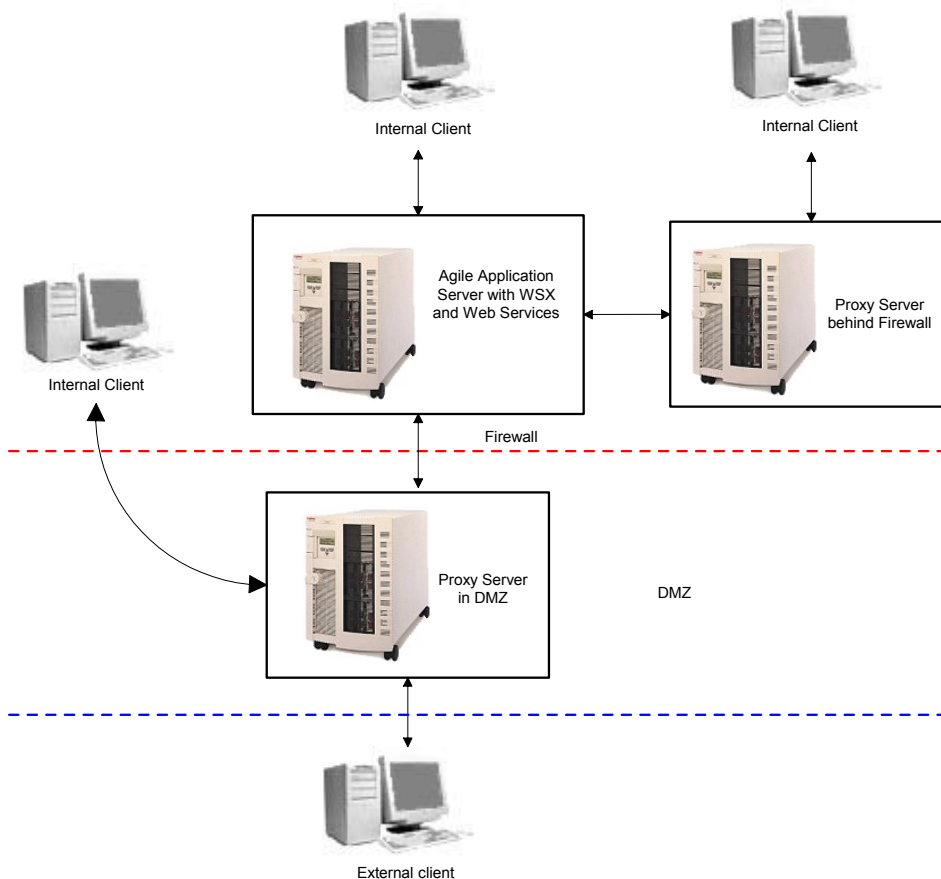
What can I do with WSX that I can't do solely with Agile's Java API?

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 21-2: How web service clients connect to the Agile PLM server



What user authentication services does WSX provide?

Agile provides an `AuthorizationHandler` that can be specified in the deployment descriptor for a web service. Username and password security is enforced whenever a WSX client invokes a service that uses the `AuthorizationHandler`. You can also use Agile's Java API to specify a username and password within a web service that doesn't leverage the `AuthorizationHandler`.

What SOAP engine does WSX use?

WSX is based on Apache Axis, an open-source implementation of SOAP. For more information about Axis, see the Axis web site located 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, see the 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) should 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, see the *Axis Reference Guide* available 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://<host>:<port>/Agile/integration/ws/xxxx`

For example, use this URL for the MyFirstWebService sample:

`http://<hostname>/Agile/integration/ws/MyFirstWebService?wsdl`

PART III

Reference

Part III provides reference information about the Agile SDK. It includes the following chapter:

- ❑ *Chapter 22, Mapping Agile PLM Client Features to the Agile API*
-

The Agile API HTML Reference provides summaries for Agile API classes and interfaces contained in the following packages:

- ❑ *com.agile.api*
- ❑ *com.agile.px*

The Agile API HTML Reference is provided with the software and documentation. For more information on installing the documentation, see “Agile SDK Installation Folders” on page 1-4.

CHAPTER 22

Mapping Agile PLM Client Features to the Agile API

This chapter provides tables that show how features in the Agile Web Client and Java Client map to equivalent Agile API methods. It contains the following topics:

- ❑ *Login Features*
- ❑ *General Features*
- ❑ *Search Features*
- ❑ *Attachment Features*
- ❑ *Workflow Features*
- ❑ *Manufacturing Site Features*
- ❑ *Folder Features*
- ❑ *Program Features*
- ❑ *Administrative Features*

Login Features

Table 22-1 lists general features for logging in to the Agile Application Server.

Table 22-1: Login features

Feature	Equivalent Method(s)
Get an instance of the Agile Application Server session	<code>AgileSessionFactory.getInstance()</code>
Create a session and log in to the Agile Application Server	<code>AgileSessionFactory.createSession()</code>
Close a session and disconnect from the Agile Application Server	<code>IAgileSession.close()</code>

General Features

Table 22-2 lists general features that apply to all Agile PLM business objects.

Table 22-2: General features

Feature	Equivalent Method(s)
Create a new object	<code>IAgileSession.createObject()</code>
Load an existing object	<code>IAgileSession.getObject()</code>
Save an object as another object	<code>IDataObject.saveAs()</code>

Table 22-2: General features (continued)

Feature	Equivalent Method(s)
Delete an object	<code>IDataObject.delete()</code> <code>IFolder.delete()</code> <code>IQuery.delete()</code>
Undelete an object	<code>IDataObject.undelete()</code>
Get a cell value for an object	<code>IDataObject.getValue()</code>
Set an cell value for an object	<code>IDataObject.setValue()</code>
Get a table for an object	<code>IDataObject.getTable()</code>
Add a row to a table	<code>ITable.createRow()</code>
Remove a row from a table	<code>ITable.removeRow()</code>
Get subscriptions for an object.	<code>ISubscribable.getSubscriptions()</code>
Enable a subscription event.	<code>ISubscription.enable()</code>
Modify subscriptions for an object.	<code>ISubscribable.modifySubscriptions()</code>

Search Features

Table 22-3 lists search, or query, features.

Table 22-3: Search features

Feature	Equivalent Method(s)
Set the name of a search	<code>IQuery.setName()</code>
Make the search public or private	<code>IQuery.setQueryType()</code>
Set the search type for a query (object search or Where Used search)	<code>IQuery.setSearchType()</code>
Set and get search criteria	<code>IQuery.setCriteria()</code> <code>IQuery.getCriteria()</code>
Run a search	<code>IQuery.execute()</code>
Make a search case-sensitive	<code>IQuery.setCaseSensitive()</code>
Delete a search	<code>IQuery.delete()</code>
Save a search as another search	<code>IQuery.saveAs()</code>

Attachment Features

Table 22-4 lists features for working with attachments and file folders.

Table 22-4: Attachment features

Feature	Equivalent Method(s)
Download all files contained in a file folder	<code>IFileFolder.getFile()</code>
Download a single file listed on the Attachments tab	<code>IAttachmentFile.getFile()</code>
Check out a file folder	<code>IFileFolder.checkOut()</code>
Check in a file folder	<code>IFileFolder.checkIn()</code>
Cancel checkout	<code>IFileFolder.cancelCheckOut()</code>
Incorporate or unincorporate an item, thereby locking or unlocking its attachments	<code>IAttachmentContainer.setIncorporated()</code>

Workflow Features

Table 22-5 lists workflow features for routable Agile PLM objects.

Table 22-5: Workflow features

Feature	Equivalent Method(s)
Audit a routable object	<code>IRoutable.audit()</code>
Change the status of a routable object	<code>IRoutable.changeStatus()</code>
Send an object to another Agile PLM user(s)	<code>IDataObject.send()</code>
Approve a routable object	<code>IRoutable.approve()</code>
Reject a routable object	<code>IRoutable.reject()</code>
Comment on a routable object	<code>IRoutable.comment()</code>
Add or remove approvers and observers for a routable object	<code>IRoutable.addApprovers()</code> <code>IRoutable.removeApprovers()</code>

Manufacturing Site Features

Table 22-6 lists features for working with manufacturing sites.

Table 22-6: Site features

Feature	Equivalent Method(s)
Get the current manufacturing site selected for an item	<code>IManufacturingSiteSelectable.getManufacturingSite()</code>
Get all manufacturing sites for an item	<code>IManufacturingSiteSelectable.getManufacturingSites()</code>
Set an item to use all manufacturing sites.	<code>IManufacturingSiteSelectable.setManufacturingSite(ManufacturingSiteConstants.ALL_SITES)</code>
Specify that an item is not site-specific and is common to all sites.	<code>IManufacturingSiteSelectable.setManufacturingSite(ManufacturingSiteConstants.COMMON_SITE)</code>
Set an item to use a specific manufacturing site.	<code>IManufacturingSiteSelectable.setManufacturingSite(site)</code>

Folder Features

Table 22-7 lists features for working with folders.

Table 22-7: Folder features

Feature	Equivalent Method(s)
Add an item (such as a query) to the folder	<code>IFolder.addChild()</code>
Set the type of folder (public or private)	<code>IFolder.setFolderType()</code>
Set the folder name	<code>IFolder.setName()</code>
Get a folder of the current user	<code>IUser.getFolder()</code>
Remove an item from the folder	<code>IFolder.removeChild()</code>
Clear all objects from the folder	<code>IFolder.clear()</code>
Delete a folder	<code>IFolder.delete()</code>

Program Features

Table 22-8 lists features for working with programs.

Table 22-8: Program features

Feature	Equivalent Method(s)
Save a program as another program or template	<code>IProgram.saveAs()</code>
Reschedule a program	<code>IProgram.reschedule()</code>
Assign users from a resource pool	<code>IProgram.assignUsersFromPool()</code>
Delegate ownership of a program to another user	<code>IProgram.delegateOwnership()</code>
Substitute program resources	<code>IProgram.substituteResource()</code>
Create a baseline	<code>IProgram.createBaseline()</code>
Select a baseline view of the program	<code>IProgram.selectBaseline()</code>
Lock or unlock a program	<code>IProgram.setLock()</code>
Reply to a discussion	<code>IMessage.reply()</code>

Administrative Features

Table 22-9 lists features for working with Admin nodes and properties in the Agile Java Client.

Table 22-9: Administrative features

Feature	Equivalent Method(s)
Get an administrative node	<code>IAdmin.getNode()</code>
Get all subnodes (children) of an administrative node	<code>ITreeNode.getChildNodes()</code>
Get all properties of an administrative node	<code>INode.getProperties()</code>
Get the value for an administrative node's property	<code>IProperty.getValue()</code>
Get the possible values for a list field	<code>IProperty.getAvailableValues()</code>
Get all Agile PLM classes	<code>IAdmin.getAgileClasses(ALL)</code>
Get all top-level Agile PLM classes	<code>IAdmin.getAgileClasses(TOP)</code>
Get all Agile PLM classes that can be instantiated	<code>IAdmin.getAgileClasses(CONCRETE)</code>
Get the list of subclasses for a specific class	<code>IAgileClass.getSubclasses()</code>
Get the Autonumber sources for a subclass	<code>IAgileClass.getAutoNumberSources()</code>
Get an array of attributes for a table	<code>IAgileClass.getTableAttributes()</code>
Get the metadata for a table	<code>IAgileClass.getTableDescriptor()</code>
Get the Agile PLM list library	<code>IAdmin.getListLibrary()</code>
Create a new Agile PLM list	<code>IListLibrary.createAdminList()</code>
Get an Agile PLM list	<code>IListLibrary.getAdminList()</code>
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	<code>IAgileSession.createObject()</code>
Set properties of a user or user group	<code>IProperty.setValue()</code>
Change user passwords	<code>IUser.changeApprovalPassword()</code> <code>IUser.changeLoginPassword()</code>

INDEX

Symbols

\$TODAY 3-5

\$USER 3-5

A

abstract classes 18-8

action items 14-15

addAll() method 7-5

addApprovers() method 11-4, 11-5

addChild() method 3-2, 6-4, 8-1, 8-9, 8-10, 8-11, 8-12, 8-13

adding table rows 4-11

administration 18-1

 getting an IAdmin instance 18-2

 interfaces 18-2

 node properties 18-13

 objects 18-2

 privileges 18-2

aggregate interfaces 2-1

Agile business objects 1-4

Agile Integration Service (AIS) 2-6

Agile product licenses 18-1

AgileAPI.jar 1-4, 2-4

AgileSDK.jar

 path to 2-4

all or any operator modifiers 3-7

Ant

 installing 21-8

Apache Extensible Interaction System
 (Axis) 21-4

approve() method 11-5

approvers 11-4

approving a change 11-5

assignUsersFromPool() method 14-10

attachments 10-1

 adding 10-7

 cloning 10-8

 deleting 10-13

 getting a file 10-12

 retrieving 10-4

 unzipping a file stream 10-13

attributes 18-10–18-12

 about 3-5

 fully qualified 18-10

 IDs 18-10

 master-child 16-5

 names 18-10

 read-through 16-5

 referencing 18-10

 retrieving 18-11

 tabs and tables 18-6

 user defined 18-12

audit() method 11-6

auditing a change 11-6

AutoNumbers 2-12

 custom autonumber sources 20-2

Available for Subscription property 16-4

Axis 21-4

axis.jar 1-4

aXML 1-2, 21-2

B

baselines 14-7

basic authentication 21-15

between operator 3-7

Bill of Substances, structure 15-8

BOMs 7-4

 copying 7-5

 expanding 7-5

 redlining 7-6

build.xml file 21-8

C

cancelCheckout() method 10-6

canceled checkout 10-6

cascading lists

 creating 8-3, 8-11

 setting values 5-7

cells 5-1

 data types 5-1

 Discovery privilege 5-2

 getting values 5-3

 list values 5-5

 read-only 5-2

 setting values 5-4

change control process 11-1

changeStatus() method 11-2, 11-8, 15-15, 15-16

changing the status of a change 11-7

checkIn() method 10-5, 10-10

checking in a file folder 10-10

checking out a file folder 10-6

checkout() method 10-6

checkoutEx() method 10-5

class path 2-4

classes 18-6

 abstract 18-8

 concrete 18-8

 IDs 18-9

 names 18-9

 referencing 18-9

 subclasses 18-6

 target type 18-9

Classes node 18-6

CLASSPATH 21-12

clear() method 4-18, 6-4, 16-6

clearing a table 4-18

client-side stubs 21-10, 21-11

cloning attachments 10-8

close() method 2-21

closing a session 2-21

comment() method 11-6

commenting a change 11-6

Compliance Manager role 15-2

concrete classes 18-8

Concurrent user 18-16

constants 2-2

cookie, j_password 20-10

cookie, j_username 20-10

createBaseline() method 14-7

createObject() method 2-10, 2-11, 2-15, 3-1, 3-2, 6-3, 9-2, 10-2, 12-3, 12-5, 13-2, 13-4, 13-6, 18-9, 18-14, 18-15, 18-18

createRow() method 4-12, 4-13, 4-14, 7-5, 9-3, 10-2, 10-7, 12-5, 13-5, 18-18

createSession() method 2-5, 20-12

createSessionEx() method 2-6

createSortBy() method 4-16

createUOM() method 15-16

creating objects 2-10

creating search filters 3-4

criteria 3-4

 all or any operator modifiers 3-7

 dates 3-9

 nested 3-8

custom actions 20-5

customer flex fields 18-12

customers 13-2

D

dates, in queries 3-9

declarations

 adding items, manufacturer parts,
 and part groups 15-6

 adding specifications 15-12

 creating 15-3

 filling out 15-15

 publishing 15-16

 routing 15-13

- submitting to Compliance Manager 15-15
- workflow statuses 15-14
- delays in propagating values 2-18
- delegateOwnership() method 14-8
- delegating programs 14-8
- delete() method 2-20, 3-20, 6-6, 8-15, 18-17
- deleting an attachment 10-13
- deleting objects 2-19
- deleting queries 3-20
- deployment descriptors 21-5
- disableWarning() method 2-20, 19-3
- Discovery privilege 5-2
- discussions 14-11
 - action items 14-15
 - creating 14-12
 - joining 14-15
 - replying to 14-13
- distributing applications 2-3
- doAction() method 20-17
- document text, searching 3-8
- dummy subparts and materials 15-9
- dynamic lists 8-6

E

- enable() method 8-15, 16-3
- enableWarning() method 2-20, 19-3
- enumerated list values 8-6
- error handling 19-1
 - disabling and enabling warning messages 19-2
 - error codes 19-2
 - error messages 19-2
 - saving and restoring the state of warning messages 19-4
- events, subscription 16-1
- exceptions 19-1
 - disabling and enabling warning messages 19-2
 - getting error codes 19-2
 - getting error messages 19-2
- execute() method 3-1
- expired passwords 18-16
- extensions folder 20-3, 20-6
- external reports 20-9, 20-14

F

- factory class 2-2
- FALCPA 15-4
- favorites *See* My Bookmarks folder
- fields *See* cells
- file content, searching 3-8
- file folders 10-2
 - canceling checkout 10-6
 - checking in 10-10
 - checking out 10-6

- file versions 10-10
- file versions 10-10
- folders 6-1
 - adding and removing items 6-3
 - creating 6-3
 - getting children 6-4
 - iterating over elements 6-5
 - loading 6-2
 - private 6-3
 - public 6-3
 - setting type 6-3
- fully qualified attributes 18-10
- functional unit interfaces 2-1

G

- gates 14-3
- getAdminInstance() method 2-11, 18-2
- getAdminList() method 8-8, 8-10, 8-13
- getAgileClass() method 2-11, 3-2
- getAgileClasses() method 2-12, 18-9
- getApprovers() method 11-4
- getApproversEx() method 11-8
- getAttachments() method 10-4
- getAttribute() method 16-2, 18-12
- getAttributes() method 4-11, 18-11
- getAutoNumberSources() method 2-13
- getAvailableValues() method 5-6, 5-7, 5-8, 8-4, 8-7, 13-4, 13-6, 13-8, 16-4, 18-17, 18-19
- getBaselines() method 14-7
- getCell() method 5-6, 8-7, 12-6, 12-7, 18-17, 18-19
- getCells() method 18-16
- getChild() method 3-19, 5-8, 6-2, 6-4, 8-13
- getChildNode() method 18-6
- getChildNodes() method 6-5, 8-6, 18-5
- getCurrentUser() method 11-5, 13-4, 13-6
- getDefaultNextStatus() method 11-2, 11-7, 11-8
- getErrorCode() method 19-2, 19-3
- getFile() method 10-4, 10-5, 10-12
- getID method() 8-5
- getID() method 2-12
- getId() method 8-5
- getInstance() method 2-5, 20-12
- getListDataType() method 8-12
- getListLibrary method 8-8
- getListLibrary() method 8-1, 8-9, 8-11, 8-13
- getManager() method 15-16
- getManufacturingSite() method 9-3
- getManufacturingSites() method 9-4
- getMessage() method 19-2

- getName() method 2-12, 11-7, 17-2
- getNextNumber() method 2-13
- getNextStatuses() method 11-7
- getNode() method 18-5, 18-6
- getObject() method 2-7, 2-8, 3-19, 6-2, 9-2, 12-3, 12-5, 12-7, 12-8, 12-9, 13-2, 18-8, 20-13
- getObservers() method 11-4
- getObserversEx() method 11-8
- getPriority() method 17-2
- getProperties() method 18-6
- getProperty() method 2-13, 8-8, 16-4, 18-13
- getReferent() method 4-21, 10-2, 12-8, 12-9
- getReferentIterator() method 3-17, 4-21, 16-6
- getResultAttributes() method 3-17
- getRevision() method 7-2, 7-3
- getRevisions() method 7-2
- getSelection() method 8-5
- getStatus() method 11-2, 11-7
- getStatusType() method 11-2
- getStub() method 21-11
- getSubclasses() method 2-11
- getSubscriptions() method 16-2, 16-3, 16-6
- getTable() method 4-11, 4-22, 9-2, 12-6, 12-8, 12-9, 13-5, 16-5, 16-6, 18-18, 18-19
- getTableAttributes() method 16-4, 18-11
- getTableDescriptor() method 4-11
- getTableIterator() method 3-17, 4-16, 12-6, 12-8, 12-9
- getTargetType() method 18-9
- getting values 5-3
- getUnit() method 15-16
- getValue() method 5-3, 5-4, 5-6, 5-7, 7-3, 8-5, 18-13
- getValues() method 8-9, 8-10, 8-11, 8-12, 8-13
- getWorkflow() method 11-3, 13-8
- getWorkflows() method 11-3, 13-8

H

- hasDiscoveryPrivilege() method 5-2
- hasNext() method 4-16, 6-5
- hasPrevious() method 4-16
- hasPrivilege() method 11-2
- Home folder 6-1, 6-2
- HTML reference 1-4
- HTTPS 2-5

I

- IAdmin 18-2
- ICheckoutable interface 10-5

IMeasure interface 15-16
 importing the Agile API package 2-4
 in operator 3-7
 incorporated status 7-3
 installing
 Ant 21-8
 Java SDK 21-7
 interrupt() method 17-2
 intersect operator 3-18
 Introductory revision 7-2
 isAlive() method 17-2
 isCheckedOut() method 10-5
 isDeleted() method 2-20
 isEnabled() method 8-8, 16-2
 isFlagSet() method 4-21
 IShareable interface 2-19
 isIncorporated() method 7-3
 isObsolete() method 8-14
 isReadOnly() method 5-2, 8-15
 isSecure() method 10-4
 issues 13-3
 isWarning() method 19-3
 items
 getting and setting the revision 7-1
 iterator() method 3-17, 4-22, 6-5
 iterator, folder 6-5
 ITwoWayIterator 4-16
 IUnit interface 15-16
 IUnitOfMeasure interface 15-16
 IUnitOfMeasureManager interface 15-16

J

j_password cookie 20-10
 j_username cookie 20-10
 Jakarta Commons CLI 21-7
 Java SDK, installing 21-7
 Java Web Services Developer Pack 21-10
 Javadocs 1-4
 join() method 17-2
 joining a discussion 14-15

K

keywords, query language 3-5

L

LDAP server 18-14
 left operand 3-4
 licenses 18-1
 lists
 adding values 8-13
 cascading 8-11
 custom 8-9
 data type 8-12
 datatypes 8-1
 dynamic 8-6

getting and setting values 5-5
 modifying 8-13
 obsolete values 8-14
 printing 8-15
 selecting a value 8-4
 static 8-6
 types of 8-1
 loading folders 6-2
 locked objects 5-5
 locking a program 14-11
 logging in 2-4
 logging out 2-21
 logical operators 3-9

M

MAKE_DEEP_COPY constant 10-8
 manufacturer part, adding substance
 compositions 15-17
 manufacturing sites
 controlling access 9-1
 creating 9-2
 disabling 9-5
 getting and setting the current site 9-3
 loading 9-2
 overview 9-1
 retrieving the Sites table 9-2
 updating the Sites table 9-3
 (Restricted) Material Provider role 15-2
 materials 15-5
 Maximum Query Results Displayed
 preference 3-18
 metadata interfaces 2-1, 4-11
 META-INF folder 20-3, 20-6
 minus operator 3-18
 modifySubscriptions() 16-3
 MultiList attributes 8-3
 MultiList datatype 8-1
 multithreading 2-3
 My Bookmarks folder 6-1

N

nested criteria 3-8
 .NET interoperability 21-12
 Web Service Extensions
 .NET interoperability 21-12
 network class loading 2-2
 new features 1-3
 next() method 4-16, 4-22, 6-5, 12-6, 12-8, 12-9
 No Privilege value 5-2
 nodes 18-3
 retrieving 18-5
 unsupported 18-5
 notifications, subscription 16-2

O

observers 11-4
 obsolete list values 8-14
 operators
 logical 3-9
 relational 3-4, 3-6
 where-used set 3-18

P

Page One, Page Two, and Page Three 4-22
 parameters, in queries 3-2
 partial result queries 3-18
 password-protected URLs 2-5
 passwords 18-14, 18-16
 PDX 21-2
 Personal Searches folder 6-1, 6-2
 phases 14-3
 popWarningState() method 19-4
 Power User 18-16
 previous() method 4-16
 prices 12-2
 creating 12-2
 loading 12-3
 price change order (PCO) 12-5
 price lines 12-3
 private folders 6-1, 6-3
 privileges
 Subscribe 16-2
 privileges, administration 18-2
 Process Extensions
 cookies 20-10, 20-11
 creating an Agile session 20-11
 custom actions 20-5
 custom autonumber sources 20-2
 FAQ 20-15
 overview 20-1
 packaging and deploying 20-3, 20-6
 retrieving an Agile object 20-12
 single signon 20-11
 timeout 20-16
 URL encoding 20-10
 URL-based 20-9
 Product Governance & Compliance 15-1-??
 product sourcing 12-1-12-9
 pricing 12-2
 projects 12-6
 RFQs 12-8
 suppliers 12-5
 programs
 action items 14-15
 assigning resources 14-8
 assigning users 14-10
 baselines 14-7

- delegating ownership 14-8
- discussions 14-11
- hierarchy 14-3
- locking 14-11
- phases, tasks, and gates 14-3
- substituting resources 14-10
- projects *See* sourcing projects
- properties 18-13
 - modifying 18-12
- public folders 6-1, 6-3
- pushWarningState() method 19-4
- put() method 8-9, 8-11
- pxapi.jar 1-4

Q

- quality 13-1–13-8
 - objects 13-1
 - roles and privileges 13-2
- queries 3-1
 - attributes 3-5
 - creating 3-20
 - criteria 3-4
 - deleting 3-20
 - file content 3-8
 - formatting dates 3-9
 - keywords 3-5
 - loading 3-19
 - maximum number of results 3-18
 - nested criteria 3-8
 - parameterized 3-2
 - performance 3-18
 - result attributes 3-12
 - results 3-17
 - saving 3-2
 - searchable attributes 3-6
 - sorting query results 3-12
 - SQL syntax 3-10
 - temporary 3-2
 - where-used 3-18
 - wildcards 3-10

R

- read-only cells 5-2
- read-only tables 4-11
- read-through attributes 16-5
- Recently Visited folder 6-1
- redlining 4-22
 - undoing redlines 4-24
- redlining BOMs 7-6
- reject() method 11-6
- rejecting a change 11-5
- related objects, propagating values 2-18
- relational operators 3-4, 3-6
 - between and in 3-7
- removeApprovers() method 11-4
- removeBaseline() method 14-7
- removeChild() method 6-4

- removeRow() method 4-17, 16-5
- reply() method 14-14
- replying to discussions 14-13
- report folders 6-2
- request for quotes (RFQ) 12-8
- required fields 2-14
- reschedule() method 14-6
- resources, substituting 14-10
- result attributes 3-12
- results, query 3-17
- retrieving objects 2-7
- revisions 7-1
 - incorporated status 7-3
- RFQ Responder 18-16
- RFQ responses 12-8
 - loading 12-9
- right operand 3-4
- RoHS Directive 15-4
- roles 18-16
 - Product Governance & Compliance 15-2
- rows
 - adding 4-11
 - checking status flags 4-21
 - iterating 4-15
 - referenced objects 4-18
 - removing 4-17
- run() method 17-2
- runner.bat 21-12
- runner.sh 21-12

S

- sample programs 1-4, 2-3
- saveAs() method 2-18, 3-2, 13-3, 13-7, 14-4, 18-15
- search filters, creating 3-4
- searchable attributes 3-6
- searching *See* queries
- selectBaseline() method 14-7
- send() method 11-8
- sending an object to users 11-8, 11-9
- sessions
 - closing 2-21
 - creating 2-4
- setAvailableValues() method 8-6, 8-10
- setCaseSensitive() method 3-1
- setCriteria() method 3-1, 3-9, 3-18
- setCurrentVersion() method 10-7, 10-10
- setDateFormats() method 3-9
- setDescription() method 8-14
- setFile() method 10-3, 10-12
- setFolderType() method 6-3
- setIncorporated() method 7-3
- setLevelName() method 8-11, 8-12, 8-14

- setLock() method 14-11
- setManufacturingSite() method 7-4, 7-8, 9-4
- setName() method 3-2, 8-14, 17-2
- setObsolete() method 8-14
- setParams() method 3-3
- setPriority() method 17-2
- setResultAttributes() method 3-16
- setRevision() method 7-2, 7-3
- setSearchType() method 3-18
- setSelection() method 5-6, 5-7, 5-8, 8-4, 8-7, 9-5, 13-4, 13-6, 13-8, 16-4, 18-17, 18-19
- setting values 5-4
- setUsersAndRoles() method 2-19
- setValue() method 5-4, 5-5, 5-6, 5-7, 5-8, 8-4, 8-7, 8-8, 9-5, 12-6, 12-7, 13-4, 13-6, 13-8, 16-4, 18-17, 18-19
- setValues() method 5-5, 8-9, 8-11, 8-12, 8-13
- setWorkflow() method 11-3, 13-8
- setWorkflows() method 11-3
- sharing objects 2-19
- Simple Object Access Protocol (SOAP) 21-2, 21-3
- SingleList attributes 8-2
- SingleList datatype 8-1
- sites *See* manufacturing sites
- Sites table 9-2, 9-3
- sleep() method 17-2
- SOAP Lite for Perl 21-10
- SOAP request
 - generating 21-10
 - submitting 21-10, 21-11
- SOAP response
 - parsing 21-11
 - processing 21-11
- SOAP response, processing 21-10
- sorting query results 3-12
- sorting tables 4-16
- sourcing projects 12-6
 - cover page values 12-7
 - loading 12-7
- specifications
 - adding substances 15-12
 - adding to a declaration 15-12
 - creating 15-4
- SQL syntax of queries 3-10
- wildcards 3-12
- start() method 17-2
- static lists 8-6
- status flags 4-21
- status, workflow 11-2
- subclasses 2-11, 18-6
 - user-defined 2-12

- subparts 15-5
- Subscribe privilege 16-2
- Subscription table 16-5
 - clearing 16-6
- subscriptions
 - configuring attributes 16-4
 - deleting subscribed objects 16-2
 - events 16-1
 - getting 16-2
 - modifying 16-3
 - notifications 16-2
 - overview 16-1
 - removing 16-5
- substance groups 15-5
- substances
 - adding to item, manufacturer part, or commodity 15-7
 - creating 15-4
- substituteResource() method 14-10
- supplier responses *See* RFQ responses
- suppliers 12-5
 - loading 12-5
 - modifying 12-6
- system folders 6-1

T

- tables 4-2
 - adding a row 4-11
 - clearing 4-18
 - iterating rows 4-15
 - metadata 4-11
 - read-only 4-11
 - redlining 4-22
 - removing a row 4-17
 - removing redlines 4-24
 - retrieving 4-2
 - retrieving referenced objects 4-18
 - sorting 4-16
- target type 18-9
- tasks 14-3
- temporary queries 2-3
- thread synchronization 2-3
- timeout 20-16
- toString() method 5-6, 5-7
- transactions 5-5
- type, folder 6-3

U

- unassigned changes 11-3
- undelete() method 2-20
- undeleting objects 2-19
- undoRedline() method 4-24
- union operator 3-18
- Universal Description, Discovery, and Integration (UDDI) 21-4
- unzipping file streams 10-13
- UOM node 15-16

- URL encoding 20-10
- URL-based Process Extensions 20-9
- user groups 18-17
 - creating 18-18
 - listing users 18-19
 - retrieving 18-18
- user-defined attributes 18-12
- user-defined subclasses 2-12
- users 18-13
 - creating 18-14
 - deleting 18-17
 - expired passwords 18-16
 - modifying properties 18-16
 - retrieving 18-14
 - Roles field 18-16
 - saving to a new user 18-15
 - User Category field 18-16

W

- warning messages
 - disabling and enabling 19-2
 - saving and restoring the state of 19-4
- Web Service Deployment Descriptor (WSDD) 21-5
- Web Service Description Language (WSDL) 21-2, 21-4
- Web Service Extensions
 - architecture 21-2
 - authentication 21-6
 - client programming languages 21-9
 - clients 21-9
 - deployment descriptors 21-5
 - developing 21-5
 - entry points 21-6
 - FAQ 21-12
 - features 21-2
 - overview 21-1
 - security 21-4
 - serializers and deserializers 21-6
- Web Services
 - architecture 21-3
 - key benefits 21-3
 - protocol stack 21-4
- WEEE Directive 15-4
- weight values 15-16
 - searching for 15-17
- where-used queries 3-18
- wildcards 3-10, 3-12
- workflow 11-1
 - adding and removing approvers 11-4
 - approving or rejecting a change 11-5
 - auditing a change 11-6

- change control process 11-1
- changing the status of a change 11-7
- commenting a change 11-6
- sending an object to users 11-8, 11-9
- status 11-2
- wsdl.exe utility 21-10

X

- XSL 21-2

Z

- ZipInputStream class 10-13
- zipped file streams 10-13

