Agile Product Lifecycle Management

SDK Developer Guide - Using APIs

Release 9.3.6

E71152-01

January 2017

Agile Product Lifecycle Management, Release 9.3.6

E71152-01

Primary Author: Oracle Corporation

Contributing Author: F. Tabibzade

Contributor:

Contents

23 Performing Administrative Tasks

A Mapping PLM Client Features to Agile API

Preface

Agile PLM is a comprehensive enterprise PLM solution for managing your product value chain.

Audience

This document is intended for administrators and users of the Agile PLM products.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction

This chapter includes the following:

- About this Guide
- Agile PLM Extension Framework
- SDK Components
- SDK Architecture
- System Requirements
- Java Requirements
- Agile SDK Installation Folders
- Checking Your Agile PLM System
- Agile PLM Business Objects

About this Guide

Oracle's Agile Software Development Kit (SDK) is a collection of Java application programming interfaces (APIs), sample applications, and documentation that enable building custom applications to access, or extend the functionalities of the Agile Application Server. Using the SDK, you can create programs that extend the functionality of the Agile product lifecycle management system (PLM) and can perform tasks against the PLM system.

The SDK enables the following operations:

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

The SDK Developer Guide is published in the following two books:

- SDK Developer Guide - Using Agile APIs - This component of the SDK Developer Guide provides information to develop batch operations against the PLM Server, integrate the PLM with other application, and process PLM data. This information is described and documented in this book.
- SDK Developer Guide - Developing Extensions - This component of the SDK Developer Guide provides background and procedural information to create additional PLM clients (extend Agile PLM functionalities) and work with PLM

Frameworks. This information is described and documented in the SDK Developer Guide - Developing PLM Extensions.

Agile PLM Extensions Framework

This component of the SDK Developer Guide provides referential and procedural information to get you started with the APIs and use the APIs to develop applications that programmatically perform batch operations against the Agile Application Server to execute tasks such as:

- Querying PLM databases
- Loading data into PLM databases
- Working with Tables, Data Cells, and Folders
- Importing and exporting data to and from Agile PLM
- Processing product data
- Interacting with PLM modules such as Product Cost Management, Product Portfolio Management, and other modules
- Managing workflow
- Managing and tracking quality
- Managing Reference objects
- Performing administrative functions

SDK Components

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

Client-Side Components

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

Documentation

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

Note: The API HTML reference files and Sample applications are in the `SDK_samples.zip` folder. You can find this folder at `http://www.oracle.com/technetwork/indexes/samplecode/agileplm-sample-520945.html`. For more information and procedures to access its contents, contact your system administrator, or refer to your PLM installation guide.

Installation

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

Server-Side Components

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

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

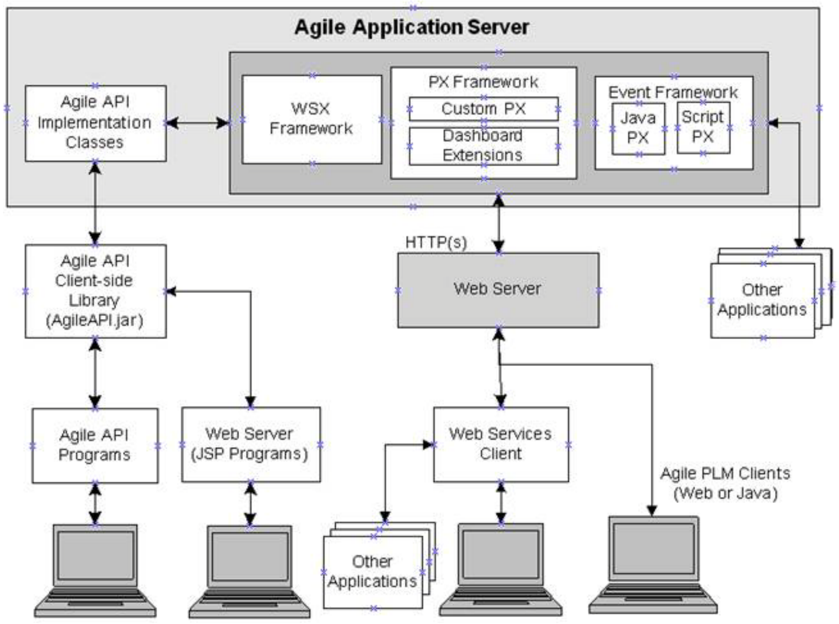
SDK Architecture

The SDK facilitates developing different types of programs to connect to the Agile Application Server. If you are using only the Agile APIs, these programs connect directly to the server. For information to develop these types of programs, refer to *SDK Developer Guide - Using Agile APIs*.

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 is accessible from inside or outside the company's demilitarized computing zone (DMZ) or perimeter network. Information for developing Web service extensions is provided in this document.

When the Agile PLM Client initiates a custom action, it either runs a program that is deployed on the server, or connects to an external resource such as a URL. WSX, Java PX and Script PX extensions can also use the Agile APIs. You can develop extensions using APIs that are not provided by Agile. This information is also provided in this document.

Figure 1–1 Agile SDK architecture



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

System Requirements

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

Java Requirements

The Agile API must be compatible with the version of Java that the application server supports. To avoid problems, an Agile API Client must use the same version of Java that the connecting Oracle WebLogic Server 12.13 running the Sun Java Runtime Environment (JRE) 1.8 for interoperability and 2007 Daylight Saving Time compliance.

Important: SDK code running under JRE 7 cannot connect to a Proxy URL protected by SSO. To establish this connection, you must directly connect your SDK code to server nodes with actual Weblogic ports, or setup a second proxy that is not protected by SSO.

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

Table 1–1 Java version requirements

Application Server	**Required Java Version**
Fusion Middleware Infrastructure 12.1.3 (includes WebLogic Server 12.1.3)	Oracle Java JDK1.8.0 update 66 (64-bit only) on the server.

JVM Parameters for Preventing Out of Memory Exceptions

To prevent out of memory errors, add the following Java Virtual Memory (JVM) parameter options in the indicated locations.

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

- If the Client is a standalone SDK Client, add the JVM option:
  `java -Ddisable.agile.sessionID.generation=true pk.sample`

- If the Client is a PX and out of memory occurs in Agile Server, add the JVM option in `<OAS_HOME>/opmn/conf/opmn.xml`.

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

- If the Client is a URL PX, add the following JVM option in the Server Start up (This is similar to `catalina.bat in Tomcat`.)
  `-Ddisable.agile.sessionID.generation=true`

Note: For more information about URL Process Extensions, or URL PXs, including how to set the Cookie Expiration Properties for URL PXs, refer to *SDK Developer Guide - Developing PLM Extensions*.

Agile SDK Installation Folders

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

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

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

- `AgileAPI.jar` - Agile API library. This file contains Agile API classes and interfaces.
- `pxapi.jar` - PX API library.This file contains interfaces used to develop custom autonumber sources and custom actions.

Checking Your Agile PLM System

Before trying to run the Agile SDK Clients on your Agile PLM system, make sure the system is configured and is working properly. In particular, make sure the HTTP ports for your application server are set correctly. To this end, type the following URL (the test URL) in your browser, and

For example, if the server is running in a Windows environment, to start and test the Agile PLM Application Server connection do as follows:

To start and test the Agile PLM Application Server connection do as follows:

1. From Windows command line, select **Start > All Programs > Agile > Agile PLM > Start Agile Server**.

   A command window may appear and this window must remain open but you can minimize it.

2. Wait until the following message "Agile PLM Server Starting Up" appears in the command window or application server log file before connecting:

3. Open your browser and use the following URL to test the Agile Web client setup:

   ```
   http://application_server_hostname:port/virtual_path/PLMServlet
   ```

   Note: The URL is case-sensitive.

4. When the login window appears, type the username and password.

   You can log in with the built-in Administrator account by typing **admin** for the user and the password you supplied as the password for the admin user in the password management screen during installation.

   The first time you log in to the application, it may take a while to load the information. For more information, refer to the *Agile PLM Installation Guide*.

Agile PLM Business Objects

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

Object	Related Agile API Interface
Changes	`IChange`
Customers	`ICustomer`
Declarations	`IDeclaration`
Design	`IDesign`
Discussions	`IDiscussion`
File Folders	`IFileFolder`
Items	`IItem`
Manufacturer parts	`IManufacturerPart`
Manufacturers	`IManufacturer`
Packages	`IPackage`

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

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

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

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

2 Getting Started with Agile API

This chapter includes the following:

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

Types of Agile API Classes and Interfaces

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

- **Aggregate interfaces** - These interfaces aggregate the applicable functional interfaces for a particular object type. For example, the `IItem` interface extends `IDataObject, IRevisioned, IManufacturingSiteSelectable, IAttachmentContainer, IHistoryManager`, and `IReferenced`. Most SDK functionalities fall within these interfaces. The Agile API's underlying implementation classes, which are not exposed, implement these interfaces.
- **Functional Unit Interfaces** - These interfaces hold units of functionality that are extended to other interfaces. For example, `IAttachmentContainer` provides a convenient way to access the attachments table for any object. Other interfaces in this group such as `IChange and IItem` extend the `IAttachmentContainer` interface. `IRoutable` is another class that serves as a functional unit; it provides methods for any object that you can route to another Agile PLM user; `IChange`, `IPackage`, and `ITransferOrder` all extend `IRoutable`.
- **Metadata interfaces** - This group of classes defines the metadata (and meta-metadata) for the Agile Application Server. Metadata is simply data that describes other data. The metadata interfaces include classes such as `IAgileClass, INode, IRoutableDesc, ITableDesc`, and `IWorkflow`.
- **Factory classes** - `AgileSessionFactory` is a factory class that is used to create a session (`IAgileSession`) and access transaction management. `IAgileSession` is also a factory object allowing you to instantiate other objects. Many Agile API objects, in turn, are factory objects for tables or other referenced objects. Tables, in turn, are factories for rows.
- **Exception classes** - There's only one Exception class, `APIException`.

- **Constants** - These classes contain IDs for attributes, tables, classes, and so on. All classes containing only constants have class names that end with "Constants," for example, `ChangeConstants, ItemConstants, UserConstants`, and so on.

Network Class Loading

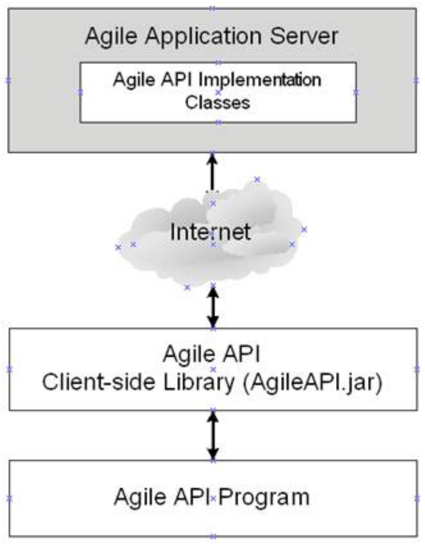
The Agile API has two main software components:

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

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

The Agile API Client-side library is composed almost entirely of interfaces, which are essentially class loaders. 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


Agile Application Server
Agile API Implementation Classes
Internet
Agile API
Client-side Library (AgileAPI.jar)
Agile API Program

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

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

Single-Threaded versus Multi-Threaded Applications

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

Packaging an Agile API Program

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

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

Distributing Agile API Files

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

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

Sample Programs

The Agile SDK provides several sample programs that demonstrate how to use its APIs. These sample programs are in the `api`, `dx`, `px`, and `wsx` folders. You can find them in the `SDK_samples` (ZIP file). To access this file, see the Note in "Client-Side Components" on page 1-2.

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

Starting an Agile API Program

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

To structure programs developed with Agile APIs:

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

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

2. Get an instance of the Agile Application Server.
3. Create an Agile session.

4. Complete one or more business processes. This is where most of your program code goes.
5. Close the Agile session.

Setting the Class Path for the Agile API Library

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

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

Importing Agile API Classes

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

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

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

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

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

Creating a Session and Logging In

Use the JVM parameter called `disable.agile.sessionID.generation=true` to create a session when the login user is a LDAP user. This is applicable only when Agile runs on Weblogic server. If the Client is a standalone SDK Client, add the JVM option:

```
java -Ddisable.agile.sessionID.generation=true pk.sample.
```

Alternatively, you can set the parameter in the code as follows:

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

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

1. **Get an instance of the Agile Application Server.**

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

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

where

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

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

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

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

Example 2–1 Creating a session and logging in

```
public static final String     USERNAME = "fjones";
public static final String     PASSWORD = "agilepwd";
public static final String     URL      = "http://yourcompany.com:7001/Agile";
public static IAgileSession    session  = null;
public static AgileSessionFactory factory;

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

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

Creating a Session by Accessing a Password Protected URL

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

Example 2–2 Creating a session by accessing a password-controlled URL

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

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

Creating a Session from an Agile Web Service

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

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

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

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

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

Creating a Session in a Cluster Environment

The `AgileSessionFactory.getInstance()` and `AgileSessionFactory.createSession()`that you use to create an instance of the AgileSessionFactory, cache the Agile server properties to get the instance. Because of this caching, the `getInstance()` method retrieves the same instance of `AgileSessionFactory` anytime it is invoked.

While retrieving the same instance of AgileSessionFactory is not an issue in single server environments, it can be problematic in Agile cluster environments when the cached server is down. This is due to the following facts:

1. When AgileSessionFactory is initiated with a proxy URL, a specific server in the cluster is cached and is used to create the session.
2. When the cached server is down, `AgileSessionFactory.createSession()` will try to establish a connection with the server, but will fail because the server is down.

To overcome this issue, the Agile SDK exposes the following APIs to refresh the `AgileSessionFactory` instance for Agile cluster environments. These new APIs clear the cached server details and create a new instance of the `AgileSessionFactory`.

- `AgileSessionFactory.refreshInstance(String url)`
- `AgileSessionFactory.refreshInstanceEx(Map params)`
- `AgileSessionFactory.refreshSessionEx(Map params)`

The following examples use these APIs to create Agile sessions in cluster environments

Example 2–3 Creating a session with public static AgileSessionFactory refreshInstance(String url)

```
AgileSessionFactory factory = AgileSessionFactory.refreshInstance(URL);
HashMap params = new HashMap();
params.put(AgileSessionFactory.USERNAME, USERNAME);
params.put(AgileSessionFactory.PASSWORD, PASSWORD);
IAgileSession lsession = factory.createSession(params);
```

Example 2–4 Creating a session with public static AgileSessionFactory refreshInstanceEx (Map map)

```
HashMap params = new HashMap();
params.put(AgileSessionFactory.URL, URL);
params.put(AgileSessionFactory.USERNAME, USERNAME);
params.put(AgileSessionFactory.PASSWORD, PASSWORD);
AgileSessionFactory factory = AgileSessionFactory.refreshInstanceEx(params);
IAgileSession lsession = factory.createSession(params);
```

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

IChange	IManufacturer	IRequestForQuote
`IItem`	`IQuery`	`IUserGroup`

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

For more information about creating queries and folders, see Chapter 3, "Creating and Loading Queries" and 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)`

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

Specifying Object Types

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

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

Note: When you use the `getObject`() method to load an object, you can specify abstract or concrete Agile PLM classes. For more information, see "Concrete and Abstract Classes" on page 23-9.

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, refer to "Identifying Attributes for Agile PLM Classes" in *SDK Developer Guide - Developing PLM Extensions*.

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

Example 2–5 Loading a part using a String

```
IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART, "1000-02");
```

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

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

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

Example 2–6 Loading a part using a Map

```
Map params = new HashMap();
 params.put(ItemConstants.ATT_TITLE_BLOCK_NUMBER,   "1000-02");
     IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART, params);;
```

Loading Different Types of Objects

The following examples load different types of Agile PLM objects.

Example 2–7 Loading Agile PLM Item, Manufacturer, and Manufacturer Parts objects

```
///Load an item
IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-02");
System.out.println("Item : " + item.getName());

//Load a manufacturer
IManufacturer mfr = (IManufacturer)m_session.getObject(IManufacturer.OBJECT_TYPE);

System.out.println("Manufacturer : " + mfr.getName());

//Load a manufacturer part
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());
```

Example 2–8 Loading Agile PLM Customer and Declaration objects

```
//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());
```

Example 2–9 Loading Agile PLM Discussion, File Folder and Folder objects

```
//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());
```

Example 2–10 Loading Agile PLM Package, Price, Program, PSR, and QCR objects

```
//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());
```

Example 2–11 Loading Agile PLM Change, Commodity, and Supplier objects

```
//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 supplier
ISupplier supplier =
(ISupplier)m_session.getObject(ISupplier.OBJECT_TYPE, "SUP20013");
System.out.println("Supplier : " + supplier.getName());
```

Creating Objects

You can use one of the following IAgileSession.createObject() methods create an Agile PLM object.

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

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

The `objectType` and `params` parameters are identical to those used in the `IAgileSession.getObject()` methods; for more information, see "Loading Objects" on page 2-8. 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 cannot 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.

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

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

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

Example 2–12 Creating a part using a Map

```
Map params = new HashMap();
params.put(ItemConstants.ATT_TITLE_BLOCK_NUMBER, "1000-02");
IItem item = (IItem)m_session.createObject(ItemConstants.CLASS_PART, params);
```

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

Example 2–13 Creating a part using a String

```
IItem item = (IItem)m_session.createObject(ItemConstants.CLASS_PART, "1000-02");
```

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 2–14 Getting Classes

```
      IAdmin m_admin = session.getAdminInstance();
// Get the Item base class
      IAgileClass itemClass =
      m_admin.getAgileClass(ItemConstants.CLASS_ITEM_BASE_CLASS);
// Get the Item subclass names
      IAgileClass[] subclasses = itemClass.getSubclasses();
      for (int i = 0; i < subclasses.length; ++i) {
      String listSubclasses = subclasses[i].getName();
      System.out.println(listSubclasses);
  }
```

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 2–15 Creating an object of a user-defined subclass

```
//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");
}
```

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 2–16 Creating objects by referencing the subclass name

```
IItem resistor = (IItem)m_session.createObject("Resistor", "R10245");
```

Using AutoNumbers

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

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

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

To get the next available AutoNumber in the sequence:

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

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

Example 2–17 Getting the next available AutoNumber

```
IAdmin admin = session.getAdminInstance();
   IAgileClass cls = admin.getAgileClass(ItemConstants.CLASS_PART);
   IAutoNumber[] numSources = cls.getAutoNumberSources();
      for (int i = 0; i < numSources.length; i++) {
     if (numSources[i].getName().equals("Part Number")) {
    String nextAutoNumber = numSources[i].getNextNumber();
    IItem part = (IItem)session.createObject(ItemConstants.CLASS_PART,
nextAutoNumber);
    System.out.println("Item: " + part.getName() + " created");
      }
       }
return null; \\Only needed if not in main method
```

Setting the Required Fields

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

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

To check for required fields:

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

Example 2–18, referenced by Example 2–19, verifies wether the single specific attribute is required and visible.

Example 2–18 Getting the required visible attributes for a class

```
/**
* Thesev attributes were set in another method. For exaple, a main method and are
passed into this method. Returns are 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;
}
```

Example 2–19 Getting the required Page 1, 2, and 3 Attributes for the 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 the 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 the 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 the 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 examples show different ways to create various types of Agile PLM objects. To simplify the code, `AutoNumbers` are not used.

Example 2–20 Creating Change, Commodity, Customer, Declaration, File Folder, Folder, Item, and Discussion

```
//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
Map params = new HashMap();
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
Map params = new HashMap();
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
Map params = new HashMap();
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 folder
Map params = new HashMap();
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 a file folder
Map params = new HashMap();
IFileFolder ff =
(IFileFolder)m_session.createObject(FileFolderConstants.CLASS_FILE_FOLDER,
"FOLDER00133");
System.out.println("File Folder : " + ff.getName());
}
```

Example 2–21 Creating Items, Manufacturers, Manufacturer Parts, Manufacturer Sites, Prices, QCRs, and PSRs

```
//Create an item
IItem part =
(IItem)m_session.createObject(ItemConstants.CLASS_PART, "1000-02");
System.out.println("Item : " + part.getName());

//Create a manufacturer
IManufacturer mfr =
(IManufacturer)m_session.createObject(ManufacturerConstants.CLASS_MANUFACTURER);
System.out.println("Manufacturer : " + mfr.getName());

//Create a manufacturer part
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 price
Map params = new HashMap();
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 =

//Create a QCR
IAgileList list = attr.getAvailableValues();
list.setSelection(new Object[] {"Fixed"});
params.clear();
params.put(ProgramConstants.ATT_GENERAL_INFO_NAME, "Wingspan Program");
params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE,
df.parse("06/01/05"));
params.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
df.parse("06/30/05"));
params.put(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE, list);
IProgram program = (IProgram)m_session.createObject(ProgramConstants.CLASS_
PROGRAM, params);
System.out.println("Program : " + program.getName());

//Create a PSR
IServiceRequest psr =
(IServiceRequest)m_session.createObject(ServiceRequestConstants.CLASS_NCR,
"NCR01562");
System.out.println("PSR : " + psr.getName());
IQualityChangeRequest qcr = (IQualityChangeRequest)m_session.createObject(
    QualityChangeRequestConstants.CLASS_CAPA, "CAPA02021");
System.out.println("QCR : " + qcr.getName());
}
```

Example 2–22 Creating a Query, Specification, Substance, Transfer Order, User, and User Group

```
//Create a query
params.clear();
IFolder parent =
(IFolder)m_session.getObject(IFolder.OBJECT_TYPE, "/Personal Searches");
String condition =

//Create a specification
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 =
"[Title Block.Number] starts with 'P'";
(IQuery)m_session.createObject(IQuery.OBJECT_TYPE, params);
System.out.println("Query : " + query.getName());
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 a `SupplierResponse`

Checking the State of Agile PLM Objects

The `IStateful` interface supports Agile objects that have either Agile Workflow status or Agile lifecycle phases. Objects that support this interface are `routable` objects.

Routable objects are:

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

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

Example 2–23 Getting the array that defines the different states of an object

```
IChange change =
     (IChange)session.getObject(ChangeConstants.CLASS_ECO, "ECO-1000-01");
IWorkflow assignedWorkflow = change.getWorkflow();
IStatus[] allStates = assignedWorkflow.getStates();
      for (int i = 0; i < allStates.length; i++) {
System.out.println(i + ", " + allStates[i].getName());
```

Example 2–24 Getting the current state of the object

```
IChange change =
      (IChange)session.getObject(ChangeConstants.CLASS_ECO, "ECO-1000-01");
IStatus currentStatus = change.getStatus();
System.out.println(currentStatus);
```

Propagating Values to Related Objects

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

Saving an Object to a New Object

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

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

Example 2–25 Saving an object as a new object

```
// 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();
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);
```

Sharing an Object

The `IShareable` interface is implemented by every Agile PLM business object that is exposed by Agile APIs. Therefore, every business object sharable. Sharing enables you to grant one or more of your roles to another Agile PLM user or an Agile PLM 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 shared an object, can perform actions permitted by the roles for that object only. They do not 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 similar to the Map returned by `getUsersAndRoles()` which maps each user to an array of roles. Each user can be assigned a different selection of roles.

There are two method, `getDataForSharing()` and `shareItem()` for this sample. Corrected the method name `getDataForSharing()` and added comment for each method in the guide

Example 2–26 Sharing an object

```
/**
* Set a specific item as shareable
*/
private void setDataForSharing() throws Exception {

//Get item
IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_ITEM_BASE_CLASS,
"P10011");

//Get users
IUser user1 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "albertl");
IUser user2 = (IUser)m_session.getObject(UserConstants.CLASS_USER, "peterl");
IUser[] users = new IUser[]{user1, user2};

//Get roles
INode nodeRoles = (INode)m_session.getAdminInstance().getNode(NodeConstants.NODE_
ROLES);
IRole role1 = (IRole)nodeRoles.getChildNode("Component Engineer");
IRole role2 = (IRole)nodeRoles.getChildNode("Incorporator");
IRole[] roles = new IRole[]{role1, role2};
//Share the item
shareItem(item, users, roles);
}

/**
* Share the specified Item for specified users with specified roles
*/
private void shareItem(IItem item,IUser[] users, IRole[] roles) throws Exception {
Map map = new HashMap();
for (int i = 0; i < users.length; i++) {
map.put(users[i], roles);
}
IShareable shareObj = (IShareable)item;
shareObj.setUsersAndRoles(map);
}
```

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

Deleting and Undeleting Objects

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

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

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

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

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

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

Example 2–27 Deleting an Item

```
IItem item = (IItem)session.getObject(IItem.OBJECT_TYPE, "1000-02");
item.delete();

// Delete the Item
   } 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);
```

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

Example 2–28 Undeleting an object

```
IItem item = (IItem)session.getObject(IItem.OBJECT_TYPE, "1000-02");
if (item.isDeleted()) {
// Restore the object
item.undelete();
}
```

Closing a Session

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

Example 2–29 Closing a session

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

3 Creating and Loading Queries

This chapter includes the following:

About Queries

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

Creating a Query

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

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

In Example 3–1, the query class is the Item class.

Example 3–1 Creating a Query

```
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();
```

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 subclass called Foobar

```
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

```
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");I
Folder folder =
(IFolder)m_session.getObject(IFolder.OBJECT_TYPE,
"/Personal Searches");
folder.addChild(query);
```

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

```
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 will automatically delete all temporary queries when the user session is closed.

Generating Ordered (sorted) or Unordered Query Results

As shown in examples for "Creating a Query" on page 3-1, executing `IQuery.execute()` and `IQuery.execute(Object[] params)` methods returns an ordered query result in `ITable`.

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

Note: To sort query results by other than the default order, see "Sorting Query Results" on page 3-25.

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

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

Example 3–5 Skip ordering in query results

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

 // The boolean is set to true to skip ordering
 ITable results = query.execute(true);
```

Specifying Query Attributes when Creating a Query

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

Table 3–1 Virtual Attributes

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 3–6 Specifying query attributes when you create a query

```
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();
```

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 is a good idea to limit the search criteria as much as possible, as the amount of data returned could be excessively large, resulting in decreased performance.

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

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

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

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

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

Search Conditions

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

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

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

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

Following are two search condition examples.

Example 3–7 Simple search condition example

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

Example 3–8 Search condition where the right operand is a set of constant expressions

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

Query Language Keywords

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

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

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

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

Specifying Search Attributes

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

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

You can specify a search attribute in the following ways:

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

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

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

Note: The APIName field, described in "Accessing Metadata Using the APIName Field" on page 9-3, does not support specifying any search attributes.

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

```
['Page Two.Unit of Measure [g or oz]']
query.setCriteria("[%0] == 'Computer'", new Object[] { attr });
```

There are other less intuitive ways to specify an attribute. For example, you can pass in an `IAttribute` reference using a parameter of the `setCriteria()` method. In Example 3–16, '%0' references the attribute in the Object array parameter.

Also, as shown below you can use String concatenation to reference an attribute constant:

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

Retrieving Searchable Attributes

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

Due to database considerations, not all attributes are searchable. Generally, users select Page One attributes such as, Title Page, Cover Page, and General Info which are searchable for each class.

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

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

To find a query's searchable attribute:

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 3-19.

Retrieving the Sequence ID of a Query

Agile PLM generates sequence IDs for all saved queries. However, these query IDs are subject to change, because they are associated with new IDs after a Re-order, because its sequence changes. The following SDK calls get a query's sequence IDs before and after a Re-Order. To load these saved queries, see "Loading Saved Queries by their IDs" on page 3-29.

Example 3–9 Getting the Sequence ID of a Saved Query

```
//Before Re-order, the id of one saved query QueryA is 1001. The SDK script is as
below when you try to get the QueryA

IQuery query = (IQuery) session.getObject(IQuery.OBJECT_TYPE, new Integer(1001));
```

Example 3–10 Getting the Sequence ID of a Query after the Re-Order

```
// After Re-order, the ID of QueryA changes. Assuming it is changed to 102, you
//must modify the SDK script to correctly get QueryA as shown below.

IQuery query = (IQuery) session.getObject(IQuery.OBJECT_TYPE, new Integer(102));
```

Using Relational Operators

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

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

Table 3–2 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.

Table 3–2 (Cont.) Relational operators

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

Using Unicode Escape Sequences

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

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

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

For contains' usage in the case of a MultiList, the query operation is different.

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

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

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

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

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

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

Using the Nested Criteria to Search for Values in Object Lists

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

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

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

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

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

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

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

The parameter query is not supported in nested queries and multiple values for one placeholder in query parameters must be specified in two dimensional arrays as shown in the following correct and incorrect examples of parameter query in nested query criteria:

- The parameter query specified in the following nested query criteria will fail to execute:

  ```
  [Page Two.User1] in ([General Info.First Name] == %0)
  ```

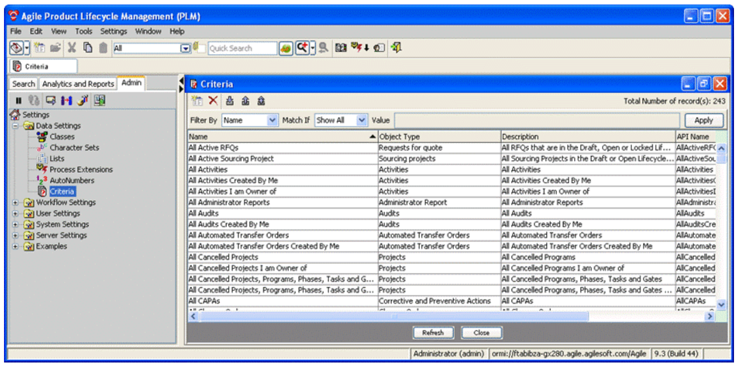
- However, when it is explicitly specified as a string value, instead of the placeholder, it will succeed:

  ```
  [Page Two.User1] in ([General Info.First Name] == 'Christopher')
  ```

Using Criteria Selected from Criteria Library in SDK Queries

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

Figure 3–1 The Criteria panel



Criteria

Total Number of record(s): 243

Filter By: Name | Match If: Show All | Value:

Name	Object Type	Description	API Name
All Active RFQs	Requests for quote	All RFQs that are in the Draft, Open or Locked Lif...	AllActiveRF(
All Active Sourcing Project	Sourcing projects	All Sourcing Projects in the Draft or Open Lifecycle...	AllActiveSou
All Activities	Activities	All Activities	AllActivities
All Activities Created By Me	Activities	All Activities Created By Me	AllActivities(
All Activities I am Owner of	Activities	All Activities I am Owner of	AllActivitiesI
All Administrator Reports	Administrator Report	All Administrator Reports	AllAdministra
All Audits	Audits	All Audits	AllAudits
All Audits Created By Me	Audits	All Audits Created By Me	AllAuditsCre
All Automated Transfer Orders	Automated Transfer Orders	All Automated Transfer Orders	AllAutomate
All Automated Transfer Orders Created By Me	Automated Transfer Orders	All Automated Transfer Orders Created By Me	AllAutomate
All Cancelled Projects	Projects	All Cancelled Programs	AllCancelled
All Cancelled Projects I am Owner of	Projects	All Cancelled Programs I am Owner of	AllCancelled
All Cancelled Projects, Programs, Phases, Tasks and G...	Projects	All Cancelled Programs, Phases, Tasks and Gates	AllCancelled
All Cancelled Projects, Programs, Phases, Tasks and G...	Projects	All Cancelled Programs, Phases, Tasks and Gates ...	AllCancelled
All CAPAs	Corrective and Preventive Actions	All CAPAs	AllCAPAs

Administrator (admin) | ormi://ftabibza-gx280.agile.agilesoft.com/Agile | 9.3 (Build 44)

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

Example 3–11 Using criteria from the Criteria Library in SDK queries

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

Using Relationships and Content in SDK Queries

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

To search using an object's Relationships:

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

Example 3–12 Using an object's Relationships as the query criteria

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

To search using a Project object's Content:

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

Example 3–13 Using a Project object's Content as the query criteria

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

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

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

Example 3–14 Using a Transfer Orders object's Selected Content as the query criteria

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

Searching for Words or Phrases Contained in Attachments

Two special attributes, `[Attachments.File Document Text]` and [Files.Document Text] are used to index files and search for files indexed by File Manager that reside on the Agile file management server. If you are hosting your database on Oracle, you can take advantage of a feature that lets you search for words or phrases contained in attachments, when you create search criteria that use either of these attributes.

There are four additional relational operators that you can use:

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

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

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

Searching for Orphaned Parts (or Parts without Parents)

Orphaned parts are parts that do not have parents. This arises from the notion that in a BOM Tree, there are Parents and Children. When the Child part no longer has any Parents, it is an Orphan part. SDK Provides the following two attributes to enable retrieving Orphan Parts:

- `Where Used.Item Number[1039]` - The null value returns all revision having no parents and `Where [1039]` is null.
- `Where Used.Item Number All Revision` - The null value returns all revision having no parents and `Where [20W00025272]` is null.

The following code sample uses these attributes to retrieve Orphan parts in the current version and all versions of the BOM.

Example 3–15 Searching for Orphan parts

```
System.out.println("Where Used.Item Number[1039] =
          " +ItemConstants.ATT_WHERE_USED_ITEM_NUMBER);
System.out.println("Where Used.Item Number All Revision[2000025272] =
          " +ItemConstants.ATT_WHERE_USED_ITEM_NUMBER_ALL_REVISIONS);
IAgileSession session =
           AgileSessionFactory.createSessionEx(params);
IQuery query =
(IQuery)session.createObject(IQuery.OBJECT_TYPE,ItemConstants.CLASS_PARTS_CLASS);
query.setCaseSensitive(false);query.setCriteria("[1039] is null");

//query.setCriteria("[2000025272] is null");
ITable results = query.execute();
System.out.println("result size:"+results.size());
```

Creating a Parameterized Query

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

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

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

Example 3–16 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 3–17 Parameterized query using IQuery.setParams()

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

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

Example 3–18 Correct use of quote characters in a parameterized search query

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

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

Example 3–19 Setting the date format for a query

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

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

Example 3–20 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 ''.

- **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 3-27. 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. For example, **"?at"** finds hat, cat, and fat, but not splat. For example, `[Title Block.Description] like '?al*'` matches any word containing "al" that is preceded by a single letter, such as tall, wall, mall, calendar, and so on.

Using Parentheses in Search Criteria

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

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

Therefore, search conditions joined by **union, intersection**, and **minusand** operators are evaluated before conditions joined by or **`or`**.

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

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

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

Setting Search Criteria for Lists Containing Large Numbers of Objects

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

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

with the following to improve performance:

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

Using SQL Syntax for Search Criteria

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

This is a simple example that demonstrates the syntax:

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

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

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

Example 3–21 Query using the standard Agile API query syntax

```
try {
IQuery query = (IQuery)m_session.createObject(IQuery.OBJECT_TYPE, "Items");
query.setCriteria("[Page Two.Nummeric01] between (1000, 2000)");
//Set result attributes
```

```
String[] attrs = { "Title Block.Number", "Title Block.Description",
query.setResultAttributes(attrs);
//Run the query
ITable results = query.execute();
} catch (APIException ex) {
System.out.println(ex);
}
```

This example shows the same query rewritten in SQL syntax. Although, the example does not contain fewer lines of code, it is more readable than Agile API query syntax. This is particularly true, if ones is familiar with SQL.

Example 3–22 Query using the 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 " +
"Page Two.Numeric01 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.

Example 3–23 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: Recall, 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 (?) wild cards 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 wild cards in the `SELECT` statement (the specified query result attributes) and the `WHERE` statement (the search condition). For example, "SELECT *" specifies all available query result attributes.

Sorting Query Results Using SQL Syntax

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

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

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

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

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

Example 3–24 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 " +
"Page Two.Nummeric01 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 3–3 Default result attributes

Query class	Default result attributes

Changes	`Cover Page.Change Type`
	`Cover Page.Number`
	`Cover Page.Description`
	`Cover Page.Status`
	`Cover Page.Workflow`

Customers	`General Info.Customer Type`
	`General Info.Customer Number`
	`General Info.Customer Name`
	`General Info.Description`
	`General Info.Lifecycle Phase`

Declarations	`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	`Cover Page.Subject`
	`Cover Page.Status`
	`Cover Page.Priority`
	`Cover Page.Type`

File Folders	`Title Block.Type`
	`Title Block.Number`
	`Title Block.Description`
	`Title Block.Lifecycle Phase`

Items	`Title Block.Item Type`
	`Title Block.Number`
	`Title Block.Description`
	`Title Block.Lifecycle Phase`
	`Title Block.Rev`

Table 3–3 (Cont.) Default result attributes

Query class	Default result attributes

Manufacturers	`General Info.Name`
	`General Info.City`
	`General Info.State`
	`General Info.Lifecycle Phase`
	`General Info.URL`

Manufacturer Parts	`General Info.Manufacturer Part Number`
	`General Info.Manufacturer Name`
	`General Info.Description`
	`General Info.Lifecycle Phase`

Packages	`Cover Page.Package Number`
	`Cover Page.Description`
	`Cover Page.Assembly Number`
	`Cover Page.Status`
	`Cover Page.Workflow`

Part Groups	`General Info.Name`
	`General Info.Description`
	`General Info.Lifecycle Phase`
	`General Info.Commodity Type`
	`General Info.Overall Compliance`

Prices	`General Info.Price Number`
	`General Info.Description`
	`General Info.Rev`
	`General Info.Price Type`
	`General Info.Lifecycle Phase`
	`General Info.Projects`
	`General Info.Customer`
	`General Info.Supplier`

Product Service Requests	`Cover Page.PSR Type`
	`Cover Page.Number`
	`Cover Page.Description`
	`Cover Page.Status`
	`Cover Page.Workflow`

Projects	`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 3–3 (Cont.) Default result attributes

Query class	Default result attributes

Sourcing Projects	`General Info.Project Type`
	`General Info.Number`
	`General Info.Description`
	`General Info.Manufacturing Site`
	`General Info.Ship To Location`
	`General Info.Projects`
	`General Info.Customer`
	`General Info.Lifecycle Phase`

Quality Change Requests	`Cover Page.QCR Type`
	`Cover Page.QCR Number`
	`Cover Page.Description`
	`Cover Page.Status`
	`Cover Page.Workflow`

RFQ Responses	`Cover Page.RFQ Number`
	`Cover Page.RFQ Description`
	`Cover Page.Lifecycle Phase`
	`Cover Page.Requested`
	`Cover Page.Completed`
	`Cover Page.Due Date`

RFQs	`Cover Page.RFQ Number`
	`Cover Page.RFQ Description`
	`Cover Page.MFG Site`
	`Cover Page.Ship-To Location`
	`Cover Page.Projects`
	`Cover Page.Customer`
	`Cover Page.Lifecycle Phase`
	`Cover Page.RFQ Type`

Sites	`General Info.Name`
	`General Info.Contact`
	`General Info.Phone`

Specifications	`General Info.Name`
	`General Info.Description`
	`General Info.Lifecycle Phase`
	`General Info.Jurisdictions`
	`General Info.Validation Type`
	`General Info.Specification Type`

Substances	`General Info.Name`
	`General Info.Description`
	`General Info.CAS Number`
	`General Info.Lifecycle Phase`
	`General Info.Substance Type`

Table 3–3 (Cont.) Default result attributes

Query class	Default result attributes

Suppliers	`General Info.Supplier Type`
	`General Info.Number`
	`General Info.Name`
	`General Info.Description`
	`General Info.Status`

Transfer Orders	`Cover Page.Transfer Order Type` (See "Retrieving the CTO Originator Name" on page 3-24.)
	`Cover Page.Transfer Order Number`
	`Cover Page.Description`
	`Cover Page.Status`
	`Cover Page.Workflow`

Specifying Result Attributes

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

Example 3–25 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
  IAribute attr1 = cls.getAttribute(ItemConstants.ATT_TITLE_BLOCK_NUMBER);
  IAribute attr2 = cls.getAttribute(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
  IAribute 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);
}
```

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

Example 3–26 Example: Getting the array of available result attributes

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

Retrieving the CTO Originator Name

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

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

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

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

Or,

```
QueryString = "[Cover Page.Originator] in ([“+UserConstants.ATT_GENERAL_
INFO_LAST_NAME+”]=='<Last_name>'”+”&&

[“+UserConstants.ATT_GENERAL_INFO_FIRST_NAME+”]=='<First_name>');
```

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

Duplicate Results for Site-Related Objects and AMLs

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

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

Working with Query Results

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

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

The `ITwoWayIterator` interface enables you to 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 4-13.

Sorting Query Results

Unlike other Agile API tables, you cannot create a sorted Iterator for query results using the `ITable.ISortBy` interface. To sort query results, use SQL syntax and specify an `ORDER BY` statement with the search criteria. For more information, see "Using SQL Syntax for Search Criteria" on page 3-16.

Query Result Datatypes

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

Important: In Agile 9.0 SDK, all values in a query results table were strings, but in post Agile 9.2, these values were converted to integers.

Using Admin Preferences Attributes to Manage Queries and Reports

Using PLM's Java Client, a user with Admin privileges can set the following Administrator Preferences by selecting **Admin > Server Settings > Preferences** and any one of these or other supported Preference. You can also set the above three Admin Preferences attributes from the SDK.

- Maximum Query Results Displayed
- Maximum BOM Reports Results
- Search Based on (Table or Row)

For more information about Admin Preferences (Systemwide Preferences), settings, and returned values, refer to *Agile PLM Administrator Guide.* For SDK-related descriptions and procedures, see "Setting the Maximum Query Results Displayed" on page 3-26, "Setting Maximum BOM Report Results" on page 3-26, and "Selecting Search Results Based on Table or Row" on page 3-27.

Setting the Maximum Query Results Displayed

This preference sets a limit on the maximum number of rows that are returned by a query and displayed on the monitor. However, this preference does not affect Agile PLM Clients. Queries that you run from an Agile SDK Client always return all results. That is, although you can access the entire query result set with the returned `ITable` object, the Agile API internally manages retrieving partial results when necessary. For example, if a particular query returns 5000 records, you can use the ITable interface to access any of these 5000 rows, regardless of how many of the 5000 rows the Agile API actually loaded into memory.

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

Setting Maximum BOM Report Results

This Admin Preference is based on the *Maximum Report Results* attribute which determines the maximum number of objects displayed in Agile PLM Custom Reports. This preference does not apply to Standard Reports and is overridden by the Full Search Display (FSD) privilege. For more information, refer to *Agile PLM Administrator Guide*.

Note: Users with the FSD privilege see all results of reports; also, all privilege checking is bypassed on users with this privilege when they view report results. Users without the FSD privilege see the maximum number of reports specified in this property.

This Preference uses the `Property.Constants.PROP_MAXIMUM_BOM_REPORT_RESULTS` attribute to set a limit on the maximum number of BOM Report Results returned.

Example 3–27 Setting Property.Constants.PROP_MAXIMUM_BOM_REPORT_RESULTS values

```
Example:  /* This example sets the maximum BOM Report Results to 666 */
   public static void testMaxBomReportResults() throws Exception {
       INode preferences =
       session.getAdminInstance().getNode(NodeConstants.NODE_PREFERENCES);
       prop =
             preferences.getProperty(PropertyConstants.PROP_MAXIMUM_BOM_REPORT_
RESULTS);
       Object original = prop.getValue();
System.out.println("Current MaxBomReportResults value:" + original);
prop.setValue("666");
System.out.println("Changed MaxBomReportResults to:" + prop.getValue());
}
```

Selecting Search Results Based on Table or Row

This preference uses the `PropertyConstants.PROP_SEARCH_BASED_ON` attribute to set the search based on Property, where Property is a list and valid values are "Table" and "Row."

Example 3–28 Setting PropertyConstants.PROP_SEARCH_BASED_ON Table or Row

```
public static void testSearchBasedOn() throws Exception {
INode preferences = session.getAdminInstance().getNode(NodeConstants.NODE_
PREFERENCES);
prop =
preferences.getProperty(PropertyConstants.PROP_SEARCH_BASED_ON);
Object original = prop.getValue();
System.out.println("Current Search Based On value:" + original);
IAgileList list = prop.getAvailableValues();
// Valid values are "Table" and "Row"
list.setSelection(new Object[] {"Table"});
prop.setValue(list);
System.out.println("Changed Search Based On property to:" + prop.getValue());
}
```

Query Performance

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

Creating a Where-Used Query

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

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

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

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

Table 3–4 Where-used 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.

Table 3–4 (Cont.) Where-used operators

Where Used set operator	Description
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 3–29 A Where-used 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();
```

Loading a Query

There are two ways to load a query:

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

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

Example 3–30 Loading a query using IAgileSession.getObject()

```
IQuery query =
(IQuery)m_session.getObject(IQuery.OBJECT_TYPE,"/Workflow Routings/Changes
Submitted To Me");
```

The following example loads a query using `IFolder.getChild()`.

Example 3–31 Loading a query using IFolder.getChild()

```
//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");}
```

Loading Saved Queries by their IDs

The following two code samples show how to load a saved query by its ID. Oracle recommends loading these saved queries by specifying the full path of the query as shown in Example 3-30

Example 3–32 Loading a Saved Query by its ID

```
// Before Re-order, the sequence id of one saved query QueryA is 61000354.
// The SDK script as shown below when you try to get the QueryA
   IQuery query =
          (IQuery) session.getObject(IQuery.OBJECT_TYPE, new Integer(61000354));
```

Example 3–33 Loading a Saved Query by its ID after a Re-Order

```
/* After a Re-order, the sequence ID of QueryA is changed. Assuming 1002,
 * it is changed to customer needs to modify the SDK script to use the new
*/ sequence id of QueryA as shown below.

     IQuery query =
        (IQuery) session.getObject(IQuery.OBJECT_TYPE, new Integer(1002));
```

Deleting Saved Queries

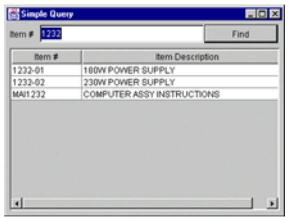
To delete a saved query, use the `IQuery.delete()` method. Temporary queries, that is, queries that are created but not saved to a folder are automatically deleted after the user session is closed. For lengthy sessions, you can use the `delete()method` to explicitly delete a temporary query after you're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

Example 3–34 Deleting a query

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

Simple Query Examples

Figure 3–2 An example of a dialog box that performs a simple query.



1232

Item #	Item Description
1232-01	180W POWER SUPPLY
1232-02	230W POWER SUPPLY
MAI1232	COMPUTER ASSY INSTRUCTIONS

The Simple Query dialog box enables the user to specify an item number as the search criteria. When you click the **Find** button, the program constructs a query to find all items that contain the specified text in the **Item#** (Item number) field. This example shows the code that runs the query when the user clicks the **Find** button.

Example 3–35 A 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);
}
```

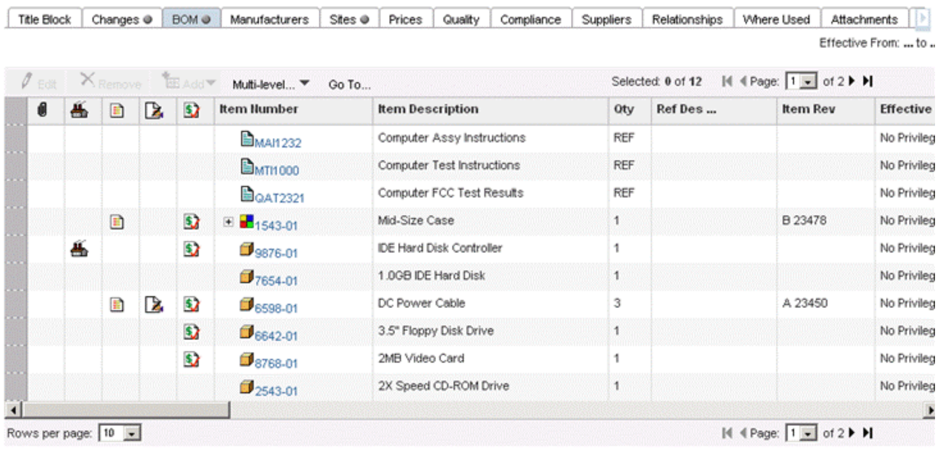
4 Working with Tables

This chapter includes the following:

About Tables

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

Figure 4–1 The BOM tab for an item in Agile Web Client.



Item Number	Item Description	Qty	Ref Des ...	Item Rev	Effective
MAI1232	Computer Assy Instructions	REF			No Privileg
MTI1000	Computer Test Instructions	REF			No Privileg
QAT2321	Computer FCC Test Results	REF			No Privileg
1543-01	Mid-Size Case	1		B 23478	No Privileg
9876-01	IDE Hard Disk Controller	1			No Privileg
7654-01	1.0GB IDE Hard Disk	1			No Privileg
6598-01	DC Power Cable	3		A 23450	No Privileg
6642-01	3.5" Floppy Disk Drive	1			No Privileg
8768-01	2MB Video Card	1			No Privileg
2543-01	2X Speed CD-ROM Drive	1			No Privileg

In some cases, a tab in Agile Web Client contains multiple tables. For example, the Changes tab for an item contains the Pending Changes table and the Change History table. The tabs and the tables that they contain is not always the same for different Agile products. Also, they are not the same for each Agile PLM Dataobject. For example, tables for Parts objects are different from tables for Manufacturers objects. See "Retrieving a Table" on page 4-3.

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`, similar to `IFolder`, extends `java.util.Collection` and supports all methods provided by that superinterface. Consequently, you can work with an `ITable` object as you would with any Java Collection.

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

Affected Read Through Tables Supported by SDK

SDK supports the Affected Read Through tables listed below.

- `Item.BOM`
- `Item.WU`
- `Item.PendingChangeWU`
- `Change.AI`
- `PSR.AI`
- `QCR.AI`
- `QCR.PSRItem`

Retrieving a Table

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

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

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

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

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

Accessing the New and Merged Relationships Tables

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

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

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

Note: For a complete list of table constants that are merged and mapped into a single constant, or mapped into a new constant, see Appendix B, "Migrating Table Constants to Release 9.2.2."

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

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

Accessing the Relationships Table

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

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

Accessing the Merged Relationships Tables

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

Accessing the Merged Relationships.AffectedBy Table

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

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

- Code recommended for this release:

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

Accessing the Merged Relationships.Affects table

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

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

- Code recommended for later releases:

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

Accessing the Merged Relationships.References Table

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

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

- Code recommended for this release:

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

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

Retrieving the BOM table of an Item

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

Example 4–1 Retrieving a BOM table

```
//Load an item
 IItem item =
       (IItem)m_session.getObject(ItemConstants.CLASS_PART, number);

//Get the BOM table
    ITable table =
         item.getTable(ItemConstants.TABLE_BOM);
```

Working with Read-only Tables

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

Retrieving the Metadata of a Table

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

Example 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;
}
```

You can also use the API Name field to identify a table's name or ID. For information to use this field, see Chapter , "Accessing Metadata Using the APIName Field." For information to use the Agile API to work with metadata, see Chapter 23, "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` is available to pass the following data:

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

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

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

You can add table rows in a batch format with `ITable.createRow()`. See "Adding and Updating Multiple Table Rows" on page 4-9.

Adding an Item to a BOM Table

The following example uses `ITable.createRow()` 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 I`Table.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.

Example 4–4 Adding a row to an Attachments table

```
private static void (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);}
}
```

Increasing the Default Value of agile.sso.expirationTime to Add Attachments

Anytime the `ITable.createRow(java.lang.Object)` method fails to add attachments and generates the `java.lang.NullPointerException` error, you must modify the value of `agile.sso.expirationTime`,the duration in seconds, between the time that you log in and authentication by the PLM FileManager. The default value is 120 seconds which is not enough, modify the default setting as shown below.

To modify the value of agile.sso.expirationTime:

1. Locate `agile.properties` file as follow:

Depending on your installation, the three possible locations for this file are:

- `<Agile_Home>\agileDomain\applications\application.ear\APP-INF\classes\`

  Note: This path is generated after PLM is installed.

- `<Agile_Home>\agileDomain\config`

  Note: This path is generated after PLM is installed.

- `<Agile_Home>\agileDomain\servers\BEJ301388-AgileServer\tmp\_WL_` user `\AgilePLM\eokg58\APP-INF\classes`

  Note: This path is generated after starting the server. You can delete it, but will regenerate as part the WebLogic Server startup process. The directory *eokg58* is generated randomly.

2. In agile.properties file, increase the value of agile.sso.expirationTime from 120 seconds to a larger number, for example, 600 seconds.
3. Restart the Agile PLM server.

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");

ManufacturerPart mfrPart =
          (IManufacturerPart)(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);
}
```

Adding a Task to the Schedule Table

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

Example 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 you are adding or updating multiple rows on the BOM table of items, or the Affected Items table of Changes.

Adding Multiple Team Members to the Team Table of a Project

The following example shows how the `ITable.createRows()` method supports the Team Table of a Project.

Example 4–8 Adding Multiple Team members to a Program with Bulk API

```
private static void createTeamRows(String[] addTeamMembers) throws APIException {
//Get the Project
  IProgram program =
    (IProgram)session.getObject(IProgram.OBJECT_TYPE, programNumber);
//Get the Team Table
  ITable teamTable =
     program.getTable(ProgramConstants.TABLE_TEAM);
  IAgileList attrRolesValues =
     teamTable.getAvailableValues(ProgramConstants.ATT_TEAM_ROLES);
     attrRolesValues.setSelection(new Object[]{"Change Analyst",
       "Program Team Member"});
//Collect team members already on Team Table
  Set presentMembers = new HashSet();
  Iterator it = teamTable.iterator();
  while(it.hasNext()) {
IRow row = (IRow)it.next();
IUser user = (IUser)row.getReferent();
presentMembers.add(user);
}
//Validate new team members and filter out existing members to and to Team Table
  IUser user = null;
  IUser[] newUsers= new IUser[addTeamMembers.length];
  int usrCount = 0;
      for(int i =0; i<addTeamMembers.length; i++ ) {
      user = (IUser)session.getObject(IUser.OBJECT_TYPE, addTeamMembers[i]);
  if(!presentMembers.contains(user) || user==null) {
      newUsers[usrCount++]=user;
      }
}
//Using createRows() API to add all Team members at onece
//In this bulk approach, make sure each map in array is complete by it self to
//create a new row in Team Table.
  List<Map> newTeam=new ArrayList<Map>();
  for (int i=0; i<usrCount; i++) {
     Map teamMap = new HashMap();
teamMap.put(ProgramConstants.ATT_TEAM_NAME, newUsers[i]);
teamMap.put(ProgramConstants.ATT_TEAM_ROLES, attrRolesValues);
teamMap.put(ProgramConstants.ATT_TEAM_ALLOCATION, 0); newTeam.add(teamMap);
  }
   teamTable.createRows(newTeam.toArray(new Object[0]));
}
```

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 4–9 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 4–10 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);
}
```

Managing PSR Affected Items Tables Using Bulk Edit APIs

Bulk Edit APIs are designed to improve PLM server performance by reducing the duration of the PSR Affected Items (AIs) edits (create, update, remove), when editing the status of these AIs. Invoking the `ITable` interface (See Metadata Interface under Types of Agile API Classes and Interfaces) deteriorates server performance because it edits a single PSR AI which triggers the entire set of the associated Events, and each associated Event in turn, invokes several additional unnecessary server calls. Thus, the resolution of this issue was sought in a similar interface that performs the functions of the `ITable` interface[1], in the Bulk execution mode.

The Bulk Edit Interface

The `com.agile.api.IServiceRequestAITable` interface enables creating, updating, and removing the entire rows of a service Request Action Item table, thus reducing a multitude of often unnecessary remote server calls to a single call. This proves to be most valuable when updating PSR AI tables having large numbers of rows.

You can cast `ITable` to `IServiceRequestAITable` table and use the methods defined in this interface.

```
IRow [] bulkCreateRows (Object[] rows) throws APIException
```

[1] See Metadata Interface under "Types of Agile API Classes and Interfaces" on page 2-1.

This bulk operation creates multiple rows with a single remote call. Rather than calling `ITable.createRow(...)` to create multiple rows, this API can improve performance and reduce unnecessary event invocations. However, if an error/warning occurred, all operations are rolled back. This method accepts an array of objects and returns an array of `IRow` corresponding objects. See `createRow(Object)` for more information on the possible makeup of each element in the rows array.

```
boolean bulkRemoveAll ( Collection c ) throws APIException
```

This operation removes all of this collection's elements from this table in a single remote call. This API can improve the performance and reduce unnecessary event Invocations. This collection's elements must be `IRow`. If an error/warning occurred, all operations are rolled back.

```
void bulkUpdateRows ( Map rows ) throws APIException
```

This bulk operation updates multiple rows with a single remote call. Rather than call `IRow.setValues(...)` for multiple rows in a table, this API can potentially improve the performance and reduce unnecessary event Invocation. If an error/warning occurred, all operations are rolled back.

Managing Table Rows in the Bulk Execution Mode

The following three bulk APIs aim to improve the performance of PLM when creating, updating, or deleting the PSR Affected Items tab rows:

- ServiceRequestAITable.bulkCreateRows(Object[] rows) throws APIException;
- ServiceRequestAITable.bulkUpdateRows(Map rows) throws APIException;
- ServiceRequestAITable.bulkRemoveAll(Collection c) throws APIException;

Example 4–11 Creating table rows in the Bulk execution mode

```
//Bulk Create
  List<Map> list = new ArrayList<Map>();
  Map map = new HashMap();
      map.put(ServiceRequestConstants.ATT_AFFECTED_ITEMS_ITEM_NUMBER,
             (IItem)session.getObject(IItem.OBJECT_TYPE,"PARTXXX"));
  list.add(map);
  aiTable.bulkCreateRows(list.toArray());

//Bulk Update
  Map retmap = new HashMap();
  Iterator it = aiTable.iterator();
  while (it.hasNext()) {
     IRow row = (IRow) it.next();
     Map map = new HashMap();
     map.put(ServiceRequestConstants.ATT_AFFECTED_ITEMS_TEXT01, "AI.Text01");
     retmap.put(row, map);
  }
  aiTable.bulkUpdateRows(retmap);

//Bulk Remove
  List list = new ArrayList();
  Iterator it = aiTable.iterator();
  while (it.hasNext()) {
     IRow row = (IRow) it.next();
     list.add(row);
  }
  aiTable.bulkRemoveAll(list);
```

Casting ITable Variables to IServiceRequestAITable Variables

Because the input and outputs parameters of ServiceRequestAITable APIs are similar to those of `ITable`, you can cast `ITable` variables to those of `IServiceRequestAITable` by renaming the method, as shown below, and performing the required tasks in the Bulk mode.

Example 4–12 ITable inputs, processes, and outputs

```
ITable affectedItemsTab = null;
affectedItemsTab = getAffectedItemsTab(isr);
affectedItemsTab.updateRows(createRowsMap);
```

Example 4–13 IServiceRequestAITable inputs, processes, and outputs

```
IServiceRequestAITable affectedItemsTab = null;
affectedItemsTab = (IServiceRequestAITable)getAffectedItemsTab(isr);
affectedItemsTab.bulkUpdateRows(createRowsMap);
```

Iterating Over Table Rows

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

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

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

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

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

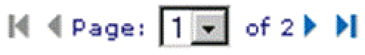
The following example demonstrates how to get an iterator for a table and use ITwoWayIterator to traverse forwards and backwards over the table rows.

Example 4–14 Iterating over table rows

```
try {
// Get an item
   IItem item = (IItem)m_session.getObject(ItemConstants.CLASS_PART, "1000-02");

// Get the BOM table
   ITable bom = item.getTable(ItemConstants.TABLE_BOM);
   ITwoWayIterator i = bom.getTableIterator();
// Traverse forwards through the table
   while (i.hasNext()) {
         IRow row = (IRow)i.next();
        // Add code here to do something with the row
}

// Traverse backwards through the table
   while (i.hasPrevious()) {
        IRow row = (IRow)i.previous();
        // Add code here to do something with the row
  }
} catch (APIException ex) {
     System.out.println(ex);
}
```

The `ITwoWayIterator` object allows a user interface to display table rows on multiple pages, which is perhaps more practical than the use of `ITwoWayIterator` shown in the preceding example. For example, instead of displaying a single scrolling page of several hundred BOM items, you can break the table into pages displaying 20 BOM items per page. To navigate from page to page, your program should provide navigation controls such as those displayed in Figure 4–2 below.

Figure 4–2 Navigation controls



Updating Objects in Query Results with Multiple Page Tables

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

To update all table rows when iterating large query results:

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

Sorting Table Rows

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

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

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

Example 4–15 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 modify the row
   }
} catch (APIException ex) {
       System.out.println(ex);
}
```

The following Product Sourcing and Projects Execution objects load tables a bit differently and therefore cannot be sorted using the `getTableIterator(ITable.ISortBy[])` method. For any tables of these objects, create an unsorted iterator using either 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 3-16.

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 4-5. 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 4–16 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 bom component 6642-01 and remove it from the table row
   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 the Collection methods to remove table rows. To remove all rows in a table, use `Collection.clear()`.

Example 4–17 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.

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

Object	Table	Referenced Object(s)
`IChange`	Affected Items	`IItem`
	Affected Prices	`IPrice`
	Attachments	`IAttachmentFile`
	Relationships	`Multiple object types`
`ICommodity`	Attachments	`IAttachmentFile`
	Compositions	`IDeclaration`
	Parts	`IItem`
	Specifications	`ISpecification`
	Substances	`ISubstance`
	Suppliers	`ISupplier`
`ICustomer`	Attachments	`IAttachmentFile`
	Quality PSRs	`IServiceRequest`
	Quality QCRs	`IQualityChangeRequest`
`IDeclaration`	Attachments	`IAttachmentFile`
	Item Composition	`ISubstance`
	Items	`IItem`
	Manufacturer Part Composition	`ISubstance`
	Manufacturer Parts	`IManufacturerPart`
	Part Group Composition	`ISubstance`
	Part Groups	`ICommodity`
	Relationships	`Multiple object types`
	Specifications	`ISpecification`
`IDiscussion`	Attachments	`IAttachmentFile`
	Where Used	`Not supported`
`IFileFolder`	Files	`IAttachmentFile`
	Relationships	`Multiple object types`
	Where Used	`Multiple object types`

Object	Table	Referenced Object(s)
`IItem`	Attachments	`IAttachmentFile`
	BOM	`IItem`
	Change History	`IChange`
	Compositions	`IDeclaration`
	Manufacturers	`IManufacturerPart`
	Pending Change Where Used	`IItem`
	Pending Changes	`IChange`
	Prices	`IPrice`
	Quality	`IServiceRequest or IQualityChangeRequest`
	Redline BOM	`IItem`
	Redline Manufacturers	`IManufacturerPart`
	Sites	`IManufacturingSite`
	Specifications	`ISpecification`
	Substances	`ISubstance`
	Where Used	`IItem`
`IManufacturerPart`	Attachments	`IAttachmentFile`
	Compositions	`IDeclaration`
	Prices	`IPrice`
	Specifications	`ISpecification`
	Substances	`ISubstance`
	Suppliers	`ISupplier`
	Where Used	`IItem`
`IManufacturer`	Attachments	`IAttachmentFile`
	Where Used	`IManufacturerPart`
`IManufacturingSite`	Attachments	`IAttachmentFile`
`IPackage`	Attachments	`IAttachmentFile`
`IPrice`	Attachments	`IAttachmentFile`
	Change History	`IChange`
	Pending Changes	`IChange`
`IProgram`	Attachments	`IAttachmentFile`
	Deliverables - Affected By	`Multiple object types`
	Deliverables - Affects	`Multiple object types`
	Dependencies - Dependent Upon	`IProgram`
	Dependencies - Required For	`IProgram`
	Discussion	`IDiscussion`
	Links	`Multiple object types`
	Schedule	`IProgram`
	Team	`IUser and IUserGroup`

Object	Table	Referenced Object(s)

`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
		`IItem`
		`IAttachmentFile`
		`IItem`
		`Multiple object types`

`IRequestForQuote`	Attachments	`IAttachmentFile`

`IServiceRequest`	Affected Items
	Attachments
	Related PSR
	Relationships
		`IItem`
		`IAttachmentFile`
		`IServiceRequest`
		`Multiple object types`

`ISpecification`	Attachments Substances	`IAttachmentFile` `ISubstance`

`ISubstance`	Attachments
	Composition
	Where Used
		`IAttachmentFile`
		`ISubstance`
		`Multiple object types`

`ISupplierResponse`	Attachments	`IAttachmentFile`

`ISupplier`	Attachments
	Manufacturers
	Quality PSRs
	Quality QCRs
		`IAttachmentFile`
		`IManufacturer`
		`IServiceRequest`
		`IQualityChangeRequest`

`ITransferOrder`	Attachments
	Selected Objects
		`IAttachmentFile`
		`Multiple object types`

`IUser`	Attachments
	Subscription
	User Group
		`IAttachmentFile`
		`Multiple object types`
		`IUserGroup`

`IUserGroup`	Attachments Users	`IAttachmentFile` `IUser`

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

Example 4–18 Retrieving a referenced Change object

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

The following example shows how to simplify the code in the previous example by using the `ITable.getReferentIterator()` method to iterate through the table's referenced objects.

Example 4–19 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 the Status Flags of a Row

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

Status flag constants are defined in the following classes:

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

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

Example 4–20 Checking the status flag of an object

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

Working with Page 1, Page 2, and Page 3

Page One (that is, Title Block, Cover Page, and General Info pages), Page Two, and Page Three contain a single row of data and are therefore not tabular in format. All other tables contain multiple rows. Consequently, the data on PLM's Page One, Page Two, and Page Three are 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 4–21 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"));
}
```

Working with Revisions and Relationships

SDK has added additional attributes in the `CommonConstants` Class to support these new features, described below.

Setting and Getting Flags for Revision Controlled Relationship Rows

SDK supports setting and getting the following flags for Relationship rows that are Revision controlled.

- Revision
- Change

- Track Impact

> **Note:** Because the impacted flag is set by releasing the change, SDK does not need to support setting the impacted flag.

To enable this feature, the following attributes are added in SDK's `CommonConstants` class:

```
public static final Integer ATT_RELATIONSHIPS_REVISION = new Integer(2000025267);
   public static final Integer ATT_RELATIONSHIPS_IMPACTED = new
Integer(2000025266);
   public static final Integer ATT_RELATIONSHIPS_TRACK_IMPACT = new
Integer(2000025268);
   public static final Integer ATT_RELATIONSHIPS_CHANGE = new Integer(2000025422);
```

The following example uses these attributes to set a revision change and track flags.

Example 4–22 Using attributes that set Revisions and track Flags

```
IItem item =
      (IItem) session.getObject(ItemConstants.CLASS_PART,"P00001");
IChange change =
      (IChange) session.getObject(ChangeConstants.CLASS_ECO,"C00001");
      item.setRevision(change);
      IRelationshipTable table = (IRelationshipTable) change.getRelationship();
      Iterator it = table.iterator();
 while (it.hasNext()) {
  IRow row = (IRow) it.next();
  Object revision = row.getValue(ItemConstants.ATT_RELATIONSHIPS_REVISION);
  Object impacted = row.getValue(ItemConstants.ATT_RELATIONSHIPS_IMPACTED);
  Object trackImpact = row.getValue(ItemConstants.ATT_RELATIONSHIPS_TRACK_IMPACT);
  Object rchange = row.getValue(ItemConstants.ATT_RELATIONSHIPS_CHANGE);
  row.setValue(ItemConstants.ATT_RELATIONSHIPS_TRACK_IMPACT, "Yes");
 }
```

Getting Flags on Relationships Tab when Adding Revision Controlled Object Revisions

The `IRelationshipTable` is the interface for operations that are specific to the Relationships table. Once a Relationships table is retrieved, you can cast the `ITable` object to `IRelationshipTable` and use the following methods:

- `getRevisions(IDataObject dataObject)` throws `APIException` - Gets all available revision values including ALL and Latest which can be used for adding revision specific relationships
- `addRevSpecificRelationship(IDataObject reatedObject,Object` - Gets all available revision values including ALL and Latest which can be used for adding revision specific relationships
- `acceptNewRevisions(IRow[] rows)` throws APIException - Accepts all new revisions
- `rejectNewRevisions(IRow[] rows) throws APIException` - Rejects all new revisions

Setting Different Relationships and Revisions for Items

Because SDK can interact with Revision, Change, Impacted attributes and the Track Impact attribute, you can set different relationships when setting different revisions for an Item. This is shown in Example 4–23.

Example 4–23 Setting changes for Item's Revisions and Relationships

```
// When Revision specific relationship is enabled on Change Object, you can add
// two different Revisions of the same Part object into one Change Object's
// relationship table.
IChange change = (IChange) session.getObject(ChangeConstants.CLASS_ECO, "C00001");
IRelationshipTable table = (IRelationshipTable) change.getRelationship();

IItem item = (IItem) session.getObject(ItemConstants.CLASS_PART, "P00001");
item.setRevision('A C00001');
// Add Part "P00001" with Rev "A" into Change "C00001"'s relationship
table.addRevSpecificRelationship(item, null);

item.setRevision('B C00002');
// Add Part "P00002" with Rev "B" into Change "C00002"'s relationship
table.addRevSpecificRelationship(item, null);
```

Redlining

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

The Agile PLM system provides the following redline tables:

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

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

Redlining BOMs, Manufacturers, and Price Lines Tables

Use the following procedure to redline these tables.

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 enable modifying an item's BOM or Manufacturers tables
   - MCOs enable modifying an item's Manufacturers table
   - SCOs enable modifying an item's site-specific BOM, Manufacturers

- PCOs enable modifying a price's Price Lines table

3. Add the item or price to the Affected Items or Affected Prices table of the change.
4. For ECOs and PCOs, specify the new revision for the change.
5. SCOs and MCOs do not affect an item's revision.
6. Modify a redline table, such as the Redline BOM, Redline Manufacturers (AML), Redline Price Lines.

Redlining the Manufacturers table of an item

Example 4–24 redlines the Manufacturers table (AML) of an item.

Example 4–24 Redlining the Manufacturers table of an item

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

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

// Get the Preferrred status value
IAgileClass cls = item.getAgileClass();
attrPrefStat = cls.getAttribute(ItemConstants.ATT_MANUFACTURERS_PREFERRED_STATUS);
listvalues = attrPrefStat.getAvailableValues();

// 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);
}
```

Redlining the Title Block of an Item

The following is an example of redlining the Title Block table of the item. It assumes Item.Page_Two and the attribute Text01 are visible and Text01 is change controlled.

Example 4–25 Redlining the Title Block table of an item

```
ITable page2Tab = item.getTable(ItemConstants.TABLE_REDLINEPAGETWO);
Iterator it = page2Tab.getTableIterator();
IRow redPage2Row = (IRow)it.next();
ICell cell = redPage2Row.getCell(CommonConstants.ATT_PAGE_TWO_TEXT01);
System.out.println("old value, before update: " + cell.getOldValue());
redPage2Row.getCell
      (CommonConstants.ATT_PAGE_TWO_TEXT01).setValue("updated Text01);
```

Removing Redline Changes

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

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

Example 4–26 Removing redline changes from the BOM table

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

Removing Redline Changes in Bulk Mode

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

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

Example 4–27 Typecasting Redline tables to IRedlinedTable interface

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

Note: Only Redline tables can be typecasted to `IRedlinedTable` interface in the following two ways. For more information, see "Redlining" on page 4-23.

- `((IRedlinedTable)bomTable).undoRedline(rows);`
- `((IRedlinedTable)bomTable).undoAllRedline();`

Identifying Redlined Rows and Redlined Cells

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

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

Example 4–28 Identifying redlined rows and cells

```
public boolean isRedlineModified()
     throws APIException;
```

Note: The IRedlined.isRedlineModified() method returns a boolean value. This value is TRUE when cells or rows are redlined and FALSE value for all cells on redline added or redline removed rows.

Using ICell.getOldValue

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

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

5

Working with Data Cells

This chapter includes the following:

- About Data Cells
- Data Types
- Checking the Discovery Privilege of the User
- Checking if the Cell is a Read-Only Cell
- Getting Values
- Setting Values
- Getting and Setting List Values
- Using Reference Designator Cells

About Data Cells

An ICell object is a data field for an Agile PLM object that you have loaded or created in your program. A cell can correspond to a field on a tab in Agile Web Client or a single cell on a table. The `ICell` object consists of several properties that describe the current state of a cell. Most of the data manipulation your Agile API programs perform involves changes made 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. The table below lists the object types of cell values for `getValue()` and `setValue()` methods.

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

DataTypeConstants	Object type associated with getValue and setValue
TYPE_TABLE	Table

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

Checking if the Cell is a Read-Only Cell

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

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

Example 5–1 Checking whether a field is a read-only field

```
// ID for "Title Block.Description"
// Set the value for the Description text field.
try {
     Integer attrID =
          ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION;
          txtDescription.setText(item.getValue(attrID).toString());

// Get the ICell object for "Title Block.Description"
// If the cell is read-only, disable the cell
   ICell cell = item.getCell(attrID);
     if (cell.isReadOnly()) {
     txtDescription.setEnabled(false);
     txtDescription.setBackground(Color.lightGray);
}
    else {
     txtDescription.setEnabled(true);
     txtDescription.setBackground(Color.white);
```

Getting Values

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

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

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

Note: You can use `ICell getAPIName()` to access Data Cell attribute values. For information to use this field, see Chapter , "Accessing Metadata Using the APIName Field."

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

Example 5–2 Specifying a cell by ID

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

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

Example 5–3 Specifying a field by its 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 5–4 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 the set values

Example 5–5 Getting a value using IDataObject.getValue(Object cellID)

```
private static Object getDescVal(IItem item) throws APIException {
   Integer attrID =
      ItemConstants.ATT_TITLE_BLOCK_NUMBER;
   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 3-25.

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

Understanding SDK Date Formats and User Preferences

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

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

Setting Values

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

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

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

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

Example 5–6 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 its values.

Example 5–7 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 `setValue()` maps attributes to cell values.

Example 5–8 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 do not 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 on handling exceptions, see Chapter 21, "Handling Exceptions."

Getting and Setting List Values

There are two different datatypes for list cells. One for `SingleList` and one for `MultiList` cells. When you get the value for a `SingleList` or `MultiList` cell, the object returned is an `IAgileList` object. For that reason, list cells are slightly more complicated to work with than other cells. The `SingleList` 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 13-17.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 5–9 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. You cannot change a `MultiList` cell, into a cascading list. The following example shows how to get and set values for a `MultiList` cell, and `Title Block.Product Lines` for an item.

Example 5–10 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

You can reconfigure a `SingleList` cell to a cascading list. A cascading list presents a list in multiple hierarchical levels, enabling you to drill down to a specific value in the list hierarchy. For more information about cascading lists, see "Cascading Lists" on page 13-2.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 5–11

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

   String value = null;
// Get the Page Two.List01 value
   IAgileList clist =
      (IAgileList)item.getValue(ItemConstants.ATT_PAGE_TWO_LIST01);

// Convert the current value from the list to a string
   value = clist.toString();
   return value;
}
private void setCascadeValue(IItem item) throws APIException {
   String value = null;

// Get the Page Two List01 cell
   ICell cell = item.getCell(ItemConstants.ATT_PAGE_TWO_LIST01);

// Get available list values for Page Two List01
   IAgileList values = cell.getAvailableValues();

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

Although the previous example shows one way to set the value for a cascading list, there is another longer form you that can use which 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 5–12 Setting the value for a cascading list (long form)

```
private void setCascadeValue(IItem item) throws APIException{
// Get the Page Two List01 cell
   ICell cell = item.getCell(CommonConstants.ATT_PAGE_TWO_LIST01);

// Get available list values for Page Two List01
   IAgileList values = cell.getAvailableValues();

// Set the continent to "North America"
   IAgileList continent = (IAgileList)values.getChildNode("North America");

// Set the country to "United States"
```

```
   IAgileList country = (IAgileList)continent.getChildNode("United States");

// Set the city to "San Jose."
   IAgileList city = (IAgileList)country.getChildNode("San Jose");
   values.setSelection(new Object[]{city});

// Set the cell value
   cell.setValue(values);
}
```

Using Reference Designator Cells

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

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

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

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

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

6 Working with Folders

This chapter includes the following:

- 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. It has its own interface called the `IFileFolder`. Files in a file folder holds can be referenced from the Attachments table of other objects. For more information about file folders, see Chapter , "Working with Attachments Table of an Object."

These are some of the Agile PLM folders:

- Private - Folders that are accessible only to the user that created them. Users can create or delete their own Private folders.
- Public - Folders that are accessible to all Agile PLM users. Only users with the `GlobalSearches` privilege can create, delete, and modify Public folders.
- System - Predefined folders that ship with the Agile PLM system. Most users cannot modify or delete System folders.
- My Bookmarks (or Favorites) - A predefined folder containing each user's bookmarks to Agile PLM objects. You cannot delete the My Bookmarks folder.
- Home - The predefined Agile PLM home folder. You cannot delete the Home folder.
- Personal Searches - The predefined parent folder for each user's personal searches. You cannot delete the Personal Searches folder.

- Recently Visited - A predefined folder containing links to recently visited objects. The SDK does not populate this folder. It is only populated by Client applications. If required, you specify this in your application

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

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

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

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

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

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

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

Using Level Separation Characters in Folders and Object Names

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

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

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

- `\|`
- `\/`

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

Loading a Folder

There are two ways to load a folder:

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

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

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

Example 6–1 Loading a folder with IAgileSession.getObject()

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

The following example loads a folder by specifying its path relative to another folder. In this case, the user's Home Folder.

Example 6–2 Loading a folder with IFolder.getChild()

```
//Get the Home Folder
try {
   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

```
//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);
```

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

```
//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);
```

Adding and Removing Folder Elements

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

Adding Folder Elements

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

Example 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);
   }
}
```

Removing Folder Elements

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

Example 6–6 Removing objects from a Folder

```
IItem item = (IItem) session.getObject(IItem.OBJECT_TYPE, "1000");
        IUser  currentUser = session.getCurrentUser();
  IFolder bookmarks =
        currentUser.getFolder(FolderConstants.TYPE_FAVORITE);
  ITreeNode bookmarksSubFolder =
        favorites.getChildNode("My Parts");
  bookmarksSubFolder.addChild(item);
```

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

```
  IItem item = (IItem) session.getObject(IItem.OBJECT_TYPE, "1000");
        IUser  currentUser = session.getCurrentUser();
   IFolder bookmarks = currentUser.getFolder(FolderConstants.TYPE_FAVORITE);
   ITreeNode bookmarksSubFolder = favorites.getChildNode("My Parts");
         bookmarksSubFolder.removeChild(item);
```

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 option 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 forward and backward, use `IFolder.getFolderIterator()` to return an `ITwoWayIterator` object. `ITwoWayIterator` provides the `previous()`, `next()`, and `skip()` methods, among others.

Example 6–9 Iterating over folder elements

```
//Load the Project X folder
try {
   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.

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

7 Working with Design Change Order Objects

This chapter includes the following:

- About Design Change Orders
- Creating Design Change Objects
- Creating a Folder
- Setting the Folder Type
- Adding and Removing Folder Elements
- Getting Folder Elements
- Deleting a Folder

About Design Change Orders Objects and Design Objects

A Design Change Orders (DCO) object also called a Design File Change Orders (DFCO) object is a subclass of the Change Orders class and is available when the affected File Tab is enabled and provides access to all Agile PLM Workflow functions.

A Design Object is a subclass of the Design Class. The relationship of a DCO to a Design Object is similar to that of an Engineering Change Order and Item. A Design object is another type of a file container object and is used with Agile PLM's Engineering Collaboration (EC) tool, EC provides data and process integration between Computer Assisted Design (CAD) applications and Agile PLM. This integration enables managing tasks related to the creation, modification, and maintenance of CAD data model structures associated with Agile Design objects.

Integration of Agile PLM and Engineering Collaboration

The integration of Agile PLM's Engineering Collaboration (EC) and Agile PLM enables accessing the Change order objects' Affected Files table and the Design/File Folder objects' Changes table. SDK also supports Affected Files enable, but you must disable the `DFCO.Affected Items` table when you are enabling the `DFCO.Affected` files table via SDK.

To enable the DFCO Affected Files tables, see Example 7–1, "Enabling the AFCO Affected Files tables" and to view actions supported by the SDK see "Change Order File Folders Actions Supported by SDK" on page 7-2.

Note: Similar to other tables, the SDK supports all table-related tasks listed in Chapter 4, "Working with Tables."

Example 7–1 Enabling the AFCO Affected Files tables

```
//enable DFCO AF table
   INode afTab = session.getAdminInstance().getNode("DFCO.AffectedFiles");
   IProperty visibleAF =afTab.getProperty(PropertyConstants.PROP_VISIBLE);
   visibleAF.setValue("Yes");

//disable DFCO AI table
   INode aiTab = session.getAdminInstance().getNode("DFCO.AffectedItems");
   IProperty visibleAI = aiTab.getProperty(PropertyConstants.PROP_VISIBLE);
   visibleAI.setValue("No");
```

Note: SDK also supports enabling the Affected Files table. However, you must disable the `DFCO.Affected Items` table when enabling the `DFCO.Affected` files table using the SDK.

Change Order File Folders Actions Supported by SDK

SDK supports Web Client actions on the following Change Order file folders:

- Change Order Affected Files Tab
- Redline/Markups Folders in Affected Files Lower Table
- Affected Files Titleblock
- File Folder and Design Objects

Note: For information on these Change Order File Folders, refer to Working with Design File Change Orders (DFCOs) in Agile PLM's Getting Started Guide.

Actions Supported on the Change Order Affected Files Tab

Actions, relevant APIs, and code samples appear below.

- Supported Action: Get Files
- API: `IAFRow.GetFile()`
- Code Sample: See Example 7–2.

Example 7–2 Get Files

```
public static void getFile(IAgileSession m_session, IChange dfco) throws
     Exception {
ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
      Iterator iterator = afTable.iterator();
      IRow selectedRow = (IRow) iterator.next();
      InputStream inputStream = ((IAFRow) selectedRow).getFile();
      }
```

- Supported Action: Check In
- API: `IAFRow.checkIn()`
- Code Sample: See Example 7–3.

Example 7–3 Check In

```
public static void checkIn(IAgileSession m_session, IChange dfco)
throws Exception {
ITable afTable =dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
       Iterator iterator = afTable.iterator();
       IRow selectedRow = (IRow) iterator.next();
       ((IAFRow) selectedRow).checkIn();
```

- Supported Action: Check Out
- API: `IAFRow.checkOutEx()`
- Code Sample: See Example 7–4.

Example 7–4 Check Out

```
public static void checkOut(IAgileSession m_session, IChange dfco)
throws Exception {
    ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
         Iterator iterator = afTable.iterator();
         IRow selectedRow = (IRow) iterator.next();
         ((IAFRow) selectedRow).checkOutEx();
         }
```

- Supported Action: Cancel Checkout
- API: `IAFRow.cancelCheckout()`
- Code Sample: See Example 7–5.

Example 7–5 Cancel Checkout

```
public static void cancelCheckOut(IAgileSession m_session, IChange dfco)
throws Exception {
   ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
         Iterator iterator = afTable.iterator();
         IRow selectedRow = (IRow) iterator.next();
         If (((IAFRow) selectedRow). isCheckedOut ()) {
            ((IAFRow) selectedRow).cancelCheckout();
         }
}
```

- Supported Action: Add Files to Change
- API: `ITable.createRow(Object param)`
- Code Sample: See Example 7–6.

Example 7–6 Add Files to Change

```
public static void addToChangeOrder(IAgileSession m_session,
       IChange dfco, String filePath) throws Exception {
       ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
         afTable.createRow(filePath);
         }
```

- Supported Action: Add Files to File Folder
- API: `ITableAF.createRow(IRow selectedRow, Object param)`
- Code Sample: See Example 7–7.

Example 7–7 Add Files to File Folder

```
public static void addToSelectedFileFolder(IAgileSession m_session,
         IChange dfco, String filePath) throws Exception {
ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
        Iterator iterator = afTable.iterator();
        IRow selectedRow = (IRow) iterator.next();
        ((IAFRow) selectedRow).checkOutEx();
        ((ITableAF) afTable).createRow(selectedRow, filePath);
        ((IAFRow) selectedRow).checkIn();
        }
```

- Supported Action: Add by search
- API: `ITable.createRow(Object param)`
- Code Sample: See Example 7–8.

Example 7–8 Add by Search

```
public static void addToChangeOrder
        (IAgileSession m_session, IChange dfco, IFileFolder fileFolder)
               throws Exception {
   ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
      afTable.createRow(fileFolder);
}
```

- Supported Action: Update Affected Files
- API: `IRow.setValue(Object cellId, Object value)`
- Code Sample: See Example 7–9.

Example 7–9 Update Affected Files

```
public static void updateAffectedFiles(
      IAgileSession m_session, IChange dfco) throws Exception {
ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
       Iterator iterator = afTable.iterator();
       selectedRow = (IRow) iterator.next();
selectedRow.setValue
     (ChangeConstants.ATT_AFFECTED_FILES_FILE_DESCRIPTION, "new value");
       }
```

- Supported Action: Remove Files from Change
- API: `ITable.removeRow(IRow row)`
- Code Sample: See Example 7–10.

Example 7–10 Remove Files from Change

```
public static void removeFromChangeOrder
        (IAgileSession m_session, IChange dfco) throws Exception {
ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES)
              Iterator iterator = afTable.iterator();
              IRow  selectedRow = (IRow) iterator.next();
              afTable.removeRow(selectedRow);
}
```

- Supported Action: Remove Files from File Folder
- API: `ITableAF.removeRow(IRow row, boolean isFromFileFolder)`
- Code Sample: See Example 7–11.

Example 7–11 Remove Files from File Folder

```
public static void removeFromSelectedFileFolder
            (IAgileSession m_session, IChange dfco) throws Exception {
   ITableAF afTable =
           (ITableAF) dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
   Iterator  it = afTable.getTableIterator();
   List rowList = new ArrayList();
       while(it.hasNext()){
   IRow row=(IRow)it.next();
   rowList.add(row);
       }
   afTable.removeRow((IRow) rowList.get(0), true);
}
```

- Supported Action: Incorporate/ Unincorporate Affected Files Folders (New)
- API: `IAFRow.setIncorporated(boolean incorporated)`
- Code Sample: See Example 7–12.

Example 7–12 Incorporate/ Unincorporate Affected Files Folders

```
public static void Incorporate
     (IAgileSession m_session, IChange dfco) throws Exception {
ITable afTable = dfco.getTable(ChangeConstants.TABLE_AFFECTEDFILES);
                 Iterator iterator = afTable.iterator();
                 IRow selectedRow = (IRow) iterator.next();
                 ((IAFRow) selectedRow).setIncorporated(true);
                 ((IAFRow) selectedRow).setIncorporated(false);
}
```

Actions Supported for Redline/Markups Folders in Affected Files Lower Table

Supported actions, relevant APIs, and code samples appear below.

Example 7–13 Add Redline/Markups

```
public static void addMarkup(IAFRow afRow) throws Exception {
    File file1 = new File("c:/test2.mkd");
    IMarkupTable tableMF = afRow.getMarkupTable();
    IRow rowMF = tableMF.createRow(row, file1);
}
```

- Supported Action: Update Redline/Markups
- API: `IRow.setValue(Object cellId, Object value)`
- Code Sample: See Example 7–14.

Example 7–14 Update Redline/Markups

```
public static void updateMarkup(IAFRow afRow, String desc) throws Exception {
   IMarkupTable tableMF = ((IAFRow) row).getMarkupTable();
   Iterator it2 = tableMF.getTableIterator();
   If(it2.hasNext()){
      IRow rowMF = (IRow) it2.next();
      rowMF.setValue(ChangeConstants.ATT_MARKUP_FILES_MARKUP_DESCRIPTION, desc);
   }
}
```

- Supported Action: Remove Redline/Markups
- API: `ITable.removeRow(IRow row)`
- Code Sample: See Example 7–15.

Example 7–15 Remove Redline/Markup Files

```
public static void updateMarkup(IAFRow afRow) throws Exception{
   IMarkupTable tableMF = ((IAFRow) row).getMarkupTable();
   Iterator it2 = tableMF.getTableIterator();
   If(it2.hasNext()){
        IRow rowMF = (IRow) it2.next();
        tableMF.removeRow(rowMF);
   }
}
```

Actions Supported on the Affected Files Titleblock Folder

SDK supports the following Web Client actions on Affected Files Titleblock file folders:

- Supported Action: `RedlineTitleblock`
- APIs:
  - `ICell.setValue(Object value) //set redline value`
  - `ICell.getOldValue()//get previous value`
- Code Sample: See Example 7–16.

Example 7–16 RedlineTitleblock

```
public static void redlineFileFolderAttr
   (IAgileSession m_session,IFileFolder fileFolder,String dfcoNumber,int
                      String dfcoNumber,int redlineAtt,String redlineValue)
                        throws Exception {
              fileFolder.setRevision(dfcoNumber);
              ITable redlineP2Table =
                 fileFolder.getTable(FileFolderConstants.TABLE_REDLINEPAGETWO);
              IRow redlineP2Row = (IRow) redlineP2Table.iterator().next();
              ICell cell = redlineP2Row.getCell(redlineAtt);
              cell.setValue(redlineValue);
              }
```

Actions that Support File Folder and Design Objects

SDK supports the following Web Client actions for File Folder Design Objects.

- Supported Action: Load Table
- API: `IDataObject.getTable(Object tableId)`
- Code Sample: See Example 7–17.

Example 7–17 Load Table

```
public static void loadPendingChangesTable(IFileFolder fileFolder)
           throws Exception {
        ITable tablePendingChanges =
             fileFolder.getTable(FileFolderConstants.TABLE_PENDINGCHANGES);
        Iterator iterator = tablePendingChanges.iterator();
}
```

8 Working with Items, BOMs, and AMLs

This chapter includes the following:

- Working with Items
- Working with BOMs
- Working with AMLs

Working with Items

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

Items are different from other Agile PLM objects because they:

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

Supported Page 2 Item Attributes

SDK supports the following Page 2 Item attributes.

- `Item.P2.Text26 to Text50` (25 new constants)
- `IItem.P2.List26 to List50` (25 new constants)
- `Item.P2.MultiList16 to MultiList25` (10 new constants)
- `Item.P2.MultiList46 to MultiList52` (7 new constants

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.

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

Example 8–1 Getting and setting the revision of an item

```
try {
// Get an item
   IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-02");

// Print the item's current revision
   System.out.println("current rev : " + item.getRevision());

// Get all revisions for the item
   Map revisions = item.getRevisions();

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

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

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

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

- A null object to specify an Introductory revision:

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

- An IChange object associated with a particular revision:

  `item.`**`setRevision(changeObject);`**

A change number (a String) associated with a particular revision:
`item.`**`setRevision("C00450")`**

revision identifier (a String such as "Introductory", "A", "B", "C", and so on): `item.`**`setRevision("A");`**

- A String containing both a revision identifier and a change number separated by eight spaces ("A   23450"):

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

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

Example 8–2 Setting the revision using BOM.Item Rev

```
// Get an Item
try {
    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 where bomRev = revID + 8 spaces +
       changeNumber)
   String bomRev = (String)row.getValue(ItemConstants.ATT_BOM_ITEM_REV);

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

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

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

Failing to get a of an Item using the Title Block.Rev field

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

Example 8–3 Failing to get a revision using the Title Block.Rev field

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

Changing the Incorporated Status of a Revision

Each revision of an item can be incorporated. When you incorporate the revision of an item, all attachments for that revision are locked and cannot be checked out. After an item is incorporated, you can still use Agile Web Client to view the item's attachments, but you cannot modify them unless you submit a new Change.

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

Only items that have revision numbers can be incorporated. Therefore, a preliminary item that has not been released cannot be incorporated. Once an ECO is submitted for that item and a pending revision number is specified, the revision can then be incorporated. Example 8–4 shows how to change the incorporated status of an item.

Example 8–4 Changing the incorporated status of an Item

```
try {
// Get an item
   IItem item = (IItem)m_session.getObject(IItem.OBJECT_TYPE, "1000-02");

//Incorporate or Unincorporate the Item, dedepending 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. For an example to retrieve a BOM table, see "Retrieving a Table" on page 4-3.

Table 8–1 BOM and ID Constants

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

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 8–5 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(((IAgileList)m_session.getCurrentUser().getValue()
      UserConstants.ATT_GENERAL_INFO_DEFAULT_SITE)).getSelection()[0].getValue();
   item.getTable(ItemConstants.TABLE_BOM).createRow(map);
}
```

For more information about manufacturing sites, see Chapter 11, "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 8–6 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

Frequently, the BOMs of two items can 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. You can use the `Collection.addAll()` method to copy the contents of one table into a target table. The `addAll()` method does not set a new revision for the item.

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

Example 8–7 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 8–8 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());
   }
}
```

Creating BOM-Related Product Reports

The SDK provides the `IProductReport` API with constants defined in `ProductReportConstants` to prepare the following BOM-related product reports. These reports are produced in the XML format.

- **BOM Explosion reports -** The BOM Explosion report displays the items that are in the bill of Material(BOM) for the one or more specified assembly, up to the desired number of levels.
- **BOM Comparison reports -** The BOM Comparison XML reports is the result of comparing two different BOMs up to the specified number of levels.

For example, when a base BOM compared with the target BOM, the comparison will show:

- **d** shown in BOM node - indicates only base assembly has the BOM
- **a** shown in BOM node - indicates only target assembly has the BOM
- **u** shown in BOM node - indicates both root assemblies have the same BOM
- **m** shown in BOM node - indicates both root assemblies have the BOM but with some differences

All **first** level BOMs of both base and target assembly are categorized into another node **BOMs**. BOM nodes under **BOMs** are first sorted by FindNum and then by ItemNumber.

There are several use cases for these reports. For example, archiving or comparative analysis with outputs of ERP systems.

To create a product report, you must use the IAgileSession object. The following examples show how to use `IAgileSession and ProductReportConstants` to prepare BOM Explosion and BOM Comparison reports.

Example 8–9 Preparing a BOM Comparison report

```
Map param = new HashMap();
   param.put(ProductReportConstants.REPORTPARAM_REPORT_TYPE,
   ProductReportConstants.REPORT_BOM_COMPARISON);
   param.put(ProductReportConstants.REPORTPARAM_ITEMREVSITE, "item1;item2");
   param.put(ProductReportConstants.BOMCOMP_BOM_ATTRS,
      ProductReportConstants.BOM_ATT_ITEM_NUM +”;” +
         ProductReportConstants.BOM_ATT_FIND_NUM);
   param.put(ProductReportConstants.BOMCOMP_BOMLEVEL, "4");

IProductReport report = (IProductReport)
   session.createObject
     (IProductReport.OBJECT_TYPE, "My BOM Comparison Report");
   String xmlReport = report.execute(param);
```

If the value for `ProductReportConstants.BOMCOMP_BOM_ATTRS` is not specified, then it is assumed its value is `Find Num;Item Number;Sites`.

Example 8–10 Preparing a BOM Explosion report

```
Map param = new HashMap();
   param.put(ProductReportConstants.REPORTPARAM_REPORT_TYPE,
   ProductReportConstants.REPORT_BOM_EXPLOSION);
   param.put(ProductReportConstants.BOMEXP_OBJTYPE, "Document;Part;");
   param.put(ProductReportConstants.
         REPORTPARAM_ITEMREVSITE, "MM75-01|23450|India;");
   param.put(ProductReportConstants.BOMEXP_MAXLEVEL, "5");
IProductReport report = (IProductReport)
   session.createObject(IProductReport.OBJECT_TYPE, "My BOM Explosion Report");
String xmlReport = report.execute(param);
```

```
In BOM Explosion reports, following are values for
ProductReportConstants.REPORTPARAM_ITEMREVSITE:
```

- `<Item_number>|<Change_number>|<Site_number> where <Change_Number> and <Site_number> are optional if:`

    -<Change_number> is not specified it is assumed to be the Latest revision

    -<Site_number> is not specified it is assumed as Common Sites

- `The value can have one or more Items delimited by semicolon`
- `Item1;Item2;Item3 are the Latest revision of Item1,Item2 and Item3 for Common Sites`
- `Item1|ECO1;Item2;Item3 (Item1 with ECO1 revision and latest revision of Item2, Item3)`
- `Item1|ECO1|Site1;Item2|ECO2 (Item1 with ECO1 revision with Site1 Specific BOM and Item2 with ECO2 revision)`
- `Item1|Site1;Item2 (Item1 with Site1 Specific BOM and the latest revision of Item2 with Common Sites)`

In BOM Comparison reports, following are values for `ProductReportConstants.REPORTPARAM_ITEMREVSITE`:

- `<Item_number>|<Change_number>|<Site_number> where <Change_Number> and <Site_number>are optional when:`

    -<Change_number> is not specified, then it is assumed as Latest revision.

    -<Site_number> is not specified, it is assumed as Common Sites.

- `The value must include two Items delimited by a semicolon`
- `Item1;Item2 (Latest revision of Item1 andItem2 and all Sites)`
- `Item1|ECO1;Item2 (Item1 with ECO1 revision and Latest revision of Item2)`
- `Item1|ECO1|Site1;Item2|ECO2 (Item1 with ECO1 revision with Site1 Specific BOM and Item2 with ECO2 revision)`
- `Item1|Site1;Item2 (Item1 with Site1 Specific BOM and Latest revision of Item2 with Common Sites)`

Redlining a BOM

To redline a BOM table, follow these steps:

1. Get a released assembly item.
2. Create a new Change Order, such as an ECO, for the item.
3. Add the item to the Affected Items table of the ECO. Also, specify the new revision in the change and set the item's revision to the associated change.
4. Modify the item's Redline BOM table.

In the following sections, there are code examples for each of these steps.

Note: You can remove redlines from a row of the BOM table. See "Removing Redline Changes" on page 4-25.

Getting a Released Assembly Item

The following example loads an assembly item from the Part subclass. Make sure the Part you specify is released and has a BOM.

Example 8–11 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);

//Check if the item is released and has a BOM
   if (item != null) {
    if (item.getRevision().equals("Introductory") ||
         !item.isFlagSet(ItemConstants.FLAG_HAS_BOM)){
   System.out.println("Item must be released and have a BOM.");
   item = null;
   }
   return item;
}
```

Creating a Change Order

To redline a BOM, you must create a Change Order, such as an ECO. Example below shows how to create an ECO and select a Workflow for the selected ECO.

Example 8–12

```
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 Items 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 in the ECO, and then set the revision for the Part to the one associated with the ECO.

Example 8–13 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 8–14 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

An AML is the Approved Manufacturer List for an item. It lists the preferred or alternate manufacturers that have been approved to supply a particular item. When you source parts for a project, you can choose to source them by assembly, by AML, or both. 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.MfrName. Each row of the Manufacturers table references a separate manufacturer part. In addition to the Manufacturers table, there is also a redline Manufacturers table, which records redline changes. When you load a Manufacturers table using the `DataObject.getTable()` method, make sure you specify the correct table ID constant.

BOM Table	ID Constant
Current Manufacturers table	`ItemConstants.TABLE_MANUFACTURERS`
Redline Manufacturers table	`ItemConstants.TABLE_REDLINEMANUFACTURERS`

Adding an Approved Manufacturer to the Manufacturers Table

Similar to the BOM Table, the Manufacturers Table requires that you specify the manufacturing site before adding a new row to the table. An approved manufacturer is either site-specific or common to all sites. Use the `IManufacturingSiteSelectable.setManufacturingSite()` method to specify the site. To add an approved manufacturer to the common Manufacturers table, use `ManufacturingSiteConstants.COMMON_SITE`. Otherwise, select a specific site, such as the user's default site.

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

Example 8–15 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 11, "Managing Manufacturing Sites."

Updating AML Split Percentages in Sourcing Projects

An AML can have manufacturer parts of status preferred, alternate, or any additional status that is configured in the Java Client. For each internal item, there can be multiple manufacturer parts. In such cases, you can specify AML percentage splits. For example, you may want to use 70% of one manufacturer part and 30% of another, based on cost or availability. The sum of the AML percentage splits must equal 100. Manufacturers listed in a project AML must already exist in the Item Master.

SDK exposes the `setAmlSplits()` method to update AML Splits by percentage, the same as in PLM Clients.

Example 8–16   Updating AML Split in a Sourcing Project

```
public static void setAmlSplits() throws APIException {
   IProject prj =
      (IProject) session.getObject(IProject.OBJECT_TYPE, "PRJ_A");
   ITable tbl_items=prj.getTable(ProjectConstants.TABLE_ITEMS);
   HashMap[] mapx = new HashMap[2];
   IRow[] rowArray = new IRow[2];
   HashMap rows = new HashMap();
   Iterator itms= tbl_items.getTableIterator();
   while (itms.hasNext()){
      IRow row_item = (IRow) itms.next();
      ITable tbl_aml = (ITable)row_item.getValue(ProjectConstants.ATT_ITEMS_AML);
      Iterator amls = tbl_aml.getTableIterator();
      while (amls.hasNext()){
      IRow row_aml=(IRow)amls.next();
      if(row_aml.getValue(ProjectConstants.ATT_AML_MFR_PART_NUMBER).
         equals("MPN_1")){
mapx[0] = new HashMap();
mapx[0].put(ProjectConstants.ATT_AML_AML_SPLIT, "08");
rowArray[0]=row_aml;
rows.put(rowArray[0], mapx[0]);
}
   else
      if(row_aml.getValue(ProjectConstants.ATT_AML_MFR_PART_NUMBER).
         equals("MPN_2")){
      mapx[1] = new HashMap();
      mapx[1].put(ProjectConstants.ATT_AML_AML_SPLIT, "12");
      rowArray[1]=row_aml;
```

```
        rows.put(rowArray[1], mapx[1]);
           }
        }
        ((ITableCell)tbl_aml).updateAMLSplit(rows);
     }
  }
```

Retrieving AML Data from the Item Master

The AML data for a project item is updated in the Item Master. Agile SDK supports retrieving AML data from the Item Master into a project as shown in the following example.

Example 8–17 Retrieving AML Data from item the Master

```
import com.agile.api.*;
import com.agile.util.CMObjectID;
import com.agile.util.CMObjectKey;
public class sampleProjectAMLsync {
   public static void connect(String url, String userName, String passWord)
      throws APIException {
   AgileSessionFactory factory = AgileSessionFactory.getInstance(url);
   Map params = new HashMap();
   params.put(AgileSessionFactory.USERNAME, userName);
   params.put(AgileSessionFactory.PASSWORD, passWord);
   session = factory.createSession(params);
   System.out.println("...Connect to PLM server:" + url);
   }
   public static void disconnect() {
      if (session != null) {
      session.close();
   }
   System.out.println("...Disconnect from PLM server!");
}
   public static void main(String[] args) throws Exception {
      String url = "http://my-pc:8888/web";
      String user = "admin";
      String pwd = "agile";
      connect(url, user, pwd);
      IProject prj = (IProject) session.
         getObject(IProject.OBJECT_TYPE, "PRJ00006");
      ITable prjTableAML=prj.getTable(ProjectConstants.TABLE_AML);
      List itmIdLst=new ArrayList();
      List itmNmbrLst=new ArrayList();
      List itmCMObjIdLst=new ArrayList();
      List itmObjLst=new ArrayList();
   for(Iterator it=prjTableAML.iterator(); it.hasNext();){
      IRow rowAML=(IRow) it.next();
      ITable xx=(ITable)rowAML.getValue(ProjectConstants.ATT_ITEMS_AML);
      String itemNumber=(String)rowAML.
         getValue(ProjectConstants.ATT_ITEMS_NUMBER);
      IItem item = (IItem) session.getObject(ItemConstants.
         CLASS_PART, itemNumber);
      Integer rowid= rowAML.getRowId
      CMObjectID rfrnt= (CMObjectID)rowAML.getReferent().getId();
      Integer itmId=rfrnt.getObjectID();
      itmIdLst.add(itmId);
      itmNmbrLst.add(itemNumber);
      itmCMObjIdLst.add(rfrnt);
      itmObjLst.add(item);
   }
   Map params=new HashMap();
      params.put(ProjectConstants.OPT_DIALOG_UPDATE_AML_ADD_DELETE, true);
      params.put(ProjectConstants.
         OPT_DIALOG_UPDATE_AML_ADD_DELETE_OVERWRITE, true);
params.put(ProjectConstants.OPT_DIALOG_UPDATE_AML_ATTR, false);
params.put(ProjectConstants.OPT_DIALOG_UPDATE_AML_ATTR_OVERWRITE, false);
      params.put(ProjectConstants.ATT_ITEMS_NUMBER, itmCMObjIdLst);
      prj.updateAMLfromIM(params);
      disconnect();
   }
   private static IAgileSession session = null;
}
```

Redlining an AML

Once an item is released, you can change the Manufacturers table only by issuing a new change order. The change order allows you to redline the Manufacturers table.

Note: You can remove redlines from a row of the Manufacturers table. See "Removing Redline Changes" on page 4-25.

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 in the change. SCOs and MCOs do not affect an item's revision.
5. Modify the Redline Manufacturers table.

9 Accessing PLM Metadata with APIName Field

This chapter includes the following:

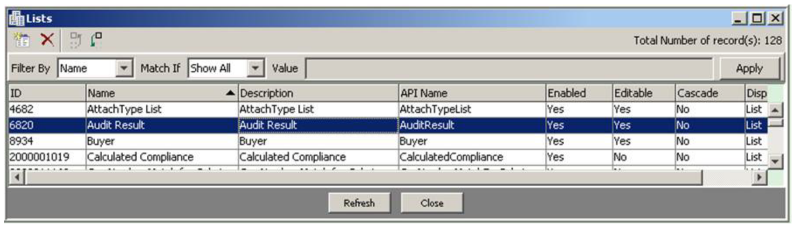
- About the `APIName` Field
- Assigning Names to `APIName` Fields
- `APIName` Validation Rules
- Accessing Metadata Using the `APIName` Field

About the APIName Field

The primary purpose of the `APIName` field is to facilitate SDK developers' access to internal Agile metadata when developing SDK applications. Prior to introduction of this field, display names or numeric IDs defined in the SDK Constants file were used to access Agile internal/metadata of objects in Classes, Tables, Attributes, Lists, and so on. The negative aspect of this approach is that numbers are difficult to remember and display names can change. However, an object's `APIName` is unique, it is easier to remember, and unlike the `DisplayName`, it is not subject to change which can break your code.

For example, your SDK application can use "`AuditResult`" which is the `APIName` of the *List* instead of its ID which is 6820, or its display name which is "Audit Result" to look up its internal value.

Figure 9–1 Accessing attribute value via API Name field



Total Number of record(s): 128

Filter By: Name | Match If: Show All | Value:

ID	Name	Description	API Name	Enabled	Editable	Cascade	Disp
4682	AttachType List	AttachType List	AttachTypeList	Yes	Yes	No	List
6820	Audit Result	Audit Result	AuditResult	Yes	Yes	No	List
8934	Buyer	Buyer	Buyer	Yes	Yes	No	List
2000001019	Calculated Compliance	Calculated Compliance	CalculatedCompliance	Yes	No	No	List

The following paragraphs describe the rules that assign a name to `APIName` fields and SDK interfaces that you can use in your SDK applications to access internal data with the `APIName` field.

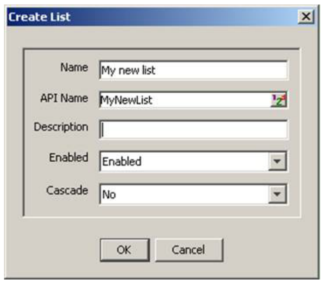
Note: When upgrading to Release 9.3, it is possible to have duplicate `APIName` assigned to previously user defined fields. For example, if you have the user defined field "Foo" on P2 and P3, the upgrade tool will assign the `APIName` Foo to both fields. To avoid these duplications, change the `APIName` for one of these fields in the Java Client.

Assigning Names to APIName Fields

Names are automatically assigned to `APIName` fields by the PLM application when authorized users create new data objects in Java Client. Objects that support `APIName` have the additional API Name field in their *Create* dialog boxes which PLM immediately populates once a name is typed in the object's Name field.

The PLM assigned `APIName` converts the contents of the Name field using the *CamelCase* naming convention. The *CamelCase* convention is adhered to by Java JDK for all core APIs and closely resembles the name of the API. For example, the class "Manufacturer Parts" is converted to "`ManufacturerParts`" and the list "My new list" is converted to "**`MyNewList`**".

Figure 9–2 API name field



Name	My new list
API Name	MyNewList
Description	
Enabled	Enabled
Cascade	No

APIName Validation Rules

The `APIName` naming convention must adhere to the following rules:

- It can contain only characters
- It must be a valid Java/XML identifier

  - Allowed characters are a-z, A-Z, 0-9, and _ (underscore)
- It is between 1 and 255 characters long
- It is case-sensitive
- It must be unique within a context, for example:

  - The attribute "Number" can exist in the cover page table for classes

    Parts" as well as "Changes" (Different context, Parts & Changes)

  - Two attributes with `APIName` "Number" cannot exist in the cover page table of "Parts" (Same context, Parts cover page)

    Two attributes with `APIName` "Number" cannot exist in Cover Page,

    Page Two and Page Three (Cover Page, Page Two and

    Page Three are a single context)

Accessing Metadata Using the APIName Field

You can use the API name of the Agile Metadata to:

- Access the metadata of Agile PLM (Node, Class, Attribute)
- Access/manipulate the value of the metadata (attributes and table attributes) of a data object

You can view the API name of Nodes, Classes, and Attributes in Java Client. SDK interfaces that support the `APIName` field are listed in the following tables.

APIs that Support the APIName Field

API	Example
`IAdmin`	
`getAgileClass(Object id)`	`getAgileClass("Parts")`
`getNode(Object id)`	`getNode ("Part.TitleBlock")`
`IAgileClass`	

`getAttribute (Object key)`	`getAttribute("TitleBlock.number")`
	`or, getAttribute("number")`

`getTableAttributes(Object tableId)`	`getTableAttributes("TitleBlock")`
`getTableDescriptor (Object id)`	`getTableDescriptor("TitleBlock")`
`isSubclassOf(Object cls)`	`isSubclassOf("Parts")`
`IAgileList`	

`getChild(Object child)`	`getChild("UNITED_STATES")`
	`UNITED_STATES is the APIName for entry 'United States' in 'Country' list`

`getChildNode(Object child)`	`getChildNode("UNITED_STATES")`

`setSelection (Object[]` *`childNodes`*`)`	`setSelection(new Object[]`
	`{"UNITED_STATES" , "INDIA"})`

`IAgileSession`	

`createObject(Object objectType, Object params)`	`Map map = new HashMap();`
	`String partNumber = "P00001"`
	`map.put("TitleBlock.number", partNumber);`
	`IDataObject dObj = (IDataObject) (m_session.createObject("Part", map));`

`createObject(int objectType, Object params)`	`Map map = new HashMap();`
	`String partNumber = "P00001"`
	`map.put("number", partNumber);`
	`IDataObject dObj = (IDataObject) (m_session.createObject("Part", map));`

API	Example

`getObject(Object objectType, Object params)`	`Map map = new HashMap();`
	`map.put("TitleBlock.number", "1000-01");`
	`IDataObject dObj = (IDataObject) (m_session.getObject("Part", map));`

`getObject(int objectType, Object params)`	`Map map = new HashMap();`
	`map.put("TitleBlock.number", "1000-01");`
	`IDataObject dObj = (IDataObject)(m_session.getObject(IItem.OBJECT_TYPE, map));`

`IDataObject`	
`getCell(Object key)`	`getCell("PageTwo.list11")` or `getCell("list11")`
`getTable(Object tableId)`	`getTable("AffectedItems")`
`getValue(Object attribute)`	`getValue ("PageTwo.list11")` or `getValue ("list11")`
`setValue(Object key,Object value)`	`setValue("PageTwo.text01","test")`

`saveAs(Object type,Object params)`	`Map params = new HashMap();`
	`params.put("number", number);`
	`IItem part2 = (IItem) part.saveAs("Document", params);`

`setValues(Map map)`	`Map map = new HashMap();`
	`map.put("TitleBlock.number", "1000-01");`
	`part.setValues(map);`

`ILibrary`	
`getAdminList(Object listId)`	`getAdminList("ActionStatus")`

`createAdminList(Map map)`	`map.put(IAdminList.ATT_NAME,"My List");`
	`map.put(IAdminList.ATT_APINAME, "MyList");`
	`map.put(IAdminList.ATT_DESCRIPTION, "My desc");`
	`map.put(IAdminList.ATT_ENABLED, new Boolean(false));`
	`map.put(IAdminList.ATT_CASCADED, new Boolean(false));`
	`IAdmin admin = m_session.getAdminInstance();`
	`IListLibrary listLibrary = admin.getListLibrary();`
	`IAdminList newList = listLibrary.createAdminList(map);`

`INode`	

API	Example

`getChild(Object child)`	`IAdmin admin = m_ session.getAdminInstance();`
	`INode node = admin.getNode(NodeConstants.NODE_ AGILE_CLASSES);`
	`INode partsClass = node.getChild("Parts");`

`getChildNode(Object child)`	`IAdmin admin = m_ session.getAdminInstance();`
	`INode node = admin.getNode("AgileClasses");`
	`INode partsClass = node.getChildNode("Parts");`

`IProgram`	

`saveAs (Object type, Object[] tablesToCopy, Object params)`	`HashMap map = new HashMap();`
	`map.put("name", new_number);`
	`map.put("scheduleStartDate", new Date());`
	`Object[] objects = new Object[]{"PageTwo", "PageThree", "Team"};`
	`IProgram program2 =`
	`(IProgram)program.saveAs("Program", objects, map);`

`saveAs(Object type, Object[]tablesToCopy, Object params, boolean applyToChildren)`	`HashMap map = new HashMap();`
	`map.put("name", new_number);`
	`map.put("scheduleStartDate", new Date());`
	`Object[] objects = new Object[]{"PageTwo", "PageThree", "Team"};`
	`IProgram program2 =`
	`(IProgram)program.saveAs("Program", objects, map, true);`

`IProject`	

`assignSupplier(Object supplierParams)`	`HashMap map = new HashMap();`
	`map.put("Responses.itemNumber", item.getName());`
	`map.put("Responses.supplier", supplier.getName());`
	`rfq.assignSupplier(map);`

`IQuery`	

`setResultAttributes (Object[] attrs)`	`String[] attrs = new String[3];`
	`attrs[0] = "TitleBlock.number";`
	`attrs[1] = "TitleBlock.description";`
	`attrs[2] = "TitleBlock.lifecyclePhase";`
	`query.setResultAttributes(attrs);`

API	Example
`IRequestForQuote`	
`assignSupplier (Object supplierParams)`	`HashMap map = new HashMap();` `map.put("Responses.itemNumber", item.getName());` `map.put("Responses.supplier", supplier.getName());` `rfq.assignSupplier(map);`
`ITable`	
`createRow(Object param)`	`HashMap params = new HashMap();` `params.put("itemNumber", "P0001");` `params.put("newRev", "A");` `ITable affectedItems =` `change.getTable("AffectedItems");` `IRow affectedItemRow =` `affectedItems.createRow(params);`
`createRows(Object[] rows)`	
`getAvailableValues(Object attr)`	`getAvailableValues("PageTwo.list01")`
`updateRows (Map rows)`	`HashMap[] mapx = new HashMap[5];` `Map rows = new HashMap();` `mapx[0] = new HashMap();` `mapx[0].put("newRev", "A");` `mapx[0].put("text01", "sdk test1");` `rows.put(rowArray[0], mapx[0]);` `mapx[1] = new HashMap();` `mapx[1].put("newRev", "A");` `mapx[1].put("text01", "sdk test2");` `rows.put(rowArray[1], mapx[1]);` `tab.updateRows(rows);`
`ITableDesc`	
`getAttribute (Object key)`	`getAttribute("number")`

SDK APIs that Get the APIName Field

Interface	Method
`IAdminList`	`getAPIName()`
`IAgileClass`	`getAPIName()`
`IAgileList`	`getAPIName()` `addChild(Object child, String apiName)`
`IAttribute`	`getAPIName()`
`ICell`	`getAPIName()`
`INode`	`getAPIName()`

IProperty	`getAPIName()`
ITableDesc	`getAPIName()`

API Names of Root Administrator Nodes

The following table lists the API names of the top level Administrator nodes which are not exposed in Agile Java Client. Top Level Admin Nodes are Admin Nodes that exist on their own. That is, no other Admin Node must exist in order for these Admin nodes to exist. For example, Class and Roles are top level nodes, but Life Cycle Phases and Attributes are not because they belong to another Admin Node. Similarly, Workflow Statuses are not top level nodes because they belong to Workflow.

Root Node	**API Name**
ACS Responses	`ACSResponses`
Account Policy	`AccountPolicy`
Activity Statuses	`ActivityStatuses`
ActivityHealths	`ActivityHealths`
Agile Classes	`AgileClasses`
Agile Workflows	`AgileWorkflows`
Agile eHubs	`AgileEHubs`
Attachment Purge Setting	`AttachmentPurgeSetting`
AutoNumbers	`AutoNumbers`
Catchers	`Catchers`
Character Set	`CharacterSet`
Cluster	`Cluster`
Company Profile	`CompanyProfile`
Criteria Library	`CriteriaLibrary`
Dashboard Management	`DashboardManagement`
Default Role Settings	`DefaultRoleSettings`
Destinations	`Destinations`
Event Handler Types	`EventHandlerTypes`
Event Handlers	`EventHandlers`
Event Subscribers	`EventSubscribers`
Event Types	`EventTypes`
Events	`Events`
Example Role/Privilege	`ExampleRolePrivilege`
Filters	`Filters`
Full Text Search Settings	`FullTextSearchSettings`
Import Preference Setting	`ImportPreferenceSetting`
LDAPConfig	`LDAPConfig`
LifeCycle Phases	`LifeCyclePhases`
My Assignments	`MyAssignments`

Notification Templates	`NotificationTemplates`
PGC SmartRules	`PGCSmartRules`
Package File Types	`PackageFileTypes`
Portals	`Portals`
Preferences	`Preferences`
Privileges	`Privileges`
Process eXtension Library	`ProcessEXtensionLibrary`
Query Cleanup	`QueryCleanup`
RFQ Terms and Conditions	`RFQTermsAndConditions`
Reports	`Reports`
Roles	`Roles`
Server Location	`ServerLocation`
Sign Off Message	`SignOffMessage`
SmartRules	`SmartRules`
Subscribers	`Subscribers`
Task Configuration	`TaskConfiguration`
UOM Families	`UOMFamilies`
Viewer and Files	`ViewerAndFiles`
wCM Servers	`WCMServers`

API Name Examples

The following example shows how to log in to an Agile PLM server, create two parts, enable Page Two text 01 and List 20, set values for them, and then add the second part to the BOM Table of the first part.

Example 9–1 Using the APIName field to access metadata

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

/**
 * This sample code shows how to use the API name.
 * It uses some of the SDK APIs with the API name.
 * For a list of API names for attributes and classes,
 * refer to Agile Java Client.
 * Some API names in Agile Java Client may differ from the ones
 * in this example. This is because a duplicate conflict
 * was detected in the API name in the same context.
 * If you detect this conflict, be sure to change the API name
 * in this sample before compiling and executing the code.
*/

public class APIName
{
   public static final String USERNAME = "admin";
   public static final String PASSWORD = "agile";
   public static final String URL = "http://localhost:<>/Agile";
   public static IAgileSession session = null;
   public static IAdmin admin = null;
   public static AgileSessionFactory factory = null;
   public static IListLibrary listLibrary = null;
/**
*   @param args
*/
   public static void main(String[] args) {
      try {
         // Create an IAgileSession instance
         session = connect(session);
         admin = session.getAdminInstance();
         listLibrary = admin.getListLibrary();

         // Create two parts
         IItem itemParent = createItem(getAutonumber());
         IItem itemChild = createItem(getAutonumber());

         // enable Page Two tab for Part
         enableP2();

         // enable Page Two Text 01 and set value
         setP2Text(itemParent);

         // create a new AdminList
         createAdminList();

// enable Page Two List 20 and set the value.
         setP2List(itemParent);

//       Add the child part to the BOM table of the parent part
         addBOM(itemParent, itemChild);
   }
   catch (Exception e) {
      e.printStackTrace();
   }
   finally {
         session.close();
      }
   }
   /**
*       @throws APIException
*/
   private static void createAdminList() throws APIException {
      Map listParams = new HashMap();
      listParams.put(IAdminList.ATT_APINAME, "MY_LIST");
      // Specify the API name of the List
      listParams.put(IAdminList.ATT_NAME, "My List");
      listParams.put(IAdminList.ATT_ENABLED, new Boolean(true));
      IAdminList myList = listLibrary.createAdminList(listParams);
      IAgileList values = myList.getValues();
      values.addChild("Value A", "VAL_A");
      // Specify the API name along with the value
      values.addChild("Value B", "VAL_B");
      values.addChild("Value C", "VAL_C");
      myList.setValues(values);
      System.out.println("Created Admin List " + myList.getName());
   }
/**
      * @throws APIException
*/
```

```
   private static void enableP2() throws APIException {
      INode p2 = admin.getNode("Part.PageTwo"); // Fully qualified API name
      IProperty visible = p2.getProperty(PropertyConstants.PROP_VISIBLE);
      IAgileList values = visible.getAvailableValues();
      values.setSelection(new Object[]{ "Yes" });
      visible.setValue(values);
      System.out.println("Page two enabled for Part class");
   }
/**
      * @param itemParent
      * @throws APIException
*/
   private static void setP2Text(IItem itemParent) throws APIException {
      IAgileClass clazz = itemParent.getAgileClass();
      ITableDesc p2TableDesc = clazz.getTableDescriptor("PageTwo");

      // 'PageTwo' is the API name of the Page Two tab
      IAttribute text01 = p2TableDesc.getAttribute("text01");
      // 'text01' is the API name of the Text01 field
      IProperty visible = text01.getProperty(PropertyConstants.PROP_VISIBLE);
      IAgileList values = visible.getAvailableValues();
      values.setSelection(new Object[]{ "Yes" });
      visible.setValue(values);
      itemParent.setValue("PageTwo.text01", "SDK test");

      //'PageTwo.text01' is the fully qualified APIName for ItemConstants.
      // ATT_PAGE_TWO_TEXT01
      System.out.println("Set P2 Text01 " + itemParent.
         getValue("PageTwo.text01") + " for Part " + itemParent.getName());
   }
/**
      * @param itemParent
      * @throws APIException
*/
private static void setP2List(IItem itemParent) throws APIException {
IAgileClass partsClass = itemParent.getAgileClass();
//'PageTwo' is the API name of the Page Two tab
ITableDesc p2TableDesc = partsClass.getTableDescriptor("PageTwo");
//'list20' is the API name of the List20 field
IAttribute list20 = p2TableDesc.getAttribute("list20");
//Enable list20 attribute
IProperty visible = list20.getProperty(PropertyConstants.PROP_VISIBLE);
IAgileList values = visible.getAvailableValues();
values.setSelection(new Object[]{ "Yes" });
visible.setValue(values);
//Assign My List to list20 attribute
IAdminList myList = listLibrary.getAdminList("MY_LIST");
IProperty assignMyList = list20.getProperty(PropertyConstants.PROP_LIST);
IAgileList availableLists = assignMyList.getAvailableValues();
availableLists.setSelection(new Object[]{myList});
assignMyList.setValue(availableLists);
//Select one of the list20 values for the parent item
IAgileList listValues = myList.getValues();
//VAL_B is the API name of the list value 'Value B'
listValues.setSelection(new Object[] {"VAL_B"} );
itemParent.setValue("PageTwo.list20", listValues);
System.out.println("Set P2 List20 " + listValues.getSelection()[0].getValue()
+ " for Part " + itemParent.getName());
}
```

```
/**
   * <p> Create an IAgileSession instance </p>
   * @param session
   * @return IAgileSession
   * @throws APIException
*/

private static IItem createItem(String number) throws APIException {
   HashMap map = new HashMap();
   map.put("TitleBlock.number", number);
// 'number' or 'TitleBlock.number' is the APIName for ItemConstants.
//ATT_TITLE_BLOCK_NUMBER
   map.put("description", "test");
// 'description' or 'TitleBlock.description' is the APIName for
//ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION
   IItem item = (IItem)session.createObject("Part",map);
// 'Part' is the API name of the Part class
    System.out.println("Created Part " + number);
return item;
}

private static IItem createItem(String number) throws APIException {
   HashMap map = new HashMap();
   map.put("TitleBlock.number", number);

// 'number' or 'TitleBlock.number' is the APIName for ItemConstants.
//ATT_TITLE_BLOCK_NUMBER
   map.put("description", "test");

// 'description' or 'TitleBlock.description' is the APIName for
//ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION
   IItem item = (IItem)session.createObject("Part",map);

// 'Part' is the API name of the Part class
   System.out.println("Created Part " + number);
   return item;
}

   private static IAgileSession connect(IAgileSession session)
         throws APIException {
      factory = AgileSessionFactory.getInstance(URL);
      HashMap params = new HashMap();
      params.put(AgileSessionFactory.USERNAME, USERNAME);
      params.put(AgileSessionFactory.PASSWORD, PASSWORD);
      session = factory.createSession(params);
      return session;
   }
/**
   * <p> Create a part </p>
   * @param parent
   * @return IItem
   * @throws APIException
*/
private static IItem createItem(String number) throws APIException {
   HashMap map = new HashMap();
   map.put("TitleBlock.number", number);

// 'number' or 'TitleBlock.number' is the APIName for ItemConstants.
//ATT_TITLE_BLOCK_NUMBER
  map.put("description", "test");
// 'description' or 'TitleBlock.description' is the APIName for
//ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION
   IItem item = (IItem)session.createObject("Part",map);

// 'Part' is the API name of the Part class
   System.out.println("Created Part " + number);
  return item;
}
/**
   * <p> Add the child parts to the BOM table of the parent part </p>
   * @param itemParent
   * @param itemChild1
   * @param itemChild2
   * @return ITable
   * @throws APIException
*/
private static ITable addBOM(IItem itemParent, IItem itemChild) throws
APIException {

// 'BOM' is APIName for ItemConstants.TABLE_BOM
   ITable table = itemParent.getTable("BOM");

// Add child item to BOM of parent item
   table.createRow(itemChild);
   System.out.println("Added Part " + itemChild.getName() +
            "to BOM of the Part " + itemParent.getName());
return table;
}
/**
   * @return
   * @throws APIException
*/
   private static String getAutonumber() throws APIException{
      IAgileClass cls = admin.getAgileClass("Part");

      // 'Part' is the API name of the Part class
      IAutoNumber auto[] = cls.getAutoNumberSources();
      String number = null;
      if(auto != null && auto.length > 0)
      number = auto[0].getNextNumber();
   else
      number = "PART" + System.currentTimeMillis();
      return number;
   }
}
```

10

Subscribing to Agile PLM Objects

This chapter includes the following:

- About User Subscriptions
- Getting Subscriptions for an Object
- Modifying Subscriptions for an Object
- Making Attributes Available for Subscription
- Working with Subscription Tables

About User Subscriptions

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

You can subscribe to both routable and nonroutable objects. Agile APIs provides an interface called ISubscribable, which enables retrieving and modifying all subscriptions for an object. All objects that a user has subscribed to are listed on the user's Subscription table.

Subscription Events

Subscription events support all related subscription event File Actions listed in the table below.

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`
Get file	`EVENT_GET_FILE`
Open file	`EVENT_OPEN_FILE`
View file	`EVENT_VIEW_FILE`
Delete File	`EVENT_DELETE_FILE`

Subscription Event	SubscriptionConstants
Checkin File	`EVENT_CHECKIN_FILE`
Checkout File	`EVENT_CHECKOUT_FILE`
Cancel Checkout File	`EVENT_CANCELCHECKOUT_FILE`

Note: There are additional subscription events for Projects Execution objects that are not supported by the Agile API.

Although most routable and nonroutable objects support the subscription events listed in the table above, there are some exceptions:

- User objects do not support the Lifecycle Change subscription event.
- File Folder objects do not support the Add File and Cancel Checkout File subscription events.

The Field Change subscription event is related to any attribute whose Available To Subscribe property has been set to "Yes." Consequently, each class and subclass can have a different set of subscribable attributes.

Subscribe Privilege

To subscribe to an object, you must have the Subscribe privilege for that class. Many predefined Agile PLM roles, such as Creator, already have the Subscribe privilege for several object classes. To change your roles and privileges, see your Agile PLM system administrator.

Subscription Notifications

Subscription events trigger two types of Agile PLM Notifications:

- **Email** - Email Notifications are sent only if the Receive Email Notification preference of the recipient is set to **Yes**. For information on user and system preferences, refer to *Agile PLM Administrator Guide*.
- **Inbox** - Inbox Notifications occur automatically regardless of user preferences

A user with Administrator privileges can create and configure these Notifications in Java Client which provides two very similar dialogs for this purpose. The reason for the two dialogs is due to the fact that there are two sets of Email and Inbox Notifications:

- Those that the "To" field is grayed out
- Those that the administrator user can select recipients who are notified when the subscription event is triggered

Sending Notifications with SDK

Notifications are briefly described in the SDK Guide under Event Notifications. SDK exposes the following API to send Notifications to designated notifiers with the specified template for Agile PLM objects. For information about Notifications, refer to *Agile PLM Administrator Guide*.

sendNotification	(IDataObject)	*object*	Object the Notification is issued for
	String	template	Name of Notification template
	Collection	notifiers	List of notifiers
	boolean	urgent	True for urgent Comments
	String	comments	about the Notification
	)		throws APIException

These parameters are defined as follows:

- **object** - object on which Notification is to be issued
- **template** - name of the Notification template
- **notifiers** - collection containing a list of users as individual `IDataObjects` such as `IUser` and `IUserGroup`
- **urgent** - value of true indicates send urgently, set to false otherwise
- **comments** - comments about the Notification

For more information about these parameters and the API, refer to Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in `SDK_samples` (ZIP file). To access this file, see the **Note** in "Client-Side Components" on page 1-2.

The following example uses `sendNotification` to send a Notification from ECO C0001 to user1 and user2 with Notification Comment and template

Example 10–1 Sending a Notification with Agile SDK

```
List notifyList = new ArrayList();
IUser user =
               (IUser) session.getObject(IUser.OBJECT_TYPE, "johndoe");
                notifyList.add(user);
    user =
                (IUser) session.getObject(IUser.OBJECT_TYPE, "janedoe");
                 notifyList.add(user);
IChange agileObject =
               (IChange)session.getObject(IChange.OBJECT_TYPE, "C00001");
               boolean urgent = true;
String comment =
               "Add ECO approver, Notify CA";
String template =
               "Custom Notification SDK Example";
session.sendNotification(agileObject,template,notifyList,urgent,comment);
```

Deleting Subscribed Objects

You can delete any Agile PLM business object using the `IDataObject.delete()` method. However, you can't delete an object until its subscriptions are removed. Users can remove their own subscriptions, but not the subscriptions of other users.

Getting Subscriptions for an Object

To retrieve the current subscriptions for an object, use `ISubscribable.getSubscriptions()`, which returns an array of all `ISubscription` objects, both enabled and disabled. The following example shows how to get subscriptions for an object.

Example 10–2 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_GET_FILE)) {
if (subs[i].isEnabled()) {
System.out.println("Get File subscription is enabled");
}
else if (subs[i].getId().equals(SubscriptionConstants.EVENT_VIEW_FILE)) {
if (subs[i].isEnabled()) {
System.out.println("View File subscription is enabled");
}
else if (subs[i].getId().equals(SubscriptionConstants.EVENT_OPEN_FILE)) {
if (subs[i].isEnabled()) {
System.out.println("Open 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 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 "Making Attributes Available for Subscription" on page 10-6.

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 10–3   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 P
   * 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 10–4 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);

for (int i = 0; i < attr.length; ++i) {
   IProperty prop = null;
   IAgileList listValues = null;
   String strVal = "";

// Check if the attribute is visible
   prop = attr[i].getProperty(PropertyConstants.PROP_VISIBLE);
   listValues = (IAgileList)prop.getValue();
   strVal = listValues.toString();
// If the attribute is visible, make it subscribable
   if (strVal.equals("Yes")) {
      prop =
         attr[i].getProperty(PropertyConstants.PROP_AVAILABLE_FOR_SUBSCRIBE);
         if (prop != null) {
           listValues = prop.getAvailableValues();
           listValues.setSelection(new Object[] { "Yes" });
           prop.setValue(listValues);
         }
      }
   }
} catch (APIException ex) {
      System.out.println(ex);
}
```

Parent and Child Attributes

Several read-only attributes have a child relationship with a parent attribute. Child attributes derive values from their parent attribute. Consequently, parent attributes are available for subscription, but child attributes are not. Examples of child attributes include BOM table attributes like `BOM.Item List02` and `BOM.Item Text01`.

Working with Subscription Tables

A user's Subscription table lists all subscriptions the user has made. The Subscription table offers limited editing capabilities. For example, you can't add new rows to the table; the only way to add subscriptions using the Agile API is to call `ISubscribable.modifySubscriptions()` for a dataobject. However, you can remove subscriptions from the table.

The following example shows how to retrieve the Subscription table for the current user. It also shows how to remove a subscription for a part with the number 1000-02.

Example 10–5 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.getReferentIterator();

// Get each object referenced in the table
   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 10–6 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 10–7 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);
}
```

11

Managing Manufacturing Sites

This chapter includes the following:

- 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 of an Item
- Selecting the Current Manufacturing Site of an Item

About Manufacturing Sites

Organizations that practice distributed manufacturing use several different manufacturing sites for their products. Agile PLM site objects allow companies to maintain site-specific information for a product's parts. For example, the various manufacturing locations may have different effectivity dates for new revisions, different manufacturing instructions due to location, or different manufacturers from whom they buy components, due to location.

Changes can affect all manufacturing sites of an item or a specific site. The Affected Items table for a change lets you select the manufacturing sites that are affected. Items may have different effectivity dates and dispositions at each site. You specify effectivity dates and dispositions on the Affected Items tab of an ECO or SCO. To create a new revision when you assign the new effectivity date or disposition, use an ECO. To assign site-specific effectivity dates and dispositions without incrementing the revision, use an SCO.

For a more detailed overview of Agile PLM's manufacturing sites functionality, refer to the *Agile PLM Product Collaboration Guide*.

Controlling Access to Sites

In order to use manufacturing sites in your organization, the Sites module must be enabled in Agile Administrator. Your organization's agreement with Oracle determines the modules that are enabled in Agile PLM.

Access to manufacturing sites of users is controlled by their assigned roles and privileges and the Sites property in their profiles. Your organization can create an unlimited number of manufacturing sites, however a user will not have access to every site unless all sites are specified in his user profile Sites property. Your organization may have implemented the Agile PLM system in such a way that users can access only the information pertaining to certain sites, as determined by their user profile Sites property.

To create a site-specific BOM for an item, the item's subclass must have the Site-specific BOM property set to **Allow**. Otherwise, items of that subclass have BOMs that are common to all sites. For information on Sites and enabling sites, refer to the *Agile PLM Administrator Guide*.

Creating a Manufacturing Site

Manufacturing sites are identified uniquely by name. To create a manufacturing site, use the IAgileSession.createObject method, specifying both the class and the site name.

All users cannot create manufacturing sites. Only users who have the Create privilege for manufacturing site objects can create manufacturing sites.

Note: When you create a manufacturing site, its Lifecycle Phase is set to Disabled by default. To use the site, make sure you enable it.

Example 11–1 Creating and enabling a manufacturing site

```
try {
// Create a manufacturing site
   IManufacturingSite mfrSite = (IManufacturingSite)m_session.createObject(
     ManufacturingSiteConstants.CLASS_SITE, "Taipei");

// 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 11–2 Loading a manufacturing site

```
try {
// Load the Hong Kong site  using IManufacturingSite
   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 using APIName
   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 11–3   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. You can also iterate over the Sites table to get the same information, but using the `getManufacturingSites()` method is easier and faster. For an example that uses this method, see "Selecting the Current Manufacturing Site of an Item" on page 11-4.

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 11–4   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 of 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 `IManufacturingSiteSelectable.getManufacturingSite()`.

Example 11–5 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 11–6 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 an item has a specific manufacturing site, it is not site-specific, or it has All Sites. To this end, do as follows:

- 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 z 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 11–7 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 site to Hong Kong
   item.setManufacturingSite(siteHK);
   String desc = (String)item.getValue(ItemConstants.ATT_TITLE_BLOCK_DESCRIPTION);
   System.out.println("Hong Kong description = " + desc);

// Set the site to San Francisco
   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].getValue());
   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 11–8 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);
}
```

12

Managing Reference Objects

This chapter includes the following:

- About Reference objects
- Creating a Reference Object
- Updating a Reference Object
- Deleting a Reference Object
- Adding a Reference Object to the Relationships Tab of the Web Client
- Removing a Reference Object
- Searching for Reference Objects

About Reference objects

Reference Objects are an Oracle Agile PLM object class. They enable PLM users to look up objects in external applications and add them to the Relationships tab of the Web Client as references. Reference objects are only supported in the Web Client. However, once a reference object is added to a Relationships tab in the Web Client, that reference object is also visible in the Java Client.The Agile administrator must create the Discover, Read, Create, Modify, and Deleted privilege masks for Relationship Objects, and apply these privilege masks to roles. Reference Objects appear in the Add drop-down menu on the Relationships tab when the end user has a Create privilege mask for Reference Objects.

- You can perform a simple search for Reference Objects only from the Relationships (or Content) tab of the Web Client. None of the other Agile PLM searches support searching for Reference Objects or their subclasses.
- The set of search result objects is determined by the external application.
- From the Relationships tab in Web Client, an end user can do the following interactively, and in the batch mode using the exposed Agile APIs:
  - PLM's Relationships table.
  - Create a new object in an external application and add a reference to that object in Agile PLM's Relationships table.
  - Search, add, and remove Reference Objects from the Agile PLM Relationships table.
- PLM Clients support an absolute URL and the SDK supports the absolute and relative URLs. For example, `/jde/servlet/1234` (relative URL) and `https://servlet:port/jde/servlet/1234` (absolute URL).

- The Relative and Absolute URL types support the following three SDK calls:
  - `getURL` (relative URL)
  - `setURL` (relative URL)
  - `getAbsoluteURL`
- You can:
  - Search for an object in an external application and add a reference to that object in Agile PLM Relationships table.
  - Search, add, and remove Reference Objects from the Agile PLM Relationships table.
  - Create a new object in an external application and add a reference to that object in the Agile PLM Relationships table.
- Advanced search does not support case sensitive or other Agile search FTS features. This limitation on the server side is due to the reference object query results are from the external system and our reference object WS schema does not have more search syntax definitions. So Agile core cannot guarantee the external search criteria.
- In Advanced Search:
  - The only supported search criteria column is `TitleBlock.Name/Number/Desc`.
  - The only supported search operator is "Contains." For example, `query.setCriteria("[TitleBlock.Name/Number/Desc] contains the phrase` '*'"); If the SDK client passes other criteria columns or operators, the Search server will throw an error stating the search condition syntax was invalid and use only one row with criteria column

Creating a Reference Object

The Reference Objects class is an `ObjectType` on a Remote or External Application that is configured or mapped to an Agile object class and accessed through the Relationships Tab of the Web Client.

To create Reference object use `IAgileSession.createObject(Object, Object).` Remember that the following required attributes must be provided at the creation time

Required Attribute	Potential Argument Types
`ATT_GENERAL_INFO_OBJECT_TYPE`	`ReferenceObject SubClass (String), IAgileClass` object
`ATT_GENERAL_INFO_NUMBER`	`ReferenceObject Number/Name (String)`
`ATT_GENERAL_INFO_PUBLIC_KEY_STRING`	`ReferenceObject Public Key (String)` This can serve as the composite key that uniquely identifies the Object on the Remote Application.

Note: A `Key(ID)` that can uniquely identify the Reference Object on the Remote system is called *Public Key*. This is different from the security-related Public/Private Key definitions.

This example shows how to create a Reference Object.

Example 12–1 Creating a Reference Object

```
Map<Integer, String> paramsMap;

//The only allowed Sub Class is Name/APIName of the Reference Object from Agile
//Java Client
   paramsMap.put(ReferenceObjectConstants.ATT_GENERAL_INFO_OBJECT_TYPE,
      referenceObjectSubClass); //Note: Should be a valid Sub Class
   paramsMap.put(ReferenceObjectConstants.ATT_GENERAL_INFO_NUMBER, "REFOBJ01");
   paramsMap.put(ReferenceObjectConstants.ATT_GENERAL_INFO_PUBLIC_KEY_STRING,
      "UniqueKeyOnRemoteApplication");
   IReferenceObject refObject = (IReferenceObject)m_session.createObject(
      IReferenceObject.OBJECT_TYPE,paramsMap);
```

Note: Web Client users can directly create a Reference Object. They cannot do this by using search, and then adding the search result, or the object in the remote system. When the required parameters are known, SDK also supports the direct creation of Reference Objects as in other Agile object types. This is designed to align Reference Objects with other object types in the SDK framework.

Updating a Reference Object

To update a Reference Object, first get the pointer to the existing Reference Object using the `IAgileSession.getObject(IReferenceObject.OBJECT_TYPE, paramsMap)` API. Once the pointer to the object is available, updating the attributes is similar to other object types in the Agile system. You can use the `getObject()` API with either Subclass and Name, or Subclass and Public Key to uniquely identify the Object.

Note: The `getObject()` API retrieves a Reference Object if and only if the object exists in the Agile system, otherwise, it will return a null value. As such, `getObject()` also determine if the object already is/is not in the Agile PLM. Because an object number can be repeated across different subclasses, for example, if part, manufacturer part, and problem report, all have the number 'P0001', it is necessary to pass the `subclassID` in order to return a unique Object ID. To this end, the `getObject()`API passes both the `subclassID` and the Object number.

Example 12–2 Updating a Reference Object

```
Map<Integer, String> paramsMap;
paramsMap.put(ReferenceObjectConstants.ATT_GENERAL_INFO_OBJECT_TYPE,
   referenceObjectSubClass); //Note: Must be a valid Sub Class
      paramsMap.put(ReferenceObjectConstants.ATT_GENERAL_INFO_NUMBER, "REFOBJ01");

paramsMap.put(ReferenceObjectConstants.ATT_GENERAL_INFO_PUBLIC_KEY_STRING,
   "UniqueKeyOnRemoteApplication");

//Get/Load Reference Object from Agile if already exist.
   IReferenceObject refObject = (IReferenceObject)m_session.getObject(
      IReferenceObject.OBJECT_TYPE, paramsMap);

//Get the cell of each attribute:
   ICell num =
      refObject.getCell(ReferenceObjectConstants.ATT_GENERAL_INFO_NUMBER);

   ICell desc =
      refObject.getCell(ReferenceObjectConstants.ATT_GENERAL_INFO_DESCRIPTION);
   ICell url =
      refObject.getCell(ReferenceObjectConstants.ATT_GENERAL_INFO_URL);
   ICell sts =
      refObject.getCell(ReferenceObjectConstants.ATT_GENERAL_INFO_STATUS);

//Update the cells with new values:
   num.setValue("New Number");
   url.setValue("Modified URL");
   desc.setValue("New Description");
   sts.setValue("Pending");

//The same above is also achieved using the following approach:
   refObject.setValue(ReferenceObjectConstants.
      ATT_GENERAL_INFO_NUMBER, "New Number");
   refObject.setValue(ReferenceObjectConstants.
      ATT_GENERAL_INFO_DESCRIPTION, "Modified URL");
   refObject.setValue(ReferenceObjectConstants.
      ATT_GENERAL_INFO_URL, "New Description");
   refObject.setValue(ReferenceObjectConstants.
      ATT_GENERAL_INFO_STATUS, "Pending");
```

Deleting a Reference Object

You can only delete the Reference Object, provided it exists, from the local Agile system and not from the Remote Agile system. This is similar to deleting other object types in the Agile PLM.

Adding a Reference Object to the Relationships Tab of the Web Client

Adding Reference Objects to the Relationships table of the Web Client is similar to adding other object types in the Agile system using the `createRow()` API. This is illustrated in Example 12–3.

Example 12–3 Adding a Reference Object to the Relationships Table in Web Client

```
IAdmin m_admin = m_session.getAdminInstance();
paramsMap.put(ReferenceObjectConstants.
   ATT_GENERAL_INFO_OBJECT_TYPE, referenceObjectSubClass);
paramsMap.put(ReferenceObjectConstants.
   ATT_GENERAL_INFO_NUMBER, "REFOBJ01");
paramsMap.put(ReferenceObjectConstants.
   ATT_GENERAL_INFO_PUBLIC_KEY_STRING, "UniqueKeyOnRemoteApplication");

//Get the Reference Object from Agile
   IReferenceObject refObject =
      (IReferenceObjectm_session.getObject(
         IReferenceObject.OBJECT_TYPE,paramsMap);

//Get Relationship Table of an Object that the Reference Object will be added to:
   IDataObject parentObj = (IDataObject) m_
      session.getObject(QualityChangeRequestConstants.CLASS_CAPA, "CAPA0001");
   ITable relTbl = parentObj.
      getTable(QualityChangeRequestConstants.TABLE_RELATIONSHIPS);
```

```
//Add the Reference Object with the createRow()
   API HashMap map = new HashMap();
   map.put(CommonConstants.ATT_RELATIONSHIPS_NAME, refObject);
   IRow targetRow = relTbl.createRow(map);
```

Removing Reference ObjectS from Relationships Tab of the Web Client

To Delete or remove a Reference Object from the Relationships table in the Web Client, use the removeRow(IRow row) API. This is similar to deleting other object types in Agile PLM tables.

Searching for Reference Objects

Use the `getCell (ReferenceObject .....)` API to search for Reference Objects. This is shown in the following example.

Example 12–4 Searching for Reference Objects

```
ITable searchResult =
      doSearch("RemoteItems", "P0000");
ArrayList objects =
      doCreate("RemoteItems", searchResult);

// Search the external system and retrieve the referenced objects in Tablular
// format:
private static ITable doSearch(String subClassName, String srchStr) throws
        APIException {
ITable results = null;
try {
    String qry =
          "["+ReferenceObjectConstants.SEARCH_KEY+"]"+ " contains '"+srchStr+"'";
    IQuery query =
          (IQuery)session.createObject(IQuery.OBJECT_TYPE, RefObjSubClassName);
    query.setCaseSensitive(false);
    query.setCriteria(qry);
    results = query.execute();
}
    catch (APIException ex) {
    System.out.println(ex);
}
return results;
}
//Create the Reference Objects retrieved by IRows in the above search:
private static ArrayList doCreate(String subClassName, ITable qryResult) throws
APIException{
ArrayList result = new ArrayList();
IDataObject refObj = null;
try{
Iterator itr = qryResult.getTableIterator();
while(itr.hasNext()){
IRowReferenceObjectWebServiceSearch row = (IRowReferenceObjectWebServiceSearch)
itr.next();
String name = row.getValue(ReferenceObjectConstants.ATT_GENERAL_INFO_
NUMBER).toString();
String pubKey = row.getPublicKey();
String description =
   row.getValue(ReferenceObjectConstants.ATT_GENERAL_INFO_DESCRIPTION).toString();
String url = row.getUrl();
Map params = createParams(subClassName, name, pubKey, description, url);
refObj = (IReferenceObject) session.getObject(ReferenceObjectConstants.CLASS_
REFERENCE_OBJECT_CLASS, params);
if (refObj==null)
refObj = (IReferenceObject) session.createObject(subClassName, params);
result.add(refObj);
}
} catch(APIException ex) {
throw ex;
}
return result;
}
```

13 Working with Lists

This chapter includes the following:

- About Lists
- Selecting a List Value
- Setting the Maximum Value Displayed by a List
- Creating Custom Lists
- Checking the Data Type of a List
- Renaming and Removing List Values
- Printing Contents of IAgileList Objects

About Lists

Many attributes in the Agile PLM system are configured as lists. Agile provides two datatypes to support list fields:

- `SingleList` - a list in which only one value can be selected.
- `MultiList` - a list in which multiple values can be selected.

Attributes, properties, and cells can all be lists. The Agile API provides methods for working with lists in the `IAgileList` interface, a generalized data structure used for all Agile lists. Because `IAgileList` represents a tree structure of available list values, it extends the `ITreeNode` interface.

You can use `ITreeNode.addChild()` to add values to a list. All list values must be unique. After adding a list value, you can prevent its selection by making it obsolete.

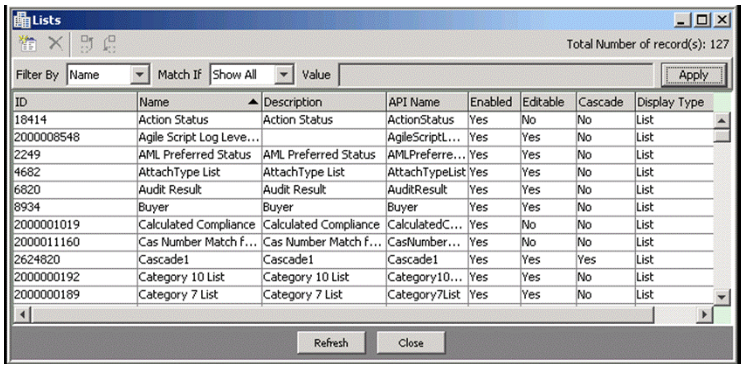
List Library

In Agile Java Client, administrators can define custom lists that can be used for Page Two and Page Three list attributes. You can also use the Agile API to define custom lists. The `IListLibrary` interface provides functionality equivalent to the list library in Agile Java Client. You can use the `IAdminList` interface to modify the values or properties of a list.

To retrieve the list library, use the `IAdmin.getListLibrary()` method. You can then use the `IAdminList` interface to create new custom lists and work with existing lists. `AdminListConstants` provide IDs for each list in the list library.

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

Figure 13–1 List Library



Lists

Total Number of record(s): 127

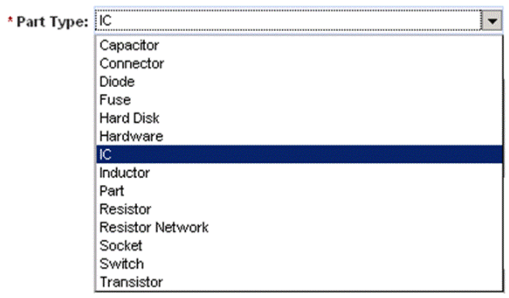
Filter By: Name   Match If: Show All   Value:

ID	Name	Description	API Name	Enabled	Editable	Cascade	Display Type
18414	Action Status	Action Status	ActionStatus	Yes	No	No	List
2000008548	Agile Script Log Leve...		AgileScriptL...	Yes	Yes	No	List
2249	AML Preferred Status	AML Preferred Status	AMLPreferre...	Yes	Yes	No	List
4682	AttachType List	AttachType List	AttachTypeList	Yes	Yes	No	List
6820	Audit Result	Audit Result	AuditResult	Yes	Yes	No	List
8934	Buyer	Buyer	Buyer	Yes	Yes	No	List
2000001019	Calculated Compliance	Calculated Compliance	CalculatedC...	Yes	No	No	List
2000011160	Cas Number Match f...	Cas Number Match f...	CasNumber...	Yes	No	No	List
2624820	Cascade1	Cascade1	Cascade1	Yes	Yes	Yes	List
2000000192	Category 10 List	Category 10 List	Category10...	Yes	Yes	No	List
2000000189	Category 7 List	Category 7 List	Category7List	Yes	Yes	No	List

SingleList Lists

A `SingleList` attribute or cell presents a list from which only one value can be selected. The following figure shows a `SingleList` cell for Part Types in Agile Web Client.

Figure 13–2 SingleList cell in the Agile Web Client



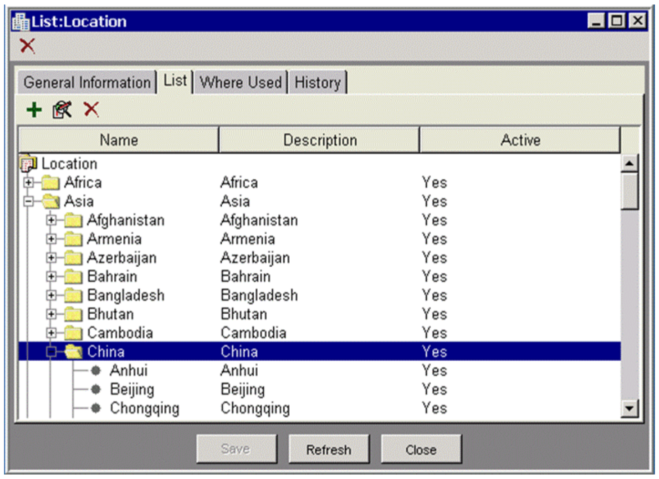
* Part Type: IC

Capacitor
Connector
Diode
Fuse
Hard Disk
Hardware
IC
Inductor
Part
Resistor
Resistor Network
Socket
Switch
Transistor

Cascading Lists

In Agile Java Client, you can configure a `SingleList` attribute to have multiple hierarchical levels. A list with multiple hierarchical levels is called a cascading list. The following figure shows the Location list, a cascading list, being configured in Agile Java Client. The list has separate levels for continent, country, and city.

Figure 13–3 Configuring a cascading list in the Agile Java Client



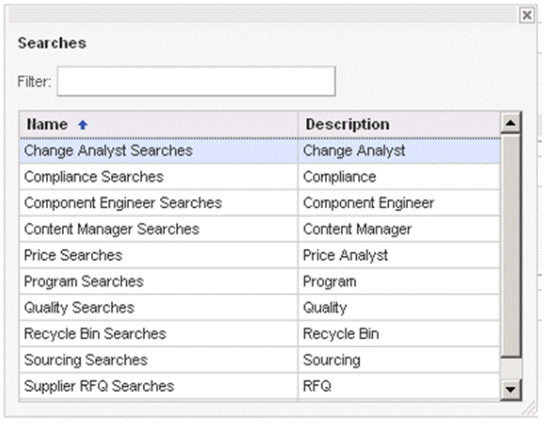
Name	Description	Active
Location		
Africa	Africa	Yes
Asia	Asia	Yes
Afghanistan	Afghanistan	Yes
Armenia	Armenia	Yes
Azerbaijan	Azerbaijan	Yes
Bahrain	Bahrain	Yes
Bangladesh	Bangladesh	Yes
Bhutan	Bhutan	Yes
Cambodia	Cambodia	Yes
China	China	Yes
Anhui	Anhui	Yes
Beijing	Beijing	Yes
Chongqing	Chongqing	Yes

Note: The Location list is the only cascading list that ships with Agile PLM. However, you can define your own cascading lists.

MultiList Lists

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

Figure 13–4 Multiple Value Selection window in the Agile Web Client



Name	Description
Change Analyst Searches	Change Analyst
Compliance Searches	Compliance
Component Engineer Searches	Component Engineer
Content Manager Searches	Content Manager
Price Searches	Price Analyst
Program Searches	Program
Quality Searches	Quality
Recycle Bin Searches	Recycle Bin
Sourcing Searches	Sourcing
Supplier RFQ Searches	RFQ

Methods that Use IAgileList

The `IAgileList` interface provides the necessary methods to get and set the selected value(s) of a list. The `IAgileList` interface represents a value object with a tree structure, which is why the interface extends `ITreeNode`.

The following Agile API methods return an `IAgileList` object (or an array of `IAgileList` objects):

- `IAdminList.getValues()`
- `IAdminList.setValues(IAgileList)`
- `IAttribute.getAvailableValues()`
- `IAttribute.setAvailableValues(IAgileList)`
- `IAgileList.getSelection()`
- `ICell.getAvailableValues()`
- `IListLibrary.createAdminList(java.util.Map)`
- `IListLibrary.getAdminList(java.lang.Object)`
- `IListLibrary.getAdminLists()`
- `IProperty.getAvailableValues()`

The following methods either return an `IAgileList` or require an `IAgileList` parameter when the related attribute, cell, or property is a list (the datatype is `SingleList` or `MultiList`):

- `ICell.getValue()` - For `SingleList` and `MultiList`cells, the returned Object is an `IAgileList`.
- `ICell.setValue(java.lang.Object value)` - For `SingleList` and `MultiList`cells cells, value is an `IAgileList`.
- `IProperty.getValue()` - For `SingleList` and `MultiList`cells properties, the returned Object is an `IAgileList`.
- `IProperty.setValue(java.lang.Object value)` - For `SingleList` and `MultiList`cells properties, value is an `IAgileList`.
- `IRow.getValue(java.lang.Object cellId)` - For `SingleList` and `MultiList`cells cells, the returned Object is an `IAgileList`.
- `IRow.getValues()` - For each `SingleList` and `MultiList`cells 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` and `MultiList`cells cell, value is an `IAgileList`.
- `IRow.setValues(java.util.Map map)` - For each `SingleList` and `MultiList`cells cell in the row, map contains an `IAgileList`.

Selecting a List Value

To select a list value, whether it is a `SingleList` or a `MultiList` list, you must first get the available values for the list. You can then, set the selected value. After selecting the list value, save the selection by setting the value for the cell or property.

The following example shows how to change the value of the Visible property of an attribute. The Visible property is a `SingleList` property with two possible values, No and Yes (or 0 and 1).

Note: You can use `IAgileList getAPIName()` to get the available values of a list. For information, see Chapter , "Accessing Metadata Using the APIName Field."

Example 13–1 Changing the Visible property of an attribute

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

// Get part sub-class
   IAgileClass partClass = admin.getAgileClass(ItemConstants.CLASS_PART);

// Get the "Page Two.List03" attribute
   IAttribute attr = partClass.getAttribute(ItemConstants.ATT_PAGE_TWO_LIST03);

// Get the Visible property
   IProperty propVisible = attr.getProperty(PropertyConstants.PROP_VISIBLE);

// Get all available values for the Visible property
   IAgileList values = propVisible.getAvailableValues();

// Set the selected list value to "Yes"
   values.setSelection(new Object[] { "Yes" });

// Instead of setting the selection to "Yes", you could also
// specify the corresponding list value ID, as in the following line:
   values.setSelection(new Object[] { new Integer(1)});

// Set the value of the property
   propVisible.setValue(values);
   } catch (APIException ex) {
     System.out.println(ex);
}
```

When you use the `IAgileList.setSelection()` method, you can specify `String[], Integer[], or IAgileList[]` values for the `childNodes` parameter. When you select a value from the `IAgileList` object, you can use its String representation or its Integer ID.

To get the currently selected value for a list, use the `IAgileList.getSelection()` method. For a `SingleList` cell or property, `IAgileList` returns an array containing one `SingleList` 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 13–2 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 13–3 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 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 13–4 Working with Lifecycle Phase cells

```
private static void setLifecyclePhase(IItem item) throws APIException {
// Get the Lifecycle Phase cell
   ICell cell = item.getCell(ItemConstants.ATT_TITLE_BLOCK_LIFECYCLE_PHASE);
// Get available list values for Lifecycle Phase
```

```
   IAgileList values = cell.getAvailableValues();
// Set the value to the second phase
   values.setSelection(new Object[] { new Integer(1)});
      cell.setValue(values);
}
```

Working with Dynamic Lists

The Agile server has both static lists and dynamic lists. Static lists contain a selection of values that do not change at run time. Dynamic lists contain a selection of values that are updated at run time. Users with administrator privileges can modify static lists and add new values and make current values obsolete. Dynamic lists cannot be modified; consequently, the Editable property of dynamic lists is set to No.

Several dynamic lists are capable of containing thousands of value objects. Items, Changes, and Users lists are examples of such lists. Although you can use these lists for Page Two and Page Three fields, you can not enumerate values for these lists.

Enumerable and Non-Enumerable Lists

As such, Agile SDK object lists are either enumerable, or non-enumerable. If a specific list is enumerable, you can read the contents of that list. If it is non-enumerable, you cannot access the list directly. For non-enumerable lists, query the Agile class that the object list uses to get the objects that are referenced by the list. The enumeration property for an object is hard coded on the server and cannot be changed.

To determine if the values for a dynamic list can be enumerated, use `IAgileList.getChildNodes()` as shown in the following example. If `getChildNodes()` returns null, the list values cannot be enumerated. However, this does not prevent you from selecting a value for the list.

Example 13–5 Checking values or a dynamic list for enumerability

```
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);
        }
    }
}
```

Directly Checking if Contents of the List is an Enumerable Set

SDK exposes the following API to directly determine if the contents of a list is enumerable:

```
boolean isEnumeratable ( ) throws APIException;
```

This method determines if the contents of the `IAgileList` are enumerable and is a convenient alternative to using `IAgileList.getChildren()` which requires accessing all child values of the list. This method returns a Boolean true if the list is enumerable. That is, it has an accessible child list, and a Boolean false if otherwise.

Non-Enumerable PG&C Lists

The following PG&C lists that were enumerable in earlier releases of the SDK, are no longer enumerable in this release.

- Declarations
- Substances
- Specifications
- Part Families
- Part Families Commodities

Setting the Maximum Value Displayed by a List

Using PLM's Java Client, an authorized user can view and set the number of List rows by selecting **Admin > Server Settings > Preferences** and selecting the **Maximum List Values Displayed** preference.

The new PropertyConstants `PROP_MAXIMUM_LIST_VALUES_DISPLAYED` provides the option to set the maximum value that a List can display in Administrator Preferences. That is, when the maximum value exceeds this set limit, an error message displays the maximum value set in the Preferences, instead of displaying the integer 250 as was done in prior releases.

With the new PropertyConstants `PROP_MAXIMUM_LIST_VALUES_DISPLAYED`, you can get/set preference "Max List Values Displayed" value using this property constant.

For example:

```
INode preferences =
   session.getAdminInstance().getNode(NodeConstants.NODE_PREFERENCES);
IProperty prop =
   preferences.getProperty(PropertyConstants.PROP_MAXIMUM_LIST_VALUES_DISPLAYED);
   System.out.println("Current Value: " + prop.getValue());
      prop.setValue("666");
      System.out.println("New Value: " + prop.getValue());
```

Selecting a List from the List Library

The `IListLibrary` interface enables working with the library of Agile lists. You can load an existing list, or create a new one. To load an existing list, use `IListLibrary.getAdminList()`. You can specify the string name of a list, such as "Disposition" You can also specify a list, by its ID, or by an `AdminListConstants`, such as `LIST_DISPOSITION_SELECTION`. Before you attempt to use a list from the list library, make sure the list is enabled.

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

Example 13–6 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 13–7   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 Agile Java Client, administrators can configure a library of custom lists by choosing **Admin > Data Settings > Lists**. In the Agile API, the IListLibrary interface provides functionality equivalent to **Admin > Data Settings > Lists**. The `IAdminList` interface provides functionality for configuring and customizing each list.

Creating a Simple List

To create a new list, use the `IListLibrary.createAdminList()` method, which takes a map parameter. The map that you pass with `createAdminList()` must contain values for the following `IAdminList` fields:

- `ATT_NAME` - the String name of the list. This is a required field. The list name must be unique.
- `ATT_DESCRIPTION` - the String description of the list. This is an optional field; the default value is an empty string.
- `ATT_ENABLED` - a Boolean value specifying whether the list is enabled. This is an optional field; the default value is **false**.
- `ATT_CASCADED` - a Boolean value specifying whether the list contains multiple levels. This is an optional field; the default value is false. The `ATT_CASCADED` value cannot be changed after the list is created.

Once the list is created, you can use the `IAdminList` interface to enable or disable the list and set values for it.

The following example shows how to create a new list called Colors. This list is a simple list with only one level.

Example 13–8 Example: Creating a simple list

```
try {
// Get the Admin instance
   IAdmin admin = m_session.getAdminInstance();
// Get the List Library
   IListLibrary listLib = admin.getListLibrary();
// Create a new Admin list
   HashMap map = new HashMap();
      String name = "Colors";
      map.put(IAdminList.ATT_NAME, name);
      map.put(IAdminList.ATT_DESCRIPTION, name);
      map.put(IAdminList.ATT_ENABLED, new Boolean(true));
      map.put(IAdminList.ATT_CASCADED, new Boolean(false));
      IAdminList listColors = listLib.createAdminList(map);
// Add values to the list
   IAgileList list = listColors.getValues();
//The list is empty at this point.
   list.addChild("Black");
   list.addChild("Blue");
   list.addChild("Green");
   list.addChild("Purple");
   list.addChild("Red");
   list.addChild("White");
   listColors.setValues(list);
} catch (APIException ex) {
    System.out.println(ex);
}
```

Lists that contain String values are case-sensitive. This means that a list can contain uppercase, lowercase, and mixed-case variations of the same value, which may not be desirable. For example, the following code snippet adds three variations of each color value to the Colors list.

Example 13–9 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");
```

Automatically Creating New Lists by Modifying Existing Lists

Each list attribute must reference an Agile list for its values. If you retrieve an Agile list and modify its values without saving the list and then use those values for a list attribute, the Agile API automatically creates a new list. In the following example, the Colors list is retrieved, but before it is used to populate the values for a list field a new value, "Violet" is added to the list. When IAttribute.setAvailableValues() is called, a new list is created.

Note: Lists that are created automatically by the Agile API have a prefix "SDK" followed by a random number. You can rename such lists, if you prefer.

Example 13–10 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 simple list. You cannot change the value of `IAdminList.ATT_CASCADED` after the list is created.

The following example shows how to create a new cascading list called "Field Office." The list has two levels.

Important: When setting level names for cascading lists, always start with the index 0 for the first level and increment the index subsequent levels as shown in the following two examples below.

Example 13–11 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();
// Add the list of countries
   IAgileList india = (IAgileList)list.addChild("India");
   IAgileList china = (IAgileList)list.addChild("China");
   IAgileList usa = (IAgileList)list.addChild("USA");
   IAgileList australia = (IAgileList)list.addChild("Australia");
// Add the list of cities
   india.addChild("Bangalore");
   china.addChild("Hong Kong");
   china.addChild("Shanghai");
   china.addChild("Suzhou");
   usa.addChild("San Jose");
   usa.addChild("Milpitas");
   usa.addChild("Seattle");
   usa.addChild("Jersey City");
   australia.addChild("Sidney");
// Save the list values
   listFO.setValues(list);
// Set level names starting with index 0 for level 1.
   list.setLevelName(0, "Field Office Country");
   list.setLevelName(1, "Field Office City");
} catch (APIException ex) {
      System.out.println(ex);
}
```

In cascading lists, level names used by the list must be unique and you cannot share them between lists. The level names are stored internally, but Agile Java Client and Web Client currently don't display them. The level names are needed only if you want to show them in a cascading list UI that you created.

After you call the `IAdminList.setValues()` method, a valid ID is assigned to each list value. Only leaf nodes, that is, nodes on the lowest level of a cascading list, have valid IDs. In the previous example, the city nodes are leaf nodes. All other nodes have a null ID. You can use the ID to set the selection of the `IAgileList` object.

You can add a list value and its parent nodes in one statement instead of adding the parent node and then its subnodes. Use the `|` character to separate nodes, which represent levels, in the string. The following example replaces a portion of the code in the previous example; it shows how to add the same list values as in the previous example, but using fewer lines of code.

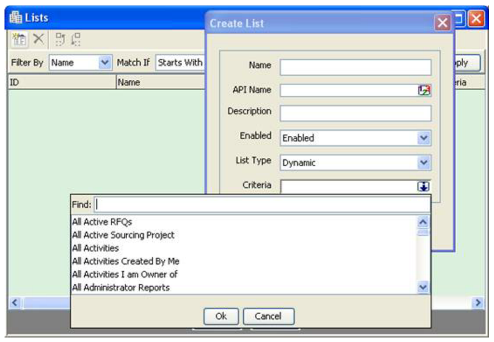
Example 13–12 Adding parent nodes and subnodes to a cascading list

```
// Get the list values
   IAgileList list = listFO.getValues(); // The list is empty at this point.
// Add nodes
   list.addChild("India|Bangalore");
   list.addChild("Hong Kong|Hong Kong");
   list.addChild("China|Suzhou");
   list.addChild("USA|San Jose");
   list.addChild("USA|Milpitas");
   list.addChild("USA|Jersey City");
   list.addChild("Australia|Sidney");
// Save the list values
listFO.setValues(list);
// Set level names
list.setLevelName(0, "Field Office Country");
list.setLevelName(1, "Field Office City");
```

Creating a Criteria-Based List

Criteria-based lists are dynamic lists whose values are defined by the criteria selected from the Agile Criteria library. These lists are created in Java Client's Create List dialog by selecting the "Dynamic" List Type in the drop-down list which opens the Agile Criteria library to select the applicable Criteria.

Figure 13–5 Creating criteria-based lists in Java Client



Filter By: Name
Match If: Starts With

Name:
API Name:
Description:
Enabled: Enabled
List Type: Dynamic
Criteria:

All Active RFQs
All Active Sourcing Project
All Activities
All Activities Created By Me
All Activities I am Owner of
All Administrator Reports

Agile SDK supports creating, loading, and modifying Criteria-based lists by exposing the necessary APIs to:

1. Get the Criteria
2. Create the Criteria-based list
3. Load the Criteria-based list
4. Replace the Criteria-based list

The following examples use the respective APIs to perform these tasks.

Example 13–13 Getting the Criteria from the Agile Criteria library

```
IListLibrary library = m_admin.getListLibrary();
INode lib = m_admin.getNode(NodeConstants.NODE_CRITERIA_LIBRARY);
ICriteria criteria = (ICriteria)lib.getChild("All Change Orders");
```

Example 13–14 Creating the Criteria-based list

```
HashMap params = new HashMap();
   String name = "SDKlist" + System.currentTimeMillis();
params.put(IAdminList.ATT_APINAME, name.toUpperCase());
params.put(IAdminList.ATT_NAME, name.toUpperCase());
params.put(IAdminList.ATT_DESCRIPTION, name.toLowerCase());
params.put(IAdminList.ATT_ENABLED, true);
params.put(IAdminList.ATT_CRITERIA, criteria);
ICriteriaBasedList list =
   (ICriteriaBasedList)library.createDynamicAdminList(params);
      System.out.println("Created list: "+list.getName());
System.out.println
        ("Criteria: "+((ICriteriaBasedList)list).getCriteria().toString());
```

Example 13–15 Loading the Criteria-based list

```
ICriteriaBasedList list =
(ICriteriaBasedList)m_admin.getListLibrary().getAdminList(name.toUpperCase());
System.out.println("Loaded list: "+list.getName());
```

Example 13–16 Replacing the Criteria - Modifying the Criteria-based list

```
ICriteria criteria =
   (ICriteria)lib.getChild("All Designs");
   list.setCriteria(criteria);
System.out.println
   ("New Criteria: "+((ICriteriaBasedList)list).getCriteria().toString());
```

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, to e IAdminList.getListDa get the data type.
- For `SingleList and MultiList` attributes that have only one list level, use `IAttribute.getListDataType()` 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 13–17 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

   }
}
```

Renaming and Removing List Values

The SDK provides the following methods to rename String element entries, or remove an entry in an Agile list:

- The `IAgileList.setValue(Object)` method to modify String list element entries in an Agile `Admin` list.

> **Note:** This method only applies to String values. You can only use this method to modify String entries and not object entries.

- The `IAgileList.clear() and ITree.removeChild(Object)` methods to remove any Agile list entry that is not restricted by the applicable business rules.

The following example uses these methods to modify and clear values of an Agile list.

Example 13–18 Renaming and removing Admin list entries

```
public void exampleClearList() throws Exception {
   IAdmin admin = m_session.getAdminInstance();
   IListLibrary listLibrary = admin.getListLibrary();
   HashMap map = new HashMap();
   String name = "Color";
   String desc = "Example";
      map.put(IAdminList.ATT_NAME, name);
      map.put(IAdminList.ATT_DESCRIPTION, desc);
      map.put(IAdminList.ATT_ENABLED, new Boolean(true));
      map.put(IAdminList.ATT_CASCADED, new Boolean(false));
   IAdminList newList = listLibrary.createAdminList(map);
   IAgileList list = newList.getValues();
   list.addChild("RED");
   list.addChild("GREEN");
   list.addChild("BLUE");
   newList.setValues(list);
   list = newList.getValues();

// Removing the selection
   IAgileList agList = (IAgileList)list.getChild("BLUE");
   Object errorCode = null;
   try {
      list.removeChild(agList);
   }catch(APIException e){
      errorCode = e.getErrorCode();
   }

// Clearing the list
```

```
   list = newList.getValues();
   list.clear();
   newList.setValues(list);

// Clean up
   newList.delete();
}
```

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 `ITreeNode.getChildNode()` to make sure the value is not already present.

Example 13–19 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");
vlistProdLine.setValues(listValues);
}
```

Making List Values Obsolete

You can prevent the selection of a list value by making the list entry obsolete. However, when you invoke `IProperty.getAvailableValues()`, the returned `IAgileList` object can include obsolete list values. This is due to the fact that when the list value is marked obsolete, the server continues to maintain the value in its obsolete list values for existing objects that use these values.

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

Example 13–20 Making a list value obsolete

```
public void checkIfObsolete(IAgileList list) throws APIException {
   if (list != null ) {
      if (list.isObsolete() == false) {
         System.out.println(list.getValue());
      }
   }
}
public void setObsolete(IAgileList list, String value) throws APIException {
   if (list != null ) {
   list.setObsolete(true);
System.out.println(list.getValue() + " is now obsolete.");
   }
}
```

Setting the List Name and Description

To create a list, you must specify a unique name for it. Therefore, when you use `IListLibrary.createAdminList()`, you must pass a value for the `IAdminList.ATT_NAME` field. Other `IAdminList` fields, such as `ATT_DESCRIPTION`, are optional. After the list is created, you can modify it's name and description. The following example shows how to set the name and description of a list.

Example 13–21 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 13-12.

Note: Level names for cascading lists are not displayed in Agile Java Client or Web Client. However, you can choose to display them in Clients you create with the Agile SDK.

Enabling or Disabling a List

When you create a custom list, you can use the `IAdminList.ATT_ENABLED` field to specify whether it's enabled. If you omit this field, the list is disabled by default. The following example shows how to enable and disable a list after it has been created.

Example 13–22 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 13–23 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 Contents of IAgileList Objects

When working with an IAgileList object, particularly one with several levels, it's helpful to print the entire hierarchy of the list. The following code prints the list nodes contained within an IAgileList object.

Example 13–24 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);
}
```

14

Working with Attachments and File Folder Objects

This chapter includes the following:

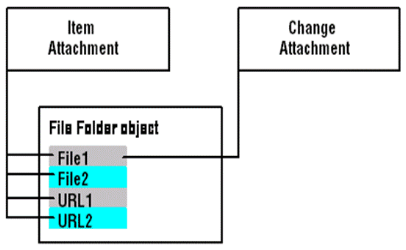
- About Attachments and File Folders
- About SDK's File Folder Utility
- Working with File Folders
- Working with Attachments Table of an Object
- Checking Out a File Folder
- Canceling a File Folder Checkout
- Adding Files and URLs to the Attachments Table

About Attachments and File Folders

Attachments to objects contain information about the object or a manufacturing process. You can attach files and URLs by referencing them in a File Folder object. The File Folder object holds pertinent content, or Attachments. Most primary Agile API objects, such as `IItem, IChange, IManufacturer, IManufacturerPart, IPackage, ITransferOrder, IUser`, and `IUserGroup`, have an Attachments table (or tab in the Java Client) that lists indirect references to the files or URLs that are in separate file folders. Each row in an Attachments table can refer to one file or to all files from a referenced file folder.

The following illustration is an example of the way files or URLs contained in a file folder are referenced indirectly from the Attachments table of multiple business objects, in this case an item and a change.

Figure 14–1 File Folder objects referenced from Item and Change Attachments



The Agile API does not provide support for viewing or printing an attachment. However, after you download a file, you can use another application to view, edit, or print the attachment.

A File Folder is a business object that specifies one or more files or URLs that are stored in the file server vault. In addition, a file folder has its own set of tables. This means that you can create and load an independent file folder and add one or more files to its Files table. You can also search for a file folder, just as you would search for an Item or Change.

Important: File Manager Internal Locator property is set in Agile Java Client. Choose Admin > Settings > Server Settings > Locations > File Manager > Advanced > File Manager Internal Locator. The format for the value is: `<protocol>://<machinename>:<port>/<virtualPath>/services/FileServer`.

For example, `http://agileserver.agile.agilesoft.com:8080/Filemgr/ services/FileServer` is a valid value. For more information about Agile PLM server settings, refer to the Agile PLM Administrator Guide.

About SDK's File Load Utility

SDK's File Load Utility supports automated transfer of files of interest into the file vault. The utility therefore enables the SDK to directly load files and eliminate the file loading step.

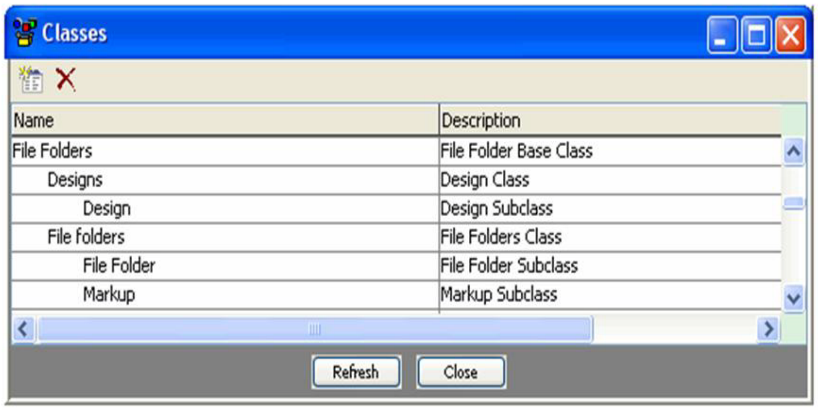
Working with File Folders

Similar to Attachments, the SDK exposes APIs to perform File Folders-related tasks such as checking-in and checking-out files associated with objects in the rows of an Attachments table, adding files and URLs to an Attachments table, and deleting attachments. This section lists and describes these features, and provides the necessary procedures to use the SDK to perform these tasks.

File Folder Classes and Subclasses

The *File Folder Base Class* has two Classes and each of these classes have their own respective Subclasses. The figure below lists the *File Folders Base Class*, *Classes*, and *Subclasses*. The Agile PLM administrator can define new file folder subclasses.

Figure 14–2 File Folders Classes and Subclasses



Name	Description
File Folders	File Folder Base Class
Designs	Design Class
Design	Design Subclass
File folders	File Folders Class
File Folder	File Folder Subclass
Markup	Markup Subclass

A description of these classes and objects appears in Table 14–1 below.

Table 14–1 PLM Classes and descriptions

Base Class	Class	Subclass	Description
File Folders	Designs	Design	Objects that permit building model structures in CAD
	File folders	File Folder Markup	Objects that include files or URLs; this class includes all file folder objects except historical report file folders.

For information about routing these objects, see "Checking the State of Agile PLM Objects" on page 2-19.

File Folder Tables and Constants

The File Folder object supports the following tables and corresponding constants:

Table 14–2 Supported tables and constants

Table	Constant	Read/Write Mode
Title Block	`TABLE_TITLEBLOCK`	Read/Write
Page Two	`TABLE_PAGETWO`	Read/Write
Page Three	`TABLE_PAGETHREE`	Read/Write
Files	`TABLE_FILES`	Read/Write
Structure	`TABLE_STRUCTURE`	Read/Write
Routing Slip/Workflow	`TABLE_WORKFLOW`	Read/Write
Relationships	`TABLE_RELATIONSHIPS`	Read-only
History	`TABLE_HISTORY`	Read-only
Where Used	`TABLE_WHEREUSED`	Read/Write
Where Used Design	`TABLE_WHEREUSEDDESIGN`	Read-only

Creating File Folder Objects

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

Example 14–1 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);
      ff.checkOutEx();
}
```

Note: Note When you add a file or a URL to the row of the Attachments table of a business object, you will automatically create x a new file folder object that contains the associated file or URL. See "Creating File Folder Objects by Adding Rows to Attachments Table" on page 14-6.

The File Folders Design class is similar to the File folder class with the additional Structures table (Tab in the Java Client UI) for CAD objects. The following examples show how to create a Design object, adding a Design object to a the Structure tree, and loading a structure table.

Example 14–2 Creating a Design object

```
// autoNum is autoNumber as usual
   IFileFolder obj = (IFileFolder) m_session.createObject(
   FileFolderConstants.CLASS_DESIGN, autoNum);
   }
```

Example 14–3 Adding Design objects to a Structure tree

```
IFileFolder obj = // some Design object
IFileFolder childObj1 = // some Design object
IFileFolder childObj2 = // some Design object
obj.checkOutEx();
ITable table = obj.getTable(FileFolderConstants.TABLE_STRUCTURE);

// add row
   Object[] vers = childObj1.getVersions();
   IRow row = table.createRow(childObj1);
   row.setValue(FileFolderConstants.
      ATT_STRUCTURE_LABEL,"label modified by creating row 1");
row = table.createRow(childObj2);
   row.setValue(FileFolderConstants.ATT_STRUCTURE_LABEL,
   "label modified by creating row 2");
   obj.checkIn();
```

Example 14–4 Loading a Structure table

```
public void testLoadingDesignStructureTable() throws Exception {

// assuming Design object Design00004 existed with some data in Structure
IFileFolder obj = (IFileFolder) m_session.getObject(
FileFolderConstants.CLASS_DESIGN, "Design00004");

// load Structure table
ITable table = obj.getTable(FileFolderConstants.TABLE_STRUCTURE);
Integer tableId = (Integer) table.getTableDescriptor().getId();
// ITable performs related tasks
}
```

Example 14–5 Loading a Structure table as a tree

```
public void testLoadingDesignStructureTree() throws Exception{

// assuming Design object Design00004 existed with some data in Structure
   IFileFolder obj =
      (IFileFolder) m_session.getObject(
   FileFolderConstants.CLASS_DESIGN, "Design00004");
   IAgileClass agileClass = obj.getAgileClass();
// load Structure table
   ITable table = obj.getTable(FileFolderConstants.TABLE_STRUCTURE);
   Integer tableId = (Integer) table.getTableDescriptor().getId();
   ITreeNode root = (ITreeNode) table;
   Collection topLevelChildren = root.getChildNodes();
   Iterator it;
   ITreeNode row;
   if (topLevelChildren != null) {
      it = topLevelChildren.iterator();
      int level = 0;
   while (it.hasNext()) {
      row = (ITreeNode) it.next();
      if(row instanceof IRow) {
         IRow aRow = (IRow) row;
         IDataObject referent =
         aRow.getReferent();
   if(referent != null) {
      System.out.println(
      "Row Referent Object ID/row:
         "+ referent.getObjectId()+ " / "+referent.getName());
            }
      }
      iterateTreeNode(agileClass, true,tableId, (ITreeNode) row);
      count++;
         }
      }
   System.out.println("The number of rows in top level is " + count);
}
private void iterateTreeNode (IAgileClass agileClass, boolean print, Integer
      tableId, ITreeNode node) throws APIException {
   Collection childNodes = node.getChildNodes();
   if (childNodes == null || childNodes.size() <= 0) {
      return;
   }
   Iterator it = childNodes.iterator();
   ITreeNode childNode;
   IRow row;
   while (it.hasNext()) {
      childNode = (ITreeNode) it.next();
   if (childNode instanceof IRow) {
      row = (IRow) childNode;
      if(row instanceof IRow) {
         IDataObject referent =
         row.getReferent();
         if(referent != null) {
            System.out.println("Row Referent Object ID/row:
              "+ referent.getObjectId()+ " / "+ referent.getName());
         }
      }
   }
iterateTreeNode(agileClass, print, tableId, (ITreeNode) childNode);
      }
   }
}
```

Creating File Folder Objects by Adding Rows to Attachments Table

When you add a file or a URL to the row of the Attachments table of a business object, you automatically create a new file folder that contains the associated file or URL. You can load the referenced file folder using the IRow.getReferent() method, as shown in the following example.

Example 14–6 Creating a file folder by adding a row to the Attachments table

```
public IFileFolder addRowToItemAttachments
(IItem item, File file) throws Exception {
  ITable attTable = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
  IRow row = attTable.createRow(file);
  IFileFolder ff = (IFileFolder)row.getReferent();
  return ff;}
```

Working with the Files Table of a File Folder

The Files table of a file folder lists the files and URLs associated with the object. To edit the table, you must first check out the file folder. You cannot add files or URLs to the Files table or delete them unless the file folder is checked out.

The following example shows how to check out a file folder and then add files and URLs to the Files table.

Example 14–7 Adding files and URLs to the Files table of a file folder

```
public void addFiles(IFileFolder ff, File[] files, URL[] urls) throws Exception {
// Check out the file folder
   ff.checkOutEx();
// Get the Files table
   ITable filesTable = ff.getTable(FileFolderConstants.TABLE_FILES);
// Add files to the Files table
   for (int i = 0; i < files.length; ++i) {
      filesTable.createRow(files[i]);
}
// Add URLs to the Files table
   for (int i = 0; i < urls.length; ++i) {
     filesTable.createRow(urls[i]);
}
// Check in the file folder
   ff.checkIn();
}
```

Accessing Files in Agile PLM File Vault with IAttachmentFile

`IAttachmentFile` is the interface that provides generalized access to files stored in the Agile PLM file vault. This interface is supported by the following Agile API objects:

- **File folder** - you can class cast `IFileFolder` to `IAttachmentFile`.
- **A row of the Files table of a file folder** - you can class cast `IRow` from the Files table to `IAttachmentFile`.
- **A row of the Attachments table of a business object** - you can class cast `IRow` from the Attachments table to `IAttachmentFile`.

`IAttachmentFile` provides the following methods for working with attachments:

- `getFile()`
- `isSecure()`

Note: `IAttachmentFile` also has a setFile() method that lets you change the file(s) for an attachment, but it is supported only for rows of the Attachments table.

Results returned by `IAttachmentFile` methods vary depending on the object that you are working with. See Table 14–3 below.

Table 14–3 Results returned by IAttachmentFile

Calling object	getFile() return value	isSecure() return value
Row from the Attachments table of any business object	Returns either a single file `InputStream` if the row refers to a specific file from the file folder or a zipped `InputStream` with all the files from the file folder.	true if the referenced file is not URL, or all the files are not URLs.
FileFolder object	Returns a zipped `InputStream` with all files from the file folder.	true if all the files contained in the file folder are not URLs.
Row from the Files table of a file folder	Returns a single file `InputStream` that refers to a specific file from the file folder.	true if the referenced file is not a URL.

Note: To read files in a zipped `InputStream`, use methods of the `java.util.zip.ZipInputStream` class.

The following example shows how to use `IAttachmentFile.isSecure()` and `IAttachmentFile.getFile()` from the row of an Attachments table for an item.

Example 14–8 Using isSecure() and getFile()

```
public InputStream getItemAttachment(IItem item) throws Exception {
   InputStream content = null;
      ITable attachments = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
   IRow row = (IRow)attachments.iterator().next();
   if (((IAttachmentFile)row).isSecure())
      content = ((IAttachmentFile)row).getFile();
   return content;
}
```

Working with Attachments Table of an Object

To work with the Attachments table of an object, follow this sequence.

1. Get the object that has the attachment you want.

   For example, you can use the `IAgileSession.getObject()` method to get a particular object, or you can create a query to return objects.

2. Get the Attachments table. Use the `IDataObject.getTable()` or `IAttachmentContainer.getAttachments()` methods to get the table.

3. Select a row in the Attachments table.

Create an iterator for the table, and then select a particular row. You can use the `ITable.getTableIterator()` method to get a bidirectional iterator for the table.

The following example below shows how to retrieve an item, get the Attachments table for the item, and then select the first attachment.

Example 14–9

```
try {
// Get Item P1000
   Map params = new HashMap();
   params.put(ItemConstants.ATT_TITLE_BLOCK_NUMBER, "P1000");
   IItem item =
      (IItem)m_session.getObject(IItem.OBJECT_TYPE, params);
// Get the attachment table for file attachments
   ITable attTable = item.getAttachments();
// Get a table iterator
   ITwoWayIterator it = attTable.getTableIterator();
// Get the first attachment in the table
   if (it.hasNext()) {
      IRow row = (IRow)it.next();
// Read the contents of the stream
   InputSteam stream = ((IAttachmentFile)row).getFile();
   }
   else {
      JOptionPane.showMessageDialog(null, "There are no files listed.",
         "Error", JOptionPane.ERROR_MESSAGE);
   }
} catch (APIException ex) {
      System.out.println(ex);
}
```

Checking In and Checking Out Files with ICheckoutable

`ICheckoutable` is an interface that you can use to check in and check out files that are associated with an object. This applies only to rows of the Attachments table. You can class cast `IRow` from the Attachments table to `ICheckoutable`.

`ICheckoutable` provides the following methods for working with attachments:

- `cancelCheckout()`
- `checkIn()`
- `checkOutEx()`
- `isCheckedOut()`

This example shows how to use the `ICheckoutable` interface to check out and check in a file from a row of the Attachments table.

Example 14–10

```
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 8-1.

Checking if the Revision Is Incorporated

When the revision for an item is released, it is possible the revision is also incorporated. The attachments for an incorporated item are locked and cannot be checked out.

However, you can still view incorporated attachments, but you cannot modify them. To modify an incorporated attachment, you must either un-incorporate the attachment, or submit a new change order to create a new revision as shown in the examples below.

Example 14–11 Incorporating Attachments

```
IItem item = (IItem) session.getObject(IItem.OBJECT_TYPE, "1000");

//Incorporate the item
item.setIncorporated(true);

//Print appropriate response
if (item.isIncorporated() == true)
     System.out.println("Part " + item.getName() + " is Incorporated.");
else System.out.println("Part " + item.getName() + " is NOT Incorporated.");
```

Example 14–12 Un-incorporating Attachments

```
IItem item =
               (IItem) session.getObject(IItem.OBJECT_TYPE, "1000");

//Incorporate the item
    item.setIncorporated(false);

//Print appropriate response
if (item.isIncorporated() == true)
     System.out.println("Part " + item.getName() + " is Incorporated.");
else System.out.println("Part " + item.getName() + " is NOT Incorporated.");
```

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 14–13 Checking out a file folder

```
void checkOutFileFolder(IFileFolder ff) throws Exception {
   ff.checkOutEx();
}
```

Note: You can also use `ICheckoutable.checkOutEx()` to check out a row of the Attachments table. See "Checking In and Checking Out Files with ICheckoutable" on page 14-8.

Canceling a File Folder Checkout

If you check out a file folder and then decide that you don't want to modify it, or you want to discard your changes and revert to the original file folder, you can cancel the checkout. When you cancel a checkout, you also make the file folder available for other users to check out.

Note: Only the user who checked out a file folder can cancel the checkout.

This example cancels a file folder checkout.

Example 14–14 Canceling checkout of a file folder

```
void cancelCheckOut(IFileFolder ff) throws Exception {
      ff.cancelCheckout();
}
```

Note: You can also use ICheckoutable.cancelCheckout() to cancel checkout of a row of the Attachments table. See "Checking In and Checking Out Files with ICheckoutable" on page 14-8.

Adding Files and URLs to the Attachments Table

The Agile API lets you add files and URLs to the Attachments table of many types of objects, such as `IItem`, `IChange`, `IManufacturerPart`, and `IManufacturer`. An attachment is one or more physical files or an Internet address (URL). A file is considered a secured attachment because it is physically stored in the Agile PLM file vault. A URL, on the other hand, is an unsecured attachment.

When you add a file or a URL to the Attachments table of a business object, the server automatically creates a new file folder containing the associated file or URL. The new row on the Attachments table references the new file folder.

When you add a URL attachment, the server stores a reference to the Internet location but does not upload a file. Therefore, you cannot download a URL attachment. The Agile API validates URL strings that you attempt to check in as an attachment. If a URL is invalid, the Agile API considers the string a filename instead of a URL.

You cannot add a file or URL to the Attachments table of an item if

- The current revision has a pending or released MCO.
- The current revision is incorporated.

When you use the ITable.createRow(java.lang.Object) method to add a row to the Attachments table, the param method can be any of the following object types:

- `String` - adds one file attachment specified by a local path.
- `String[]` - adds multiple file attachments specified by an array of local paths.
- `File` - adds one file attachment.
- `File[]`- adds multiple file attachments.
- `InputStream` - adds one file attachment.
- `InputStream[]`- adds multiple file attachments.
- `URL` - adds one URL attachment.
- `URL[]` - adds multiple URL attachments.
- `IRow` (of the Attachments or Files tables) - adds a file or URL attachment.
- `IFileFolder` - adds all files and URLs for the specified file folder.

- Map - adds one or more files specified by a hash table containing Attachment parameters.

Note: The File object type performs best when adding attachments.

When you add a file or a URL to the row of the Attachments table of a business object, you automatically create a new file folder that contains the associated file or URL. You can load the referenced file folder using the `IRow.getReferent()` method, as shown in the following example.

Example 14–15 Creating a file folder by adding a row to the Attachments table

```
public IFileFolder addRowToItemAttachments
      (IItem item, File file) throws Exception; {
   ITable attTable = item.getTable(ItemConstants.TABLE_ATTACHMENTS);
   IRow row = attTable.createRow(file);
   IFileFolder ff = (IFileFolder)row.getReferent();
   return ff;
}
```

This example uses several instances of the `addAttachment()` methods to illustrate the different ways you can add rows to an Attachments table.

Example 14–16 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 a 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 & specify versions for all files
   public static IRow addAttachmentWithVersion
      (ITable attTable, IFileFolder ff) throws APIException {
      ff.setCurrentVersion(new Integer(1));
      IRow row = attTable.createRow(ff);
      return row;
   }
```

Deep Cloning Attachments and Files from One Object to Another

To simplify copying file attachments from one object to another, use the `CommonConstants.MAKE_DEEP_COPY` virtual attribute as a Boolean parameter of `ITable.createRow(Object)`. This parameter allows your program to create a new copy of the file in the Agile File Manager vault instead of referencing the old file.

Example 14–17 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 14–18 Deep cloning the Files table row of a File Folder

```
// Clone a Files table row and its file from one File Folder to another
   public static cloneFilesRow
      (IFileFolder folder1, IFileFolder folder2, File file) throws APIException {

// Check out folder1 and folder2
   folder1.checkOutEx();
   folder2.checkOutEx();

// Get the Files tables of folder1 and folder2
   ITable tblFiles1 = folder1.getTable(FileFolderConstants.TABLE_FILES);
   ITable tblFiles2 = folder2.getTable(FileFolderConstants.TABLE_FILES);

// Prepare params for the first row
   HashMap params = new HashMap();
   params.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, file);

// Add the file to the attachments table of folder1
   IRow row1 = tblFiles1.createRow(params);

// Prepare params for the second row
   params.clear();
   params.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, row1);
   params.put(CommonConstants.MAKE_DEEP_COPY, Boolean.TRUE);

// Add the same file to the Files table of folder2
   IRow row2 = tblFiles2.createRow(params);

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

Specifying the File Folder Subclass When Adding Attachments

You can set up your Agile PLM system with multiple file folder subclasses. If so, when you add a file folder to the Attachments table of a business object, you may want to specify which file folder subclass to use. If you do not specify a subclass, the Agile API uses the default File Folder subclass. The virtual attribute `CommonConstants.ATT_ATTACHMENTS_FOLDERCLASS` makes it easier to specify the required file folder subclass. It enables you to set the attribute to any file folder subclass.

The following example shows how to use the `ATT_ATTACHMENTS_FOLDERCLASS` attribute to specify a subclass when you add a file folder to the Attachments table.

Example 14–19 Specifying the file folder subclass when adding attachments

```
IAgileClass ffclass = m_admin.getAgileClass("MyFileFolder");

// init item
   IItem item = (IItem)session.createObject(ItemConstants.CLASS_PART, "P0001");

// get attachments table
   ITable tab_attachment = item.getAttachments();

// prepare map
   HashMap map = new HashMap();
   map.put(CommonConstants.ATT_ATTACHMENTS_CONTENT, new File("files/file.txt"));
   map.put(CommonConstants.ATT_ATTACHMENTS_FOLDERCLASS, ffclass);
```

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

Retrieving Attachment Files

If a file folder is checked out by another user, you can still retrieve a copy of the file folder file(s) and save it to your local machine. The `IAttachmentFile.getFile()` method returns the file stream associated with a row of the Attachments table. The file stream can be for one file or it can be a zipped file stream for multiple files, depending on how many files the associated file folder has. You can also use `IAttachmentFile.getFile()` to get one or more files directly from a file folder instead of accessing the Attachments table of another business object. If you call `getFile()` from the file folder object, you return the zipped file stream for all files listed on the Files table. If you call `getFile()` from a row of the Files table of a file folder, you return a file stream for the specific file associated with that row.

Note: When you use IAttachmentFile.getFile(), only file attachments are included in the returned file stream. URL attachments don't have files associated with them.

This example retrieves a copy of an attached file.

Example 14–20 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` file using methods of the `java.util.zip.ZipInputStream` class, as shown in the following example.

Example 14–21 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();
   }
}
```

The Agile API provides no direct method for opening an attachment file. However, you can retrieve a file and then have your program open it in a separate application or display it in a browser window.

Deleting Attachments and File Folders

To delete a file folder, which may contain multiple files, use the IDataObject.delete() method. You must have the Delete privilege for file folders to be able to delete them. For more information about deleting objects, see "Deleting and Undeleting Objects" on page 2-21.

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 4-16. Removing a row from the Attachments table does not delete the associated file folder. You cannot delete a row from the Attachments table in the following situations:

- The parent object is an Item whose revision is incorporated.
- The selected attachment is currently checked out.

Working with Thumbnails

Agile PLM supports adding small static graphical images (thumbnails) to key objects which either represent graphical objects or require images. For example, documents attached as files such as Excel worksheets, text files, PDF files, CAD files and so on, can have associated thumbnail images. Thumbnails display scaled down versions of these files and, in the case of Part objects, show how they relate to each other.

The SDK supports the following Thumbnail-related functions:

- Regenerating Thumbnails
- Sequencing Thumbnails
- Setting Master Thumbnails
- Generating Thumbnails while adding Files to Attachments tab

Accessing Thumbnails

Agile SDK provides the `IThumbnailContainer` interface for generalized access to thumbnail-related operations for file folder and business objects. This interface is supported by the following API objects:

- `IFileFolder`
- `IItem`
- `IManufacturerPart`

For `IFileFolder` objects, set the applicable version using `IFileFolder.setCurrentVersion` before calling the above APIs. The default version is

LATEST_VERSION. For IItem or IManufacturerPart objects, use the revision that is already set on these objects.

The following example gets thumbnail details from TitleBlock of an IItem or IFileFolder object.

Example 14–22 Getting thumbnail details from TitleBlock of IItem or IFileFolder objects

```
IItem dataObj = (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
ITable titleBlockTable = dataObj.getTable(TableTypeConstants.TYPE_PAGE_ONE);
Iterator i = titleBlockTable.getTableIterator();
while (i.hasNext()) {
   IRow row = (IRow)i.next();
   Object thumbnailIDDetails =
   row.getValue(ThumbnailConstants.ATT_THUMBNAIL_ATTACHMENT_TAB);
   IAgileList[] nodes =((IAgileList)thumbnailIDDetails).getSelection();
for(int ii=0; ii<nodes.length; ii++) {
   IAgileList childNode = nodes[ii];
   IThumbnailID thumbnailID = (IThumbnailID)childNode.getValue();
   }
}
```

Regenerating Thumbnails

Regenerating a thumbnail means generating a thumbnail for an existing (generated) thumbnail for file folder and item objects. This feature is of particular interest in assembly structures where a change in the child of the assembly structure is reflected in the thumbnail after the thumbnail is regenerated.

Agile SDK provides the IThumbnailContainer.generateThumbnail(IThumbnailID) API for this purpose. When invoked, it will generate and return a new thumbnail. In case of IFileFolder objects, API will use the current *version* of the object. For IItem or IManufacturerPart objects, it will use the current *revision* of the object. An APIException is thrown when the API fails to regenerate the thumbnail for the specified thumbnailID parameter.

Example 14–23 Regenerating a thumbnail for an IFileFolder object

```
IFileFolder ff =
   (IFileFolder)session.getObject(IFileFolder.OBJECT_TYPE, "FOLDER00037");
   ff.setCurrentVersion(new Integer(1));
   IThumbnailID oldThumbnailID = "";

//get this id from row of supported tables like Title Block
   ff.generateThumbnail(oldThumbnailID);
   //Regenerating a thumbnail for an IItem object
     IItem itemObj =
      (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
      IThumbnailID oldThumbnailID = "";

//get this id from row of supported tables like Title Block
   itemObj.generateThumbnail(oldThumbnailID);
```

Setting Master Thumbnails

In Agile PLM, a file folder object is represented by a thumbnail file which can contain several files in its Files tab. Using `setMasterThumbnail`, you can decide which row in the Files tab will represent the selected thumbnail's file folder.

SDK provides the `setMasterThumbnail` API to set master thumbnails on file folder objects. An exception is thrown if the function fails to set the master thumbnail represented by the parameter `masterRow`.

Example 14–24 Setting a master thumbnail

```
IFileFolder ff =
       (IFileFolder)session.getObject(IFileFolder.OBJECT_TYPE, "FOLDER00036");
ITable attachmentTable =
        ff.getTable(FileFolderConstants.TABLE_FILES);
Iterator<?> i =
        attachmentTable.getTableIterator();
while (i.hasNext()) {
         IRow row = (IRow)i.next();
         if (row.getValue("fileName").toString().
                   equals("Resistor Family Data Sheet.pdf")){
   ff.setMasterThumbnail(row);
         }
}
```

Replacing Thumbnails

You can replace an Agile PLM generated thumbnail with a user provided image for file folder and item objects. The SDK provides the following API for this purpose.

`IThumbnailID replaceThumbnail (IThumbnailID oldThumbnailID, byte[] bytes)` throws `APIException`

This API will replace the thumbnail referred to in `oldThumbnailID` with the image file referred to in the input stream. That is, it will return the `ThumbnailID` of the replaced thumbnail.

For `IFileFolder` objects, the API will use the *version* that is already set on the object. For `IItem` or `IManufacturerPart` objects, it will use the *revision* that is already set on the object. An `APIException` is thrown if it fails to replace the thumbnail specified in the `oldThumbnailID` parameter.

Example 14–25 Replacing a thumbnail for an IFileFolder object

```
IFileFolder ff =
   (IFileFolder)session.getObject(IFileFolder.OBJECT_TYPE, "FOLDER00037");
ff.setCurrentVersion(new Integer(1));
IThumbnailID oldThumbnailID = "";
//get this id from row of supported tables like Title Block
   String filePath = "C:\\Earth.bmp";
   File file1_tmp = new File(filePath);
   byte[] b1 = new byte[(int)file1_tmp.length()];
   FileInputStream fileInputStream = new FileInputStream(file1_tmp);
   fileInputStream.read(b1);
   IThumbnailID newThumbnailID =
      itemObj.replaceThumbnail(oldThumbnailID, b1);
   String filePath = "C:Earth.bmp";
```

Example 14–26 Replacing a thumbnail for an IItem object

```
IItem itemObj =
   (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
   IThumbnailID oldThumbnailID = "";
//get this id from row of supported tables like Title Block
   String filePath = "C:Earth.bmp";
   File file1_tmp = new File(filePath);
   byte[] b1 = new byte[(int)file1_tmp.length()];
   FileInputStream fileInputStream = new FileInputStream(file1_tmp);
   fileInputStream.read(b1);
IThumbnailID newThumbnailID = itemObj.replaceThumbnail(oldThumbnailID, b1);
```

Sequencing Thumbnails

When Web Client users add attachment files to business objects, they can also set the order (sequence) of their appearance in the Thumbnail Navigator. Agile PLM provides the `setThumbnailSequence` API to enable this feature in the SDK. For `IItem` or `IManufacturerPart` objects, the API will use the revision that is already set on the object. The API will sort (sequence) the order of appearance using the `thumbnailID` parameter. An exception is thrown if the function fails to set the master thumbnail.

Example 14–27 Sequencing thumbnails

```
IItem itemObj =
   (IItem)session.getObject(IItem.OBJECT_TYPE, "P00015");
IThumbnailID[] thumbnailIDs = null;

//get this id from row of Title Block table
   IThumbnailID[] newSeqOfThumbnailIDs = null;

//generate new order using thumbnail IDs
   itemObj.setThumbnailSequence(newSeqOfThumbnailIDs);
```

Generating Thumbnails while Adding Files to Attachments Tab

There are no APIs specifically for this purpose. When you add a file to the Attachments tab of an Item, a thumbnail is generated for that file provided thumbnail support is enabled in the Web Client.

Working with Design Objects

A *Design* object is a business object that specifies one or more URLs or files that are stored in Agile PLM's File Management Server, and contain information about the binary files that are attached to it. Similar to other Agile PLM business objects, Design objects appear in Agile PLM's class hierarchy as a separate base class.

Design class objects are used with Agile PLM's Engineering Collaboration (EC) module which is used to manage CAD data in Agile PLM. Objects created in this class have many of the same properties and behaviors of File folders. In Java Client, users with administrator privileges can enable other users to open and work with Design objects. Agile PLM users can then access and work exclusively with these objects in the Web Client.

Agile SDK supports the following Design object-related functions:

- Managing (adding, removing, getting, and editing) version specific Relationships between two Design objects
- Using where-used queries for Design object deployments in Agile PLM Class structures. For information about where-used queries, see "Creating a Where-Used Query" on page 3-27.

Adding and Loading Design Objects

To create or get an `IDesign` object, you can use `IAgileSession.createObject()` or `IAgileSession.getObject()`. The following examples show the different methods provided by the SDK to create and get Design objects.

Example 14–28 Creating a Design by class name

```
IDesign des =
   (IDesign) m_session.createObject("Design", "DESIGN00133");
```

Example 14–29 Creating a Design by class ID

```
IDesign des = (IDesign)

   m_session.createObject(FileFolderConstants.CLASS_DESIGN, "DESIGN00133");
```

Example 14–30 Creating a Design by IAgileClass reference

```
IDesign des = (IDesign)
   m_session.createObject(desClass, "DESIGN00133");
```

Example 14–31 Loading a Design object

```
IDesign des = (IDesign)
m_session.getObject(IDesign.OBJECT_TYPE, "DESIGN00133");
```

Managing Version Specific Relationships between Design Objects

Agile SDK supports the following version specific Relationships functions between Design objects:

Note: These version specific functions only apply to Design objects.

- Adding version specific relationships between Design objects
- Removing version specific relationships between Design objects
- Getting version specific relationships for specific versions of Design objects
- Editing version specific relationships for Design objects

Adding Relationships for Specific Versions of Design Objects

The SDK provides the following API to add relationships between two specific versions of Design objects:

`IDesign.addVersionSpecificRelationship(Object versionNum, IDesign relatedDesign, Object relatedVersionNum)`

The parameters are:

- `versionNum` - This an integer showing the version number of this Design object.
- `relatedDesign` - The Design object you are creating the Relationships for.
- `relatedversionNum` - This an integer showing the version number of the Design object you are creating the Relationship for.

An `APIException` is thrown if the version specific relationship between the two Design objects was not created.

Alternatively, you can load the object's `RelationshipsTable` and call `createRow(Object params)` with the following params:

```
HashMap params = new HashMap();
params.put(DesignConstants.ATT_RELATIONSHIPS_REV_VERSION, versionNum);
params.put(DesignConstants.ATT_RELATIONSHIPS_NAME, relatedDesign);
params.put(DesignConstants.ATT_DESIGN_VERSION, relatedVersionNum);
```

Removing Relationships for Specific Versions of Design Objects

To remove Version Specific Relationships for `IDesign`:

```
IDesign des1 = (IDesign)session.getObject(IFileFolder.OBJECT_TYPE, "DESIGN00001");
des1.setCurrentVersion(new Integer(4));
ITable relationshipTable = des1.getRelationship();
relationshipTable.removeRow(row);
```

Getting Relationships for Specific Versions of Design Objects

To get the Relationships for a specific version of `IDesign`:

```
IDesign des1 = (IDesign)session.getObject(IFileFolder.OBJECT_TYPE, "DESIGN00001");
des1.setCurrentVersion(new Integer(4)); //set desired version
ITable relationshipTable = des1.getRelationship();
```

Editing Relationships for Specific Versions of Design Objects

To edit the Relationships for a specific version of the `IDesign` object:

```
IDesign des1 = (IDesign)session.getObject(IFileFolder.OBJECT_TYPE, "DESIGN00001");
des1.setCurrentVersion(new Integer(4));
ITable relationshipTable = des1.getRelationship();
HashMap mapForUpdate=new HashMap();
HashMap rowUpdateMap = new HashMap();
rowUpdateMap.put(DesignConstants.ATT_DESIGN_VERSION, new Integer(1));
mapForUpdate.put(row1, rowUpdateMap);
relationshipTable.updateRows(mapForUpdate);
```

Purging Specific Versions of Design Objects

The SDK provides the `IDesign.purgeVersions(Object[] versions)` API for purging specific versions of Design objects and relevant versions of its child objects. The versions parameter, an integer value, specifies the version number you want purged. An exception is thrown if the API fails to purge the object.

Searching Design Object Deployments with Where-Used Queries

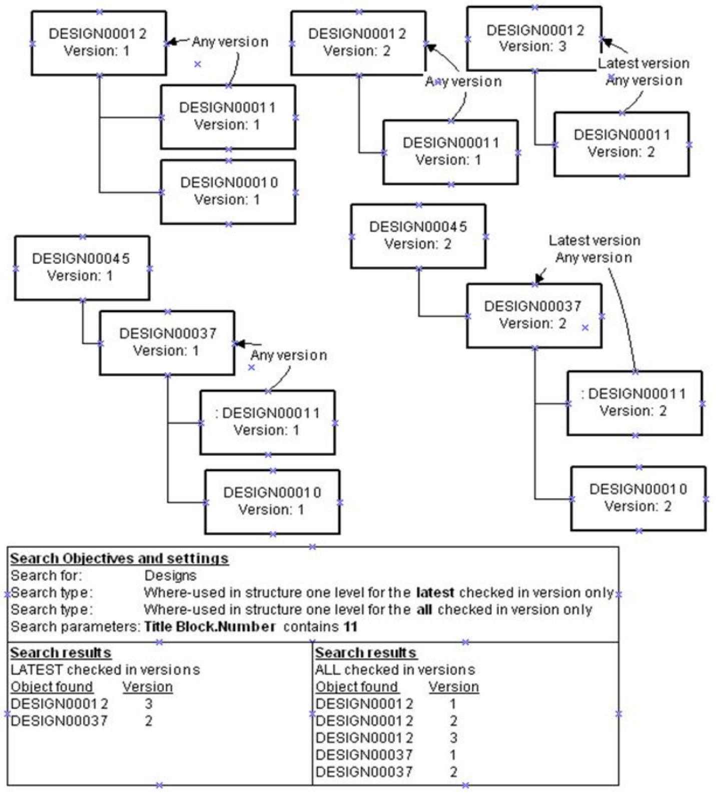
The Structure tab for Design objects enables users to create structures of different Design objects having different versions. The SDK supports searching for Design object usage in Agile PLM Class Structures for the *latest* checked in versions and *all* checked in versions with the following queries and query constants:

- `WHERE_USED_IN_STRUCTURE_ONE_LEVEL_LATEST_CHECKEDIN` - This `WHERE_USED` query *returns* the *LATEST* version of the immediate parent of the Design object which uses the input Design object as a child in the design structure. The constant `QueryConstants.WHERE_USED_IN_STRUCTURE_ONE_LEVEL_LATEST_CHECKEDIN` supports this search.
- `WHERE_USED_IN_STRUCTURE_ALL_LEVEL_LATEST_CHECKEDIN` - This `WHERE_USED` query *returns ALL* versions of the immediate parent of the Design object which uses the input Design object as a child in the design structure. The constant `QueryConstants.WHERE_USED_IN_STRUCTURE_ONE_LEVEL_ALL_CHECKEDIN` supports this search.

You can find code samples using `QueryConstants` in `SDK_samples.zip` folder. To access this file, see the **Note** in "Client-Side Components" on page 1-2. These are the Javadoc generated HTML files in the documentation folder.

The two searches and their respective results are explained with the aid of the following illustration. It shows the Design objects and level one structures. The search parameter `Title Block.Number` includes the number 11.

Figure 14–3 Design objects and search results



<u>Search Objectives and settings</u>

Search for:	Designs
Search type:	Where-used in structure one level for the **latest** checked in version only
Search type:	Where-used in structure one level for the **all** checked in version only
Search parameters:	**Title Block.Number** contains **11**

<u>Search results</u>
LATEST checked in versions

Object found	Version
DESIGN00012	3
DESIGN00037	2

<u>Search results</u>
ALL checked in versions

Object found	Version
DESIGN00012	1
DESIGN00012	2
DESIGN00012	3
DESIGN00037	1
DESIGN00037	2

15

Importing and Exporting Data with SDK

This chapter includes the following:

About Importing and Exporting Data

You can use the SDK to import and export data from external databases into the PLM system. The source can be an Agile database, a third party Product Data Management (PDM) system, or an Enterprise Resource Planning (ERP) system. The following paragraphs provide background information, procedures, and examples to perform these tasks using the agile SDK.

You can use the SDK to import and export data from external databases into the PLM system. The source can be an Agile database, a third party Product Data Management (PDM) system, or an Enterprise Resource Planning (ERP) system. The following paragraphs provide background information, procedures, and examples to perform these tasks using the agile SDK.

Validating Import Data and Importing Data

When you import data, you have the option to validate the data, or ignore this step. The purpose of import validation is to check the data for compliance with applicable server rules such as length tolerances, allowable values, and other constraints. The validation process informs you of the data that will fail to import before initiating the process.

The SDK exposes two methods to programmatically perform the following import-related tasks:

- The IImportManager.validateData(byte[], String, byte[], byte[], String[], List) method to validate the imported data for compliance with server business rules. This action is performed before importing the data to identify the invalid items in the input source data.
- The IImportManager.importData(byte[], String, byte[], byte[], String[], List) method supports importing data into the PLM databases. This action is performed after running the IImportManager.validateData() method to select the data that meets the server business rules and is importable into the PLM system.

For more information about importing data, refer to *Agile Integration Services Developer Guide* and *Agile Import and Export Guide*.

The following example uses these methods to validate the imported data for compliance and import it into the PLM system upon validation.

Example 15–1 Validating and Importing Data into PLM

```
import com.agile.api.*;
import java.util.*;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class ImportClient {
   public static IAgileSession session = null;
   public static AgileSessionFactory factory;
   public static void main(String[] args) {

try {
   String _url="http://localhost/Agile";
   String _user="admin";
   String _pwd="agile";
   String srcFilePath="bom.txt";

/* Supported file types: aXML,IPC2571, ExcelFile, DelimitedTextFile
* The value of "-f" parameter is the same
* in Import AIS sample command
*/
   String srcFileType="DelimitedTextFile";

// Null implies loading the default mapping
   String mappingPath="NewMapFile.xml";

// Null implies do not transform
   String transformPath=null;

/* The value used by operations and options are the same as the value of  "-t"
   parameter in the import AIS sample command.
*/
   String [] operations=new String[]{"items", "items.bom","items.aml"};
*/
   List options=new ArrayList();
   options.add("BusinessRuleOptions|ChangeMode=Authoring");
   options.add("BusinessRuleOptions|BehaviorUponNonExistingObjects=Accept");
   String _output="log.xml";
   FileOutputStream fop=new FileOutputStream(_output);

// Create an instance of IAgileSession
   session = connect(_url,_user,_pwd);
   IImportManager imgr = (IImportManager)
      session.getManager(IImportManager.class);
   byte[] logData=null;
// Insert Code to import and validate data. See sample in Example 15-2.
   int n=0;
   InputStream logStream=byte2stream(logData);
   while((n=logStream.read(buf))!=-1){
   fop.write(buf, 0, n);
      }
       fop.close();
      }
       catch (Exception e) {e.printStackTrace();
      }
      finally {session.close();
      }
   }

/*
   * <p> Create an IAgileSession instance </p>
   * @return IAgileSession
   * @throws APIException
*/

   private static IAgileSession connect(
      String _url,String _user,String _pwd) throws APIException {
   factory = AgileSessionFactory.getInstance(_url);
   HashMap params = new HashMap();
   params.put(AgileSessionFactory.USERNAME, _user);
   params.put(AgileSessionFactory.PASSWORD, _pwd);
   session = factory.createSession(params);
   return session;
   }
   private static byte[] stream2byte(InputStream stream)
      throws IOException {
   ByteArrayOutputStream outStream=new ByteArrayOutputStream();
   byte buf[]=new byte[1024*4];
   int n=0;
   while((n=stream.read(buf))!=-1){
      outStream.write(buf, 0, n);
      }
   byte[] data=outStream.toByteArray();
   outStream.close();
   return data;
   }

   private static InputStream byte2stream(byte[] data) throws IOException{
   ByteArrayInputStream stream=new ByteArrayInputStream(data);
   return stream;
   }

   private static byte[] convertFiletoStream(String path) throws IOException{
   if(path==null || path.equals(""))
      return null;
      return stream2byte(new FileInputStream(path));
      }
   }
```

Example 15–2 Sample code to import and validate data

```
/* Sample code for Example 15-1, "Validating and Importing Data into PLM"
   * Remove comments to run the importData example.
   * byte[]logData=imgr.importData(stream2byte
   * (new FileInputStream(srcFilePath)),
   * srcFileType, convertFiletoStream(mappingPath),
   * convertFiletoStream(transformPath),
   * operations, options);
   * Sample code to validate data
   * Remove comments to run the validateData example
   * logData=imgr.validateData(stream2byte
   * (new FileInputStream(srcFilePath)),
   * srcFileType, convertFiletoStream(mappingPath),
   * convertFiletoStream(transformPath), * operations, options);
   * byte buf[]=new byte[1024*4];
   * }
```

Exporting Data from the SDK

The SDK exposes the exportData()method to programmatically export data from PLM databases. This method is designed to overcome performance and memory issues that are encountered when loading large BOMs into the SDK programs. To overcome this issue, you can invoke the export functionality to load the BOM. The SDK programs are then able to read and export the data from extracted XML files.

For more information about exporting data, refer to the *Agile Integration Services Developer Guide* and *Agile Import and Export Guide*.

Invoking SDK's Export Function

Use the following call to invoke the export function of the SDK.

public byte[] **exportData** (Object[], Integer, String[])

In this call,

- exportData - Is the method that returns the exported data in an array of bytes. The byte array represents a ZIP file that contains the export XML file in aXML or PDX formats and any file attachments that are included in the exported package.
- Object[]- Is the array of objects that are exported from PLM to the external system. These objects are passed as IDataObject objects.
- Integer - Is the indicator (constants that are provided in ExportConstants.java) to identify whether the output export format should be aXML or PDX. These are the two formats that the SDK supports.
- String[] - Is the array of ACS filter names that are used for the export. The filter names are not case sensitive and must match the names of filters defined by the Admin tool for ACS.

The conditions that causes the exportData method to throw an exception and the respective exceptions are:

- Invalid Data Format - The method was called with an unrecognized value for the export data format. Only aXML (provide constant label) and PDX (provide constant label) values are valid.
- No Filter Specified - The method was called but no filters were specified. At least one valid filter must be provided.
- Specified Filter Not Found - The method was called with specified filter which was not found in the system

Example 15–3 Exporting data from PLM using the SDK

```
   ... //

   IItem item = (IItem) session.getObject(IItem.OBJECT_TYPE, "P0001");
   if (item == null) {
   ... // throw an error, the part wasn't found
   }

   IDataObject[] expObjs = {item};
   String[] filters = {"Default Item Filter"};

   ...

   IExportManager eMgr = (IExportManager)
      session.getManager(IExportManager.class);
      try {
      byte[] exportData =
      eMgr.exportData(expObjs, ExportConstants.EXPORT_FORMAT_PDX, filters);
   if (exportData != null) {
      String fileName = createOutputFileName();
      FileOutputStream outputFile = new FileOutputStream(fileName);
      outputFile.write(exportData);
      outputFile.close();
......System.out.println("Data exported to file: " + fileName);
      }
   } catch (Throwable t) {
       // error handling
   }
```

Importing and Exporting Microsoft Project 2010 Files

Microsoft Project 2010 (MSP) integration with PPM is based on the XML format used by MSP. Project data between MSP and PPM is transferred in XML format. The XML format used for this integration is the same as the one used by MSP. This integration is carried out with the aid of the following three operations:

- Validate - All MSP data is validated before importation
- Import - Imports project data by synchronizing the data from MSP to the corresponding PPM project tree. In the event a PPM Project tree is not available, this operation will create a new project tree.
- Export - Exports project data by synchronizing the data from PPM to MSP

You can perform these operations programmatically using APIs listed in the following paragraphs.

Validating MSP Import Data

This API validates MSP data in XML format. In this process, it takes the document object as the parameter and returns a Boolean True if the validation is successful. If there are any validation errors, it throws an API exception to the client

```
Object validateXML(Object documentObject) throws APIException
```

- Validate API is exposed in `IAgileSession`.and enables validating the XML data that was newly created in MSP.

- Validate API is also exposed in `IProgram` and enables validating the data in the document that was retrieved from an existing Program in PPM.

Validate APIs of session and program uses the `validateMSPXML` API of the Activity to validate do the XML data. It returns the `MspSyncActivityVO` object which provides the validation results. The `MspSyncActivityVO` object is passed to the `validateXML` method of `PCUtil` which processes the validation information. The `validateXML` method returns true if the validation is successful and there were no errors. In case of any validation errors, the exceptions are added to a batch exception file that is thrown.

Example 15–4 Validate the data prior to publication

```
// Create an IAgileSession instance - Login to Agile server.
   session = connect();
//Validate the document before publication
   session.validateXML(document)
```

Importing MSP Data

This Import operation reads the MSP project data sent in the XML format and synchronizes it with project data in the PPM. The File Manager (DFM) is used as an intermediary tool to transfer the MSP XML file from the Client machine to the Server.

The Import operation is performed in one of the two following modes:

- Create Mode - A new PPM project tree is created for the imported MSP project XML file.
- Update Mode - Updates the existing PPM project tree with the MSP project data.

This API imports data from MSP to PPM and returns the `IProgram` to the Client upon successful completion of the Import operation.

Create Mode

For Create mode, the Import API is exposed in `IAgileSession` and enables publishing the data which was newly created in the XML format from MSP to PPM.

- API Signature

  ```
  Object publishXML(Object documentObject, Object resNameToUserIdObject, Object
  resNameToRolesObject, Integer templateType, boolean setScheduleEditorToPPM)
  throws APIException
  ```
- API Parameters
  - `documentObject` - This parameter is a Document Object retrieved from the MSP XML.
  - `resNameToUserIdObject` - This parameter is a `Map<String, Long/IUser>` of User Name to Agile User ID. You can pass the user ID can as a Long object or an `IUser` object.
  - `resNameToRolesObject` - This parameter is a `Map<String, String/IRole[]>` of User Name to array of Roles. Roles can be a String array of Role IDs or an array of `IRole` Objects.
  - `templateType` - This parameter specifies the template type for the new project. Values of template type are defined in ProgramConstants.java as `ACTIVE_STATE, TEMPLATE_STATE and PROPOSED_STATE`.
  - `setScheduleEditorToPPM` - This parameter determine whether the Schedule Editor for the new Program is set to PPM or not.

IProgram program = (IProgram)session.publishXML(document, resNameToUserId, resNameToRoles, ProgramConstants.ACTIVE_STATE, false)

Update Mode

The import API for Update mode is exposed in IProgram and enables users to publish the data that was retrieved from PPM and modified in MSP

- API Signature - This API takes only three parameters explained above.

  ```
  Object publishXML(Object xmlDocument, Object resNameToUserIdObject,Object
  resNameToRolesObject) throws APIException
  ```
- Code sample

  ```
  IProgram program = (IProgram)program.publishXML(document, resNameToUserId,
  resNameToRoles);
  ```

The Import API for session and program publishes the XML data to PPM only if data validation is successful. It will then call the validate API to ensure the data designated for publication is correct. Once validation is successful without any errors, `SyncMspToPE` API of Activity is used to publish the XML data to PPM. Data inand `resNameToRolesObject` is prepared by `prepareUsersForMSP` of `PCUtil` in the required format for `SyncMspToPE`.

Exporting MSP Data

The Export operation is invoked on a particular PPM project to generate the PPM project data in MSP XML format.

The export operation is performed in one of the two following modes

- Create Mode - A new XML file in MSP XML format is generated with the selected PPM project data. This XML file will contain a limited subset of the XML tags.
- Update Mode - It there are any previously imported MSP XML files for the PPM project that are stored in DFM, the XML file is sent to the Client once it is updated with the latest PPM project data such as task name,% complete, description, and so on.

API Signature

```
Object saveAsXML(Integer saveAsXMLMode) throws APIException;
```

The `saveAsXMLMode` is the only parameter that is supplied to the API. It defines the mode in which the data is exported. The values of `saveAsXMLMode` are defined in `ProgramConstants.java` as `SAVE_AS_XML_READ and SAVE_AS_XML_EDIT`. If the mode type is Read, then the data exported to MSP is available for read-only purposes and cannot be published back to PPM. If the mode type is Edit, then the exported data can be modified in MSP and can be published back to PPM.

```
document =
   (org.jdom.Document)program.saveAsXML(ProgramConstants.SAVE_AS_XML_EDIT);
```

The export API uses the `SyncPEToMSP` API of Activity to return the program data in the form of `org.jdom.Document` object.

16 Managing Workflow

This chapter includes the following:

- About Workflow
- The Change Control Process
- Dynamics of Workflow Functionality
- Selecting a Workflow
- Adding and Removing Approvers
- Managing Functional Teams

About Workflow

- Route changes automatically to the users who need to approve or observe the change.
- Send mail 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.

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

The Change Control Process

The change control process can vary for each Workflow defined for a routable object. The table below lists the sequences for the default Work flows 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 16–1 Workflow default sequences for routable objects

Workflow	Default sequence
Default Activities	Not Started > In Process > Complete
Default Attachments	Review
Default Audits	Prepared > Initiated > Audited > Issued > Corrected > Validated > Closed

Table 16–1 (Cont.) Workflow default sequences for routable objects

Workflow	Default sequence
Default CAP As	Identified > Acknowledged > Investigated > Implemented > Validated > Closed
Default Change Orders	Pending > Submitted > CCB > Released > Implemented
Default Change Requests	Pending > Submitted > CCB > Released > Closed
Default CT Os	Pending> Review > Released > Complete
Default Declarations	Pending > Open to Supplier > Submit to Manager > Review > Released > Implemented
Default Deviations	Pending > Submitted > CCB > Released > Expired
Default Gates	Closed > In Review > Open
Default Manufacturer Orders	Pending > Submitted > CCB > Released > First Article Complete
Default Non-Conformance Reports	Pending > Submitted > Review > Released > Closed
Default Packages	Pending > Submitted > Review > Accepted > Closed
Default Price Change Orders	Pending > Submitted > Price Review > Released > Implemented
Default Problem Reports	Pending > Submitted > Review > Released > Closed
Default Sites Change Orders	Pending > Submitted > CCB > Released > Implemented
Default Stop Ships	Pending > Submitted > CCB > Released > Resumed

Dynamics of Workflow Functionality

The Workflow functionality available to each user for a particular routable object depends on the status of the routable object and the user's privileges. Your Agile API program should take these Workflow dynamics into account and, where possible, adjust your program accordingly.

How the Status of a Change Affects Workflow Functionality

The Workflow actions available for a pending change are different from those for a released change. To check the status of a change to determine whether it's pending or released, use the `IRoutable.getStatus()` method. The `getStatus()` method returns an `IStatus` object for the Workflow status. `IStatus` extends the `IStatus` 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 16–1 Getting the status of a change object

```
IChange change = (IChange)session.getObject(IChange.OBJECT_TYPE, "C00008");
System.out.println(change.getStatus())
```

Example 16–2 Getting status of a change object and comparing it to a status type

```
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. The table below lists privileges needed to perform Workflow actions.

Table 16–2 Agile privileges and related Workflow APIs

Privilege	Related API
Change Status	`IRoutable.changeStatus()`
Comment	`IRoutable.comment()`
Send	`DataObject.send()`

To determine at run time whether a user has the appropriate privileges to perform an action, use the `IUser.hasPrivilege()` method. You can adjust your program's UI based on the user's privileges. The following example shows how to check whether a user has the privilege to change the status of a change before calling the `IRoutable.changeStatus()` method.

Example 16–3 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)) {
    List<?> approvers = Arrays.asList(user);
    IStatus nextStatus = change.getDefaultNextStatus();
    change.changeStatus(nextStatus, true, "", true, true,
    null, approvers, null, null, false);
   } else {
   System.out.println("Insufficient privileges to change status.");
   }
}
```

Selecting a Workflow

When you create a new change, package, product service request, or quality change order, you must select a Workflow. Otherwise, the object is in an unassigned state and cannot progress through a Workflow process. Your Agile system can have multiple Workflows defined for each type of routable object. To retrieve the valid Workflows for an object, use the `IRoutable.getWorkflows()` method. If a routable object has not been assigned a Workflow yet, you can use the `IRoutable.getWorkflows()` method to select a Workflow.

As long as a change is in the Pending status, you can select a different Workflow. Once a change moves beyond the Pending status, you can't change the Workflow.

Example 16–4 Selecting a Workflow

```
private static IWorkflow addWorkflow(IChange change) throws APIException {
IWorkflow[] wfs = change.getWorkflows();
      IWorkflow   workflow = null;
      for (int i = 0; i < wfs.length; i++) {
      if (wfs[i].getName().equals("Default Change Orders"))
            workflow = wfs[i];
      }
      change.setWorkflow(workflow);
```

```
        return workflow;
}
```

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 `IRoutable.setWorkflow()` and specify null for the Workflow parameter.

Example 16–5 Making a change unassigned

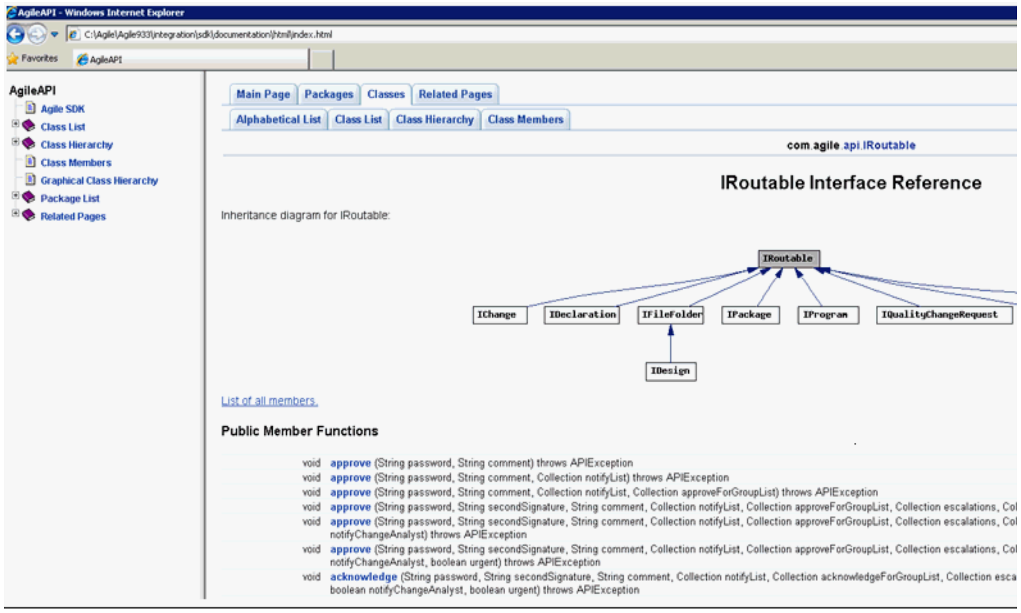
```
private void unassign(IChange change) throws APIException {
   change.setWorkflow(null);
}
```

Adding and Removing Approvers

After a change is routed and the online approval process is initiated, it is sometimes necessary to add or remove people from the list of approvers or observers. To add or remove approvers or observers, a user must have the Route privilege.

There is no need to load the Workflow table to modify the list of approvers. Once you have a routable object, such as an ECO, you can modify its list of approvers using the `IRoutable.addApprovers()` and `IRoutable.removeApprovers()` methods. When you use `addApprovers()` or `removeApprovers()`, you specify the lists of approvers and observers, whether the Notification is urgent, and an optional comment. The Agile API provides overloaded `addApprovers()` and `removeApprovers()` methods for adding or removing a user or a user group from the list of approvers. For more information, refer to API Reference files at Oracle® E-Cloud Web site (`http://edelivery.oracle.com/`). These files are available and accessible from `http://edelivery.oracle.com`, after installing the Agile SDK. See Figure 16–1.

Figure 16–1 API reference files



IRoutable Interface Reference

Inheritance diagram for IRoutable:

Public Member Functions

```
void  approve (String password, String comment) throws APIException
void  approve (String password, String comment, Collection notifyList) throws APIException
void  approve (String password, String comment, Collection notifyList, Collection approveForGroupList) throws APIException
void  approve (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList, Collection escalations, Col
void  approve (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList, Collection escalations, Col
      notifyChangeAnalyst) throws APIException
void  approve (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList, Collection escalations, Col
      notifyChangeAnalyst, boolean urgent) throws APIException
void  acknowledge (String password, String secondSignature, String comment, Collection notifyList, Collection acknowledgeForGroupList, Collection esca
      boolean notifyChangeAnalyst, boolean urgent) throws APIException
```

If the user that you select as an approver or observer, do not have the appropriate privileges to view a change, your program will throw 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 16–6 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 hhawkes to approvers and flang 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 16–7 Adding approvers without changing observers

```
public void addMeToApprovers(IChange change) {

// Get the current user
   try {
      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);
   }
}
```

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. Example 16–7 adds current users to the list of approvers without changing this list.

Approving or Rejecting Change

After a change is routed to a group of approvers, the online approval process begins. Users listed in the Workflow table for a change can approve or reject the change.

When you approve a change, the Agile system records the approval on the Workflow table. When all approvers have approved the change, the system sends an email Notification to the change analyst or component engineer indicating that the change is ready to be released.

Note: To approve or reject a change, users must have the correct privileges. For more information, refer to *Agile PLM Administrator Guide*.

When you use the IRoutable.approve() method, specify the user's approval password and an optional comment. The approve() methods enable you to specify a Notification list and a collection of user groups for which you are approving.

After installing the Agile SDK, API Reference files, shown in Figure 16–1, "API reference files" are available and accessible from the Oracle® E-Cloud Web site at `http://edelivery.oracle.com`. You can locate and download the void approve (String, Collection) throws an `APIException` method from this site.

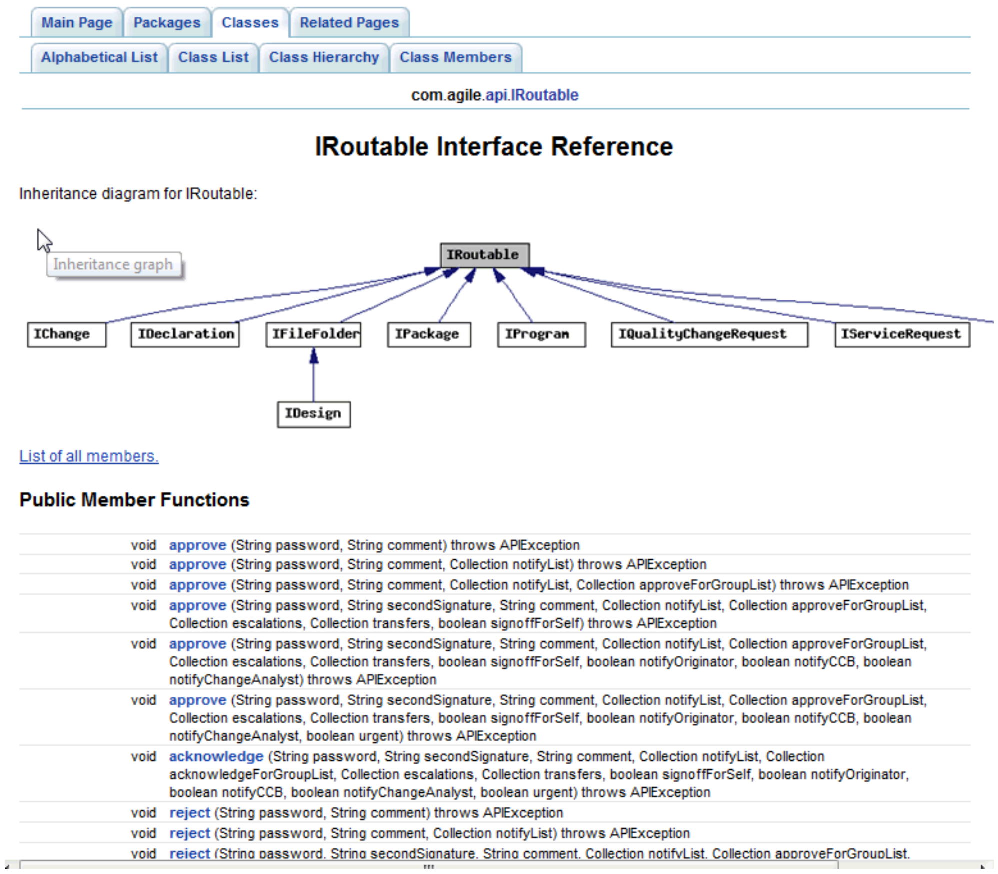
Following paragraphs document approving or rejecting a given routable object. APIs that support approving or rejecting a change object when a second signature is required are described in detail in Setting the “Signoff User Dual Identification” Preference.

Example 16–8

```
public void approveChange(IChange change){
   try {
        change.approve("agile","", "Looks good to me",
              null, null, null, null, false, false, false, false, false);
   } catch (APIException ex) {
         System.out.println(ex);
   }
```

To download SDK’s approve and reject methods, locate the `IRoutable` Interface Reference from `http://edelivery.oracle.com` and then look for these methods under "Public Member Functions." See "IRoutable Approve and Reject methods."

Figure 16–2 IRoutable Approve and Reject methods



com.agile.api.IRoutable

IRoutable Interface Reference

Inheritance diagram for IRoutable:

List of all members.

Public Member Functions

void approve (String password, String comment) throws APIException
void approve (String password, String comment, Collection notifyList) throws APIException
void approve (String password, String comment, Collection notifyList, Collection approveForGroupList) throws APIException
void approve (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList, Collection escalations, Collection transfers, boolean signoffForSelf) throws APIException
void approve (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList, Collection escalations, Collection transfers, boolean signoffForSelf, boolean notifyOriginator, boolean notifyCCB, boolean notifyChangeAnalyst) throws APIException
void approve (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList, Collection escalations, Collection transfers, boolean signoffForSelf, boolean notifyOriginator, boolean notifyCCB, boolean notifyChangeAnalyst, boolean urgent) throws APIException
void acknowledge (String password, String secondSignature, String comment, Collection notifyList, Collection acknowledgeForGroupList, Collection escalations, Collection transfers, boolean signoffForSelf, boolean notifyOriginator, boolean notifyCCB, boolean notifyChangeAnalyst, boolean urgent) throws APIException
void reject (String password, String comment) throws APIException
void reject (String password, String comment, Collection notifyList) throws APIException
void reject (String password, String secondSignature, String comment, Collection notifyList, Collection approveForGroupList,

If a change has a fundamental flaw, users listed on the Workflow table may reject it. When you reject a change, the system records the rejection on the Workflow tab for the change and sends an email Notification to the change analyst or component engineer. The change analyst or component engineer may decide to return the rejected change to the originator, thus reverting its status to Pending.

When you use the IRoutable.reject() method, you must specify the `userNotifications` approval password and optional comments. An overloaded `reject()` method enables specifying a Notification list and a collection of user groups for which you're approving. For more information and additional examples similar to Figure 16–9, refer to the API Reference files at Oracle® E-Cloud Web site (`http://edelivery.oracle.com/`).

The following example shows how to reject a change.

Example 16–9 Rejecting a change

```
public void rejectChange(IChange change) {
   try {
       change.reject("agile","", "Looks good to me",
       null, null, null, null, false, false, false, false, false);
   } catch (APIException ex) {
System.out.println(ex);
   }
}
```

Approving or Rejecting a Change Without Password

Agile PLM's Java Client provides the option to configure Workflow Settings to enable the approval or rejection of a change with or without typing a password. Users with the Administrator role and privileges configure this option by selecting the **Yes** or **No** option in the **Password Required** field.

The following example uses the Agile SDK to programmatically configure this requirement.

Example 16–10 Approving or rejecting change without a password

```
IAdmin admin = session.getAdminInstance();
INode root = admin.getNode(NodeConstants.NODE_AGILE_WORKFLOWS);
INode CCBStatus =
         (INode)root.getChildNode("Default Change
              Orders/Status List/CCB/Status Properties");
IProperty PwdReq =
         CCBStatus.getProperty(PropertyConstants.PROP_WORKFLOW_PASSWORD_REQUIRED);
IAgileList value = (IAgileList)PwdReq.getAvailableValues();
value.setSelection(new Object[] {"Approve Only"});
PwdReq.setValue(value);

// Approve change without passing password
change.approve(null, null, "Looks good to me", null, null, null, null, false,
false, false, false, false);
```

Commenting a Change

When you comment a change, you send a comment to other CCB reviewers during the online approval process. In addition to the comment, you can specify whether to notify the originator, the change analyst, and the change control board. An overloaded comment() method allows you to specify a Notification list. For more information, similar to refer to the API Reference files at Oracle® E-Cloud Web site (`http://edelivery.oracle.com/`).

The following example shows how to comment a change.

Example 16–11 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 workflow, you can audit it to determine if any required entry cells are not completed or if the change violates any Agile Smart Rules. 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 16–12 Auditing a change

```
public void auditChange(IChange change) {
   try {

// Audit the release
   Map results = change.audit();

// Get the set view of the map
   Set set = results.entrySet();
// Get an iterator for the set
   Iterator it = set.iterator();

// Iterate through the cells and print each cell name and exception
   while (it.hasNext()) {
      Map.Entry entry = (Map.Entry)it.next();
      ICell cell = (ICell)entry.getKey();
   if(cell != null) {
      System.out.println("Cell : " + cell.getName());
   }
   else {
      System.out.println("Cell : No associated data cell");
   }

//Iterate through exceptions for each map entry.
//(There can be multiple exceptions for each data cell.)
   Iterator jt = ((Collection)entry.getValue()).iterator();
      while (jt.hasNext()) {
        APIException e = (APIException)jt.next();
        System.out.println("Exception : " + e.getMessage());
      }
   }
} catch (APIException ex) {
    System.out.println(ex);
   }
}
```

Changing the Workflow Status of an Object

The IRouteable.changeStatus() method is a general purpose method for changing the status of an Agile object. For example, you can use `changeStatus()` to submit, release, or cancel a change. In instances such as failed audits, it throws the compound exception `ExceptionConstants.API_SEE_MULTIPLE_ROOT_CAUSES`. You can disable this exception by modifying the code that caught the exception. See the example below.

Example 16–13 Throwing compound exceptions

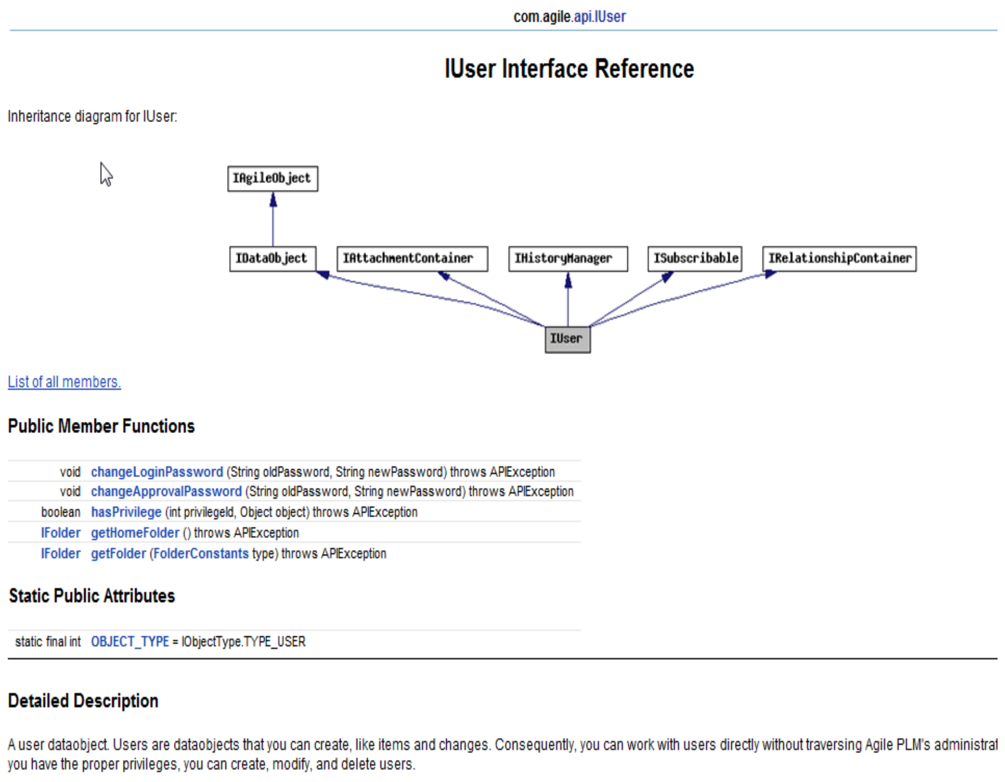
```
while (true) {
   try {
      change.changeStatus(
   wf.getStates(expectStatus)[0],
   false,
   "comment",
   false,
   false,
   null,
   null,
   null,
   false);
   } catch (APIException ae) {
      try {
         if
            (ae.getErrorCode().equals
               (ExceptionConstants.API_SEE_MULTIPLE_ROOT_CAUSES)){
                  Throwable[] causes = ae.getRootCauses();
                  for (int i = 0; i < causes.length; i++) {
                       m_session.disableWarning(
                       (Integer)((APIException)causes[i]).getErrorCode());
                       }
                  } else {
                     m_session.disableWarning((Integer)ae.getErrorCode());
                  }
               } catch (Exception e) {
                  throw ae;
               }
               continue;
      }
      break;
}
```

In general, you release a change after it is signed off by CCB members. In addition to modifying the status of a change, you can also use `changeStatus()` to specify a Notification list, optional comments, and whether to notify the originator and change control board.

> **Note:** 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.

Depending on the overloaded `changeStatus()` method you use, the `notifyList` parameter is an array of `IUser` or `IUserGroup` objects that you must notify about the change in status. For more information, refer to the API Reference files at Oracle® E-Cloud Web site (`http://edelivery.oracle.com/`), navigate to `IRoutable` Interface Reference, and these methods are listed under Public Member Functions. In this case, point to and select the `IUser`. The Public Member Functions of this interface is shown in Part 16–3.

Figure 16–3 Member Functions of the IUser Interface



com.agile.api.IUser

IUser Interface Reference

Inheritance diagram for IUser:

List of all members.

Public Member Functions

void	changeLoginPassword (String oldPassword, String newPassword) throws APIException
void	changeApprovalPassword (String oldPassword, String newPassword) throws APIException
boolean	hasPrivilege (int privilegeId, Object object) throws APIException
IFolder	getHomeFolder () throws APIException
IFolder	getFolder (FolderConstants type) throws APIException

Static Public Attributes

static final int	OBJECT_TYPE = IObjectType.TYPE_USER

Detailed Description

A user dataobject. Users are dataobjects that you can create, like items and changes. Consequently, you can work with users directly without traversing Agile PLM's administrat
you have the proper privileges, you can create, modify, and delete users.

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 16–14 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 use the default approvers and observers when you change the status of a routable object.

Example 16–15 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
ISignoffReviewer[] defaultApprovers =
      change.getReviewers(nextStatus, WorkflowConstants.USER_APPROVER);
List<ISignoffReviewer> approverList =
      Arrays.asList(defaultApprovers);

// Get default observers for the next status
ISignoffReviewer[] defaultObservers =
      change.getReviewers(nextStatus, WorkflowConstants.USER_OBSERVER);
List<ISignoffReviewer> observerList =
      Arrays.asList(defaultObservers);

// Change to the next status
change.changeStatus(nextStatus, false, "", false, false, null, approverList,
observerList, null, 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 sends an object to all users.

Example 16–16 Sending an Agile object to selected users

```
public static void sendToAll(IDataObject object) {

// Get all user groups
   try {
    IQuery q =
    (IQuery)session.createObject(IQuery.OBJECT_TYPE, "select * from [UserGroup]");
   ArrayList usergroupList = new ArrayList();
   Iterator i = q.execute().getReferentIterator();
while (i.hasNext()) {
   usergroupList.add(i.next());
}
IUserGroup[] group = new IUserGroup[usergroupList.size()];
System.arraycopy(usergroupList.toArray(), 0, group, 0, usergroupList.size());

// Send the object to all user groups
object.send(group, "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 the `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 16–17 Sending an Agile object to selected user groups

```
public static void sendToAll(IDataObject object) {
// Get all user groups
   try {
    IQuery q =
      (IQuery)session.createObject(IQuery.OBJECT_TYPE, "select * from
        [UserGroup]");ArrayList usergroupList = new ArrayList();
   Iterator i = q.execute().getReferentIterator();
   while (i.hasNext()) {
       usergroupList.add(i.next());
   }
   IUserGroup[] group = new IUserGroup[usergroupList.size()];
   System.arraycopy(usergroupList.toArray(), 0, group, 0, usergroupList.size());

// Send the object to all user groups
   object.send(group, "Please read this important document.");
   } catch (APIException ex) {
    System.out.println(ex);
   }
}
```

Managing Functional Teams

The Functional Team feature streamlines the approval process by eliminating the need to update the list and roles of workflow approvers (the approval “criteria”). This is achieved by enabling the workflow process to create a Workflow Template and assign users to workflow actions. Users are then linked to the job functions they perform. Job functions and users are linked together to form the Functional Teams. These Functional Teams are added to routable objects and when they are routed through the workflow, the combination of Functional Team, Workflow Criteria, and the Approval Template determine which user performs the required workflow action.

Job Functions, Functional Teams, and Workflow Actions

You can use the SDK to eliminate updating the approval criteria in the event of a change in the approval criteria by defining the following Workflow-related objects and attributes:

- **Job Function** - The job performed by a user. Examples are product manager, product marketing manager, development lead, development manager, and so on.
- **Functional Team** - Group of users and/or user groups who work as a team to perform specific job functions. Logically, a Functional Team is a Job Function and not a User, because a Job Function can be associated with a Functional Team

without any Users, but a User can not be added to a Functional Team without an assigned Job Function.

- **Reviewers** - Based on the role, a Reviewer can be an Approver, an Acknowledger, or an Observer. A user can have more than one Reviewer role.

Note: Users can have more than one Reviewer role, but they cannot be Acknowledgers, Observers and/or Approvers.

  - **Observers** - A workflow action that requires the notified users to signoff when notified. Observers can be Users, User Groups or Functional Teams.
  - **Acknowledgers** - A workflow action that requires the notified users to signoff the change they have approved. This is different from an approval signoff. Acknowledgers can be Users, User Groups or Functional Teams.
  - **Approvers** - An approvers can sign -off for multiple job functions. Approvers can be Users, User Groups or Functional Teams.

Creating a Functional Team

Functional Teams are uniquely identified by name and they are a class under the User Group Base Class with the Functional Team as the default sub class. Creating a Functional Team object is similar to creating any Agile object. To create a Functional Team, use the `IAgileSession.createObject` method, specifying both the class and the name of the Team.

All users cannot create a Functional Team. Only users who have the Create privilege for this object can create Functional Teams.

Example 16–18 Creating a Functional Team

```
IAgileClass ft_class =
      session.getAdminInstance().getAgileClass("Functional Team");
HashMap<Integer, String> map = new HashMap<Integer, String>();

//Put values into map
  map.put(UserGroupConstants.ATT_GENERAL_INFO_NAME, "PGC");
  map.put(UserGroupConstants.ATT_GENERAL_INFO_DESCRIPTION, "PGC Functional Team");
  IDataObject FtObj = (IDataObject)session.getObject(UserGroupConstants.CLASS_
     FUNCTIONAL_TEAM, map);
   if (FtObj == null){

// Create Functional Team with this Map //
  FtObj = (IDataObject)session.createObject(ft_class, map);
}
```

Functional Teams is a class under User Group Base Class having out of the box sub class 'Functional Team'. Creating Functional Team object is similar to creating any Agile object as detailed below:

Managing Timesheets

PPM has the capability to manage the following Timesheet-related functions from the Web Client. The SDK enables performing the functions programmatically, by exposing the `ITimesheetManager`.

- **Retrieving tasks a user can report time against** - This feature enables retrieving the Web Client's Timesheet view and the list of tasks the user can report time against programmatically.
- **Logging or modifying time reported for a user on a task** - Similar to the Web Client, this feature supports logging or modifying time by day or activity for the user.
- **Retrieving hours for a user or activity** - This feature enables searching and r**etrieving hours for a user**, an activity, a project, the date in between, and Canceled and Soft-Deleted activities. Results are provided as CSV or XLS similar to the UI

Important: As the SDK developer, you must have the proper privileges to perform Timesheet-related tasks.

Retrieving Tasks Users Can Log Time Against

As the SDK developer with proper privileges, you can retrieve Timesheet records that display activities against which users can log time worked. The `ITimesheetManager` API returns the Timesheet table listing input dates for the week as shown the following example.

Function: `ITimesheetManager:: retrieveTimesheet(Date selectedDate)`

Example 16–19 Retrieving Timesheet table with week's input dates

```
//connect to Agile Server
   IAgileSession ses = connect();
   ITimesheetManager timesheetManager = (ITimesheetManager)
   ses.getManager(ITimesheetManager.class);
   Calendar cal = new GregorianCalendar(TimeZone.getTimeZone("GMT"));
   cal.set(Calendar.YEAR, 2013);
   cal.set(Calendar.MONTH, 2);
   cal.set(Calendar.DAY_OF_MONTH, 27);
// Retrieve a list of tasks the user can report time against
   ITable result = timesheetManager.retrieveTimesheet(cal);
```

Logging or Modifying the Reported Time for a User on a Task

After retrieving a Timesheet, the user can log or modify hours for the activity in the returned Timesheet records.

Function: `ITimesheetManager:: logOrChangeTimesheet(ITable tsTable)`

You can find detailed sample codes for `logOrChangeTimesheet()` in `samples\api\LogOrChangeTimesheet`, accessible from "Client-Side Components" on page 1-2.

Retrieving Hours for a Given User or Activity

This API enables searching the Timesheet by User, Project, Date Between, as well as Canceled and Soft-Deleted Activities. The search result table is exported as a CSV or XLS file to the specified device.

Function: `ITimesheetManager:: exportSearchedTimesheet (Map params)`

You can find detailed sample codes for exportSearchedTimesheet() in `samples\api\exportSearchedTimesheet`, accessible from "Client-Side Components" on page 1-2.

Creating a Job Functions Table

A Functional Team object contains a valid Job function and associated members such as users, user groups, and so on, in its Job Functions Table.

Example 16–20 Creating a Job Functions Table

```
//get Job Functions Table of the Functional Team Object
   ITable jobFuncTbl=FtObj.getTable(UserGroupConstants.TABLE_JOBFUNCTION);
   IDataObject usr0 =
      (IDataObject) session.getObject(UserConstants.CLASS_USER, "usr01");
   IDataObject usr1 =
      (IDataObject) session.getObject(UserConstants.CLASS_USER, "usr02");
   IDataObject uGp0 =
      (IDataObject) session.getObject
         (UserGroupConstants.CLASS_USER_GROUP, "usrGroup01");
   IRow ft_jf_row;
   try{
      map.clear();
      map.put(UserGroupConstants.ATT_JOB_FUNCTION_NAME, "Developer");
      //a valid value from 'Job Functioins' list
      map.put(UserGroupConstants.
         ATT_JOB_FUNCTION_USERS_USERGROUPS, new Object[] {usr0, usr1, uGp0});
   ft_jf_row = jobFuncTbl.createRow(map);
      map.clear();
      map.put(UserGroupConstants.ATT_JOB_FUNCTION_NAME, "Product Manager");
      //a valid value from 'Job Functioins' list
      map.put(UserGroupConstants.
         ATT_JOB_FUNCTION_USERS_USERGROUPS, new Object[] {usr1, uGp0});
   ft_jf_row = jobFuncTbl.createRow(map);
   }catch(APIException e){
      System.out.println(e.getRootCause());
   }
```

Adding a Discussion to a Functional Team

Discussions are integrated into the Product Portfolio Management solution, and are also applicable to the Product Cost Management solution. Discussions are created similar to other Agile PLM objects. Discussions are always viewed in the Web Client. To access a discussion while using Java Client, you can search to locate an existing discussion and open the object, which will open the Web Client. Also, you can open the Web Client and proceed from there to create a discussions.

Note: Discussions, Action Items, and News is a subset of Activity-specific subscription events that only apply to Product Portfolio Management and users can subscribe to actions related to Discussions, Action Items, and News.

The following example adds a Discussion to a Functional Team.

Example 16–21 Adding a discussion to a Functional Team

```
ITable ft_dc_tbl = (ITable)FtObj.getTable(UserGroupConstants.TABLE_DISCUSSIONS);
//Create a New discussion and add it to Functional Team.Discussion Table
   IAgileClass discClass =      session.getAdminInstance()
      .getAgileClass(DiscussionConstants.CLASS_DISCUSSION);
   String discNmbr = "D00003";
   HashMap<Integer, String> map = new HashMap<Integer, String>();
      map.put(DiscussionConstants.ATT_COVER_PAGE_NUMBER, discNmbr);
      map.put(DiscussionConstants.ATT_COVER_PAGE_SUBJECT,
         "Testing Func Team: "+discNmbr );
   IDiscussion disc =
      (IDiscussion)session.getObject(DiscussionConstants.CLASS_DISCUSSION, map);
   if (disc == null){
      disc = (IDiscussion)session.createObject
         (DiscussionConstants.CLASS_DISCUSSION, map);
   }

//Add Discussion to FunctionalTeam.Discussions Table
   map.clear();
   map.put(UserGroupConstants.ATT_DISCUSSIONS_NUMBER, disc.getName());
   map.put(UserGroupConstants.ATT_DISCUSSIONS_NUMBER, disc);
   IRow newRow = ft_dc_tbl.createRow(map);
```

Assigning Actions Items to Functional Teams

Action items are routable objects that require an action by the assigned user or user group. They inform the assigned users of a request for an action or activity. For a list and description of potential users, see "Job Functions, Functional Teams, and Workflow Actions" on page 16-13. The following example shows how to assign action item using the SDK and the various fields such as dates, assignee, the action, and so on.

Note: In Agile Web Client, Action Items is a sub-level tab of the Discussions tab.

Example 16–22 Assigning Action items to Functional Teams

```
//Get FunctionalTeam.ActionItems Table
   ITable ft_ai_tbl = FtObj.getTable(UserGroupConstants.TABLE_ACTION_ITEMS);
//Add an Action Item to FunctionalTeam.ActionItems Table
   SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy");
   Date dueDate = df.parse((String) "11/18/2011");
   HashMap<Integer, String> map = new HashMap<Integer, String>();
   IDataObject usr0 =
      (IDataObject) session.getObject(UserConstants.CLASS_USER, "usr0");
   map.put(UserGroupConstants.ATT_ACTIONITEM_SUBJECT,
      " ActItm for: "+usr0.getName());
   map.put(UserGroupConstants.ATT_ACTIONITEMS_ASSIGNEDTO, usr0);
   map.put(UserGroupConstants.ATT_ACTIONITEM_DUEDATE, dueDate);
   IRow newRow1 = ft_ai_tbl.createRow(map);
   IDataObject usr1 =
      (IDataObject) session.getObject(UserConstants.CLASS_USER, "usr1");
      map.put(UserGroupConstants.ATT_ACTIONITEM_SUBJECT,
         " ActItm for: "+usr.getName());
      map.put(UserGroupConstants.ATT_ACTIONITEMS_ASSIGNEDTO, usr1);
   IRow newRow2 = ft_ai_tbl.createRow(map);
```

Updating a Functional Team's News Table

As information passes through the Agile system, users receive news of status changes, requests, and notifications. Similar to Action Items, News is a sub-level tab of the Discussions tab, described earlier in the **Note** in "Adding a Discussion to a Functional Team" on page 16-16.

The following example shows the various fields and how to use the SDK to update the applicable data elements.

Example 16–23 Updating a Functional Teams' News Table

```
//Get the Functional Team.News Table
   ITable ft_ns_tbl = FtObj.getTable(UserGroupConstants.TABLE_NEWS);
//Add new News to Functional Team.News Table
   IAgileList newsTypeLst =
      session.getAdminInstance().getListLibrary().getAdminList
         (AdminListConstants.LIST_NEWS_TYPE).getValues();
   newsTypeLst.setSelection(new Object[]{"Information"});
   HashMap<Integer, String> map = new HashMap<Integer, String>();;
   map.put(UserGroupConstants.ATT_NEWS_TYPE, newsTypeLst);
   map.put(UserGroupConstants.ATT_NEWS_TITLE, "Publishing News Title ");
   map.put(UserGroupConstants.ATT_NEWS_NEWS,
      "This is testing the news publishing feature" + System.currentTimeMillis());
   ft_ns_tbl.createRow(map);
```

17 Managing and Tracking Quality

This chapter includes the following:

- 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 enables 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, NCRs, CAPAs, and Audits, 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, and NCRs.
- **Quality Administrator** - role for users who manage audits and CAPAs.

For more information about roles and privileges, refer to the *Agile PLM Administrator Guide*.

Working with Customers

This section describes how to create, load, and save ICustomer objects.

About Customers

The QualityChangeRequest object stores contact information about a customer. What role does a customer have in the Agile PLM system? Customers provide feedback on your company's products, alerting you to quality issues or problems they encounter.

This object can originate in another system, such as a CRM system. You can use the Agile API to import customer data and problem reports from CRM systems into the Agile PLM system.

Creating a Customer

To create a customer, use the IAgileSession.createObject() method. At a minimum, you should specify values for the General Info.Customer Name and General Info.Customer Number attributes.

Example 17–1 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 17–2 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);
}
```

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 17–3 Saving a customer as another customer

```
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. The following example shows how to create problem reports and NCRs.

Example 17–4 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;
}
```

Assigning a Product Service Request to a Quality Analyst

To assign a problem report or NCR to a Quality Analyst, set the value for the Cover Page | Quality Analyst field, which is a list field. The available values for the list field consists of Agile PLM users. The following example shows how to set the value for the Cover Page.Quality Analyst field for a problem report or NCR.

Example 17–5 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 17–6 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 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(ServiceRequest
      Constants.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. The single PSR you create is the Parent PSR. All the added PSRs on the Related PSR tab are child PSRs.

Note: SmartRules settings control whether a PSR can be associated with both affected items and related PSRs, or only affected items or related PSRs. Depending on enabling/disabling the "PSR contains Items and Related PSRs" SmartRule setting, you can enable/disable the Affected Items tab after you add Related PSRs. For information on SmartRules and PSRs, refer to *Agile PLM Product Quality Management User Guide* and *Agile PLM Administrator Guide*. The following example assumes SmartRules are set properly.

Example 17–7 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 a corrective or preventive action (CAPA). This may result in changes to a product, process, or supplier by initiating an ECO or MCO. QCRs also provide an audit trail between problems, corrective and preventive actions, and engineering changes.

Agile PLM provides two classes of Quality Change Requests:

- **CAPA** - Stands for Corrective Action/Preventive Action, which addresses defects that (generally) surfaced from problem reports. By the time a problem reaches the CAPA stage, the team has figured out which specific items must be fixed. Consequently, the affected item for a CAPA may be different from the affected item of its related problem report. For example, say a customer reported a problem with a DVD-ROM drive. A CAPA is initiated and the root-cause is identified to be a defect in the IDE controller. Therefore, the CAPA and its related problem report have different affected items.
- **Audit** - Systematic, independent and documented processes for obtaining evidence and evaluating it objectively to determine the extent to which criteria are

fulfilled. Audits can be initiated against items for which no problems have been reported.

Creating a Quality Change Request

To create a QCR, use the IAgileSession.createObject() method. The only required attribute value you must specify is the object's number. The example below shows how to create both CAPA and Audit QCRs.

Example 17–8 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;
}
```

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 17–9 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
   IAgileList values = cell.getAvailableValues();

//Set the value to the current user
   IUser user = m_session.getCurrentUser();
   values.setSelection(new Object[] { user });
   cell.setValue(values);
}
```

Saving a Quality Change Request as a Change

You can use the IDataObject.saveAs() method to save a QCR as another QCR or as an ECO (or another type of change order). When you save a QCR as an ECO, the items affected by the QCR are not automatically transferred to the Affected Items tab of the ECO. If you want to transfer affected items from the QCR to the ECO, you must write the code in your program to provide that functionality. Workflow is a required input parameter for using saveAs() on QCRs.

Note: If you try to save a QCR to an object that is not a subclass of either the Quality Change Request or Change superclasses, the Agile API throws an exception.

Example 17–10

```
public IChange saveQCRasECO(IAgileSession session, IQualityChangeRequest qcr)
   throws APIException {

// Get the ECO class
   IAgileClass cls = m_admin.getAgileClass(ChangeConstants.CLASS_ECO);

// Get autonumber sources for the ECO class
   IAutoNumber[] numbers = cls.getAutoNumberSources();

// Get Workflow for the ECO class
   IWorkflow ecoWf =((IRoutableDesc)session.getAdminInstance().getAgileClass
      (ChangeConstants.CLASS_ECO)).getWorkflows()[0];

// Save the QCR as an ECO
   HashMap map = new HashMap();
      map.put(ChangeConstants.ATT_COVER_PAGE_NUMBER, numbers[0]);
      map.put(ChangeConstants.ATT_COVER_PAGE_WORKFLOW, ecoWf);
      IChange eco = (IChange)qcr.saveAs(ChangeConstants.CLASS_ECO, map);

// Add code here to copy affected items from the QCR to the ECO
   return eco;
}
```

Using Workflow Features with PSRs and QCRs

PSRs and QCRs derive all Workflow functionality from the IRoutable interface. The following table lists the Workflow commands you can use to manage product quality objects.

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

Add or remove approvers for a PSR or QCR	IRoutable.addApprovers()
	IRoutable.removeApprovers()

Selecting Workflows for PSRs and QCRs

When you create a new Product Service Request or a Quality Change Request, you must select a workflow. Your Agile PLM system can have multiple workflows defined for each type of Product Service Request and Quality Change Request. To retrieve the valid workflows for an object, use IRoutable.getWorkflows(). If a Workflow has not been assigned yet, you can use IRoutable.getWorkflows() to select a workflow, as shown in the following example.

Example 17–11 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 which is a null object
* because  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 17–12 Selecting a Workflow by setting the value of Cover Page.Workflow

```
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
```

18 Managing Product Governance & Compliance

This chapter includes the following:

- About Agile Product Governance & Compliance
- Agile PG&C Interfaces and Classes
- Agile PG&C Roles
- Creating Declarations, Specifications, and Substances
- Creating Substances
- Adding Items, Manufacturer Parts, and Part Groups to Declarations
- Adding Substances to Declarations
- Adding Substances to a Specification
- Adding Specifications to a Declaration
- Routing Declarations
- Completing a Declaration
- Submitting Declarations to Compliance Managers
- Adding Substance Compositions for Manufacturer Parts
- Rolling Up Compliance Data

About Agile Product Governance & 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 enables PLM users to cost-effectively comply with environmental regulations. they can use Agile PG&C to obtain compliance data for parts from their suppliers. This enables PLM installations 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 the installation's products adhere to government regulations and company policy. As the Material Provider, the supplier completes and signs off on material declarations, thereby disclosing which hazardous substances are contained within the components and subassemblies it provides. For a more detailed overview of Agile PG&C features, refer to the Product Governance & Compliance User Guide

Agile PG&C Interfaces and Classes

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

Table 18–1 List of PG&C-related interfaces and classes

Object	Interface	Constants Class
Declaration	`IDeclaration`	`DeclarationConstants`
Specification	`ISpecification`	`SpecficationConstants`
Substance	`ISubstance`	`SubstanceConstants`
Part Groups	`ICommodity`	`PartGroupConstants`

Items, Manufacturer Parts, and Part Groups are objects that are also related to Agile PG&C. They have Specifications, Compositions, and Substances tables that are populated with data when declarations are released. For Manufacturer Parts, you can edit the Compositions and Substances tables directly without submitting a declaration.

Note: The terms "part group" and "commodity" are used interchangeably in this guide to refer to any `ICommodityobject` where `ICommodity` represents the Part Group base class, which includes the Commodity and Part Family subclasses.

Also, other common Agile API interfaces, such as `ITable, IDataObject`, and `ICell`, are also used to work with Agile PG&C objects.

Agile PG&C Roles

Agile PLM provides two out-of-the-box roles designed for Agile PG&C users:

- **Compliance Manager** - This provides privileges needed to create and manage Agile PG&C objects, such as Declarations, Substances, and Specifications, and run Agile PG&C reports. Compliance Managers are responsible for routing material declarations to suppliers.
- **(Restricted) Material Provider** - This provides privileges needed to create and modify declarations, as well as read all other types of Agile PG&C objects. This role is typically assigned to supplier users, who have restricted access to the Agile PLM system. Material Providers are responsible for completing and signing off on material declarations.

To use Agile PG&C APIs mentioned in this chapter, make sure you log in as a user assigned either the Compliance Manager role, or the (Restricted) Material Provider role. For more information about Agile PLM roles, refer to the *Agile PLM Administrator Guide*.

Note: The *Discover Change* privilege mask is not included in the *Compliance Manager* role. If you only have the Compliance Manager role, then you do not have sufficient privileges to use the API to set the calculated compliance of a part in a Declaration, and pass the *Change Number* to the SDK call. To pass the Change Number in the SDK call, you must have the *Discover Change* privilege mask for that object in the Change Orders class. For more information, see Setting Values in the Calculated Compliance Field for Declaration Objects.

Creating Declarations, Specifications, and Substances

The following paragraphs provide definitions and procedures to define and manage these PG&C classes.

Creating Declarations

A Declaration object is the main record of Agile PG&C. It tracks the substances and materials that are used for items, manufacturer parts, and part groups. When you release a declaration, the information gathered from it is published to the product record, thereby updating the Composition data contained within the items, manufacturer parts, and part groups listed by the declaration.

There are seven declaration subclasses provided with Agile PLM:

- Homogeneous Material Declaration - A homogeneous material composition declaration that uses material-level specifications.
- IPC 1752-1 Declaration - A material composition declaration for electronic products that conforms to IPC standards and uses only one part-level specification.
- IPC 1752-2 Declaration - A homogeneous material composition declaration for electronic products that conforms to IPC standards and uses only one material-level specification.
- JGPSSI Declaration - A material composition declaration that follows the Japanese Green Procurement (JGP) standard and uses part-level specifications.
- Part Declaration - A material composition declaration that uses part-level or material-level specifications.
- Substance Declaration - A material composition declaration for each substance within part-level specifications.
- Supplier Declaration of Conformance - A questionnaire to assess supplier compliance with specifications from customers and government agencies. The survey addresses compliance at a general company level. Can be used for CSR type declarations.

The procedure for creating a declaration is the same for all declaration subclasses. You must specify the declaration subclass as well as values for the Cover Page.Name and Cover Page.Supplier attributes. Other declaration attributes are optional.

By default, the Cover Page.Name field uses an Autonumber format with the prefix “MD” (for “Material Declaration”). Although the Autonumber format is not required, it is a good practice 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. For more information about defining and modifying character sets, Refer to the *Agile PLM Administrator Guide*.

Supplier users with the (Restricted) Material Provider role can also create declarations. However, in this case, only the Cover Page.Name attribute is required to create the object. The CoverPage.Supplier attribute is filled in automatically with the user's supplier organization.

The following example shows how to create a Substance declaration.

Example 18–1 Creating a Substance Declaration

```
public void CreateSubstanceDeclaration(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 Substance Declaration subclass
   IAgileClass declClass = m_session.getAdminInstance().getAgileClass(
   DeclarationConstants.CLASS_SUBSTANCE_DECLARATION);

// Create a new Substance declaration
   IDeclaration object = (
      IDeclaration)m_session.createObject(declClass, params);
}
```

Creating Specifications

Specifications are used to state the criteria that a product is expected to meet or exceed. They are generally used to limit the amount of restricted substances contained in a product. Specifications can be internal documents issued by a company or industry, or, more commonly, they are regulations issued by a governing body. Here are some examples of government regulations:

- Restrictions on the Use of Certain Hazardous Substances in Electrical and Electronic Equipment (RoHS) Directive, issued by the European Union
- Waste Electrical and Electronic Equipment (WEEE) Directive, issued by the European Union
- Food Allergen Labeling and Consumer Protection Act (FALCPA), issued by the U.S.A. Food and Drug Administration (FDA)

A specification defines a list of substances, the parts-per-million (PPM) threshold for each substance, and whether a particular substance is restricted. Compliance Managers can use specifications to pre-populate material declarations with appropriate substances to ensure compliance.

The only required attribute you must specify when you create a specification is General Info.Name. The name must be unique. The name is case-insensitive, which means “ROHS” is treated the same as "Rohs".

The General Info.Validation Type attribute is important because it determines whether the specification is Part Level (the default) or Homogeneous Material Level, which affects the types of declarations that can be used with the specification. Another optional attribute is General Info.Lifecycle Phase. When you create a specification, the default lifecycle phase is Active. To make the specification obsolete, change the value of its lifecycle phase attribute to Obsolete.

Example 18–2 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.

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

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 18–3 Creating a substance

```
public void createSubstance(String num, String casNumber)
   throws Exception {

// Create a Map object to store parameters and initlize the params object.
   Map params = new HashMap();
   params.put(SubstanceConstants.ATT_GENERAL_INFO_NAME, num);
   params.put(SubstanceConstants.ATT_GENERAL_INFO_CAS_NUMBER, casNumber);

// Get the Substance subclass
   IAgileClass subsClass =
      m_session.getAdminInstance().getAgileClass
         (SubstanceConstants.CLASS_SUBSTANCE);

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

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 18–4 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 subsClass =
      m_session.getAdminInstance().
         getAgileClass(SubstanceConstants.CLASS_SUBSTANCE_GROUP);

// Create a new Substance Group
   ISubstance sub = (ISubstance)m_session.createObject(subsClass, 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 18–5 Creating a material object and adding substances to the new 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 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 18–6 Creating a subpart

```
public void createSubpart(String num) throws Exception {

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

// Initialize the map object
   params.put(SubstanceConstants.ATT_GENERAL_INFO_NAME, num);

// Get the Subpart subclass
   IAgileClass subsClass =
      m_session.getAdminInstance()
         .getAgileClass(SubstanceConstants.CLASS_SUBPART);

// Create a new Subpart
    ISubstance sub =
      (ISubstance)m_session.createObject(class, params);
}
```

Adding Items, Manufacturer Parts, and Part Groups to Declarations

Each declaration has separate tables for items, manufacturer parts, and part groups. Each of these also has an associated composition table: Item Composition, Manufacturer Part Composition, and Part Group Composition.

When you add an item to the Items table of a declaration, the latest released revision of the item is used. If the item does not have a released revision, the Introductory revision is used.

The following example shows how to add items, manufacturer parts, and part groups to a declaration.

Example 18–7 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 Declarations

You can add substances to the Item Composition, Manufacturer Part Composition, and Part Group Composition tables contained within a declaration. To publish substances into items, manufacturer parts, and part groups, you release the declaration. When the declaration is released, the substances get added automatically to the Substances tables of the corresponding items, manufacturer parts, and part groups.

To add a row to the composition tables of a declaration, use the ITable.createRow() method. Because the composition tables are mapping tables, you cannot pass an ISubstance object to create the row. Instead, specify a Map object containing attribute-value pairs.

Important: The Substances and Composition tables for items and part groups are read-only. They get populated with data only when declarations are released

To add a substance to one of the Composition tables of a declaration:

1. Add an item, manufacturer part, or part group to the Items, Manufacturer Parts, or Part Groups tables of a declaration, respectively.
2. Add a substance row to the Composition table that references the parent row on the Items, Manufacturer Parts, or Part Groups table. Use the virtual attribute DeclarationConstants.ATT_PARENT_ROW to specify the parent row. When you add a substance, specify the substance name and substance type.

Important: For the Agile SDK, Composition tables for declarations list all parent objects contained in the Items, Manufacturer Parts, and Part Groups tables. Agile Web Client represents Composition tables differently. It shows a separate Composition table for each parent object.

When you create a row in the Composition tables, you pass a Map object containing attribute-value pairs. The following table lists the attributes the Map object must contain:

Table 18–2 Required attributes in the Map objects

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`

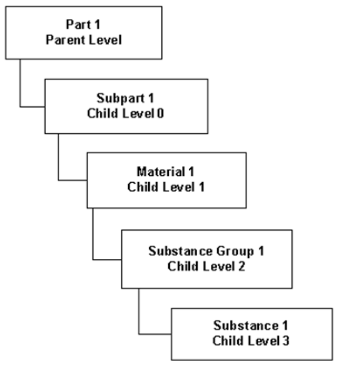
Part Group Composition	Part Group Row Substance Name	`ATT_PARENT_ROW`
		 `ATT_PART_GROUP_COMPOSITION_SUBSTANCE_NAME`

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.

Figure 18–1 Hierarchy for a Bill of Substances (Composition) with four child levels



Rules for Adding Substances

Follow these rules when adding substances to a Composition table:

- Parent objects must be added before their children.

- Subparts can have the following children: other Subparts, Materials, Substance Groups, or Substances.
  - A Subpart cannot contain Subparts, Materials, Substance Groups, and Substances all at the same level.
  - A Subpart can contain other Subparts and Material at the same level.
  - A Subpart can contain Substance Groups and Substances at the same level.
- Material can have the following children: Substance Groups or Substances.
- Substance Groups can have the following children: Substances only.

Adding Subparts and Materials that Do Not Exist

When you add substances to a Composition table of a declaration, you can specify “dummy” subparts and materials that do not exist in the Agile PLM system. Such subparts and materials will be visible only within the Composition table. When you add “dummy” subparts and materials to the Composition table, you must specify the Substance Type attribute:

- ATT_ITEM_COMPOSITION_SUBSTANCE_TYPE
- ATT_MANUFACTURER_PART_COMPOSITION_SUBSTANCE_TYPE
- ATT_PART_GROUP_COMPOSITION_SUBSTANCE_TYPE

The following example shows how to add a dummy subpart or material to the Manufacturer Part Composition table. Because the Substance Type field is a list field, the value passed for it is an IAgileList.

Example 18–8 Adding dummy subpart/material to Manufacturer Part Composition

```
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);
   }
}
```

Examples of Adding Substances

The following examples show how to add:

- Substances to the Manufacturer Part Composition Table of a Homogeneous Material Declaration
- Substances to the Manufacturer Part Composition Table of a Substance Declaration

Adding Substances to Manufacturer Part Composition Table of Homogeneous Material Declarations

The following example shows how to add substances to a Manufacturer Part Composition table of a Homogeneous Material Declaration. The table has four levels: subparts, materials, substance groups, and substances. When you add a substance row to the table, we recommend that you pass a substance object (ISubstance) instead of a substance name (String) as the input parameter.

Example 18–9 Adding Homogeneous Material Level substances to 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 Homogeneneous Material Level");
   IRow rowSpec = tblSpec.createRow(spec);

// Add a Manufacturer Part to the Manufacturer Parts table
   ITable tblMfrParts =
      dec.getTable(DeclarationConstants.TABLE_MANUFACTURERPARTS);
   params.clear();
   params.put(DeclarationConstants.
      ATT_MANUFACTURER_PARTS_MFR_PART_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);
   }
}
```

Adding Substances to Manufacturer Part Composition Table of Substance Declarations

The following example shows how to add substances to a Manufacturer Part Composition table of a Substance Declaration. The table has two levels: substance groups and substances.

Example 18–10   Adding Part Level substances to 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 Compounds");
      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 the Substances table of a 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 18–11 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 to the ensure that suppliers comply with any restrictions stated in the specifications.RU

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 18–3 Declaration types and specification requirements

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

Table 18–3 (Cont.) Declaration types and specification requirements

Declaration Type	Supported Specification Validation Types
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.

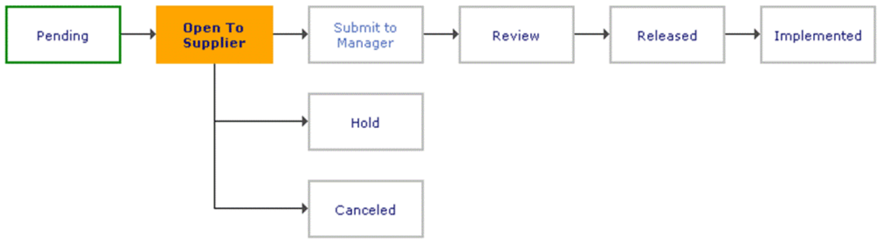
Example 18–12 Adding specifications to the Specification table

```
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 Figure 18–2.

Figure 18–2 Default Declarations workflow



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

Table 18–4 Default Declarations workflow status

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.

Table 18–4 (Cont.) Default Declarations workflow status

Status	Description
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 18–13 Setting values for 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 18–14 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 from Open to Supplier status
   dec.changeStatus(status, false, null, false, false, null, null, null, false);
}
```

For more information about Agile APIs that support Workflow processes, see Chapter 17, "Managing and Tracking Quality."

Completing Declarations

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 and Declared PPM fields or the Declared Compliance field for every substance listed on the Item Composition, Manufacturer Part Composition, and Part Group Composition tables, particularly for substances that are restricted by specifications.
- Complete other flex fields on the Composition tables as necessary.
- Add or remove substances from the declaration.

When the declaration is complete, the Material Provider user can sign off and submit the declaration to the Compliance Manager, described below.

Submitting Declarations to Compliance Managers

When the supplier changes the status of the declaration from “Open to Supplier” to “Submit to 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 18–15 Signing off and submitting declarations to Compliance Manager

```
public void submitToCM(IDeclaration dec) throws Exception {

IStatus status = null;

// Get the Submit to 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 Submit to Manager status (signoff password is "agile")
   dec.changeStatus(status, false, null, false, false, null,
      null, null, false, "agile");
}
```

Publishing a Declaration

The Agile API does not provide a method to publish a material declaration to the product record. Instead, a declaration is automatically published when it is released. Therefore, as far as the API is concerned, the substances table for an item, manufacturer part, or part group always reflects the last released declarations. However, Agile Web Client allows you to select an later declaration and publish it, thereby updating the substances information in the product record.

Getting and Setting Weight Values

Unit of Measure fields in Agile PLM 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`

Converting an Object's Unit of Measure to a Different Unit of Measure

This conversion is explained with the aid of the following example which converts an object's weight from Kilograms to Grams. You can use this example to convert the unit of measure that measured the volume of the object, or its height.

Note: This example converts the unit of measure in the source object from 1234.21 Kilograms to its equivalent weight in Grams. Thus, convert From is the source number you want to convert, Kilogram is source unit of measure, and Gram is that target unit of measure which are all randomly selected for this example.

Example 18–16 Converting the Unit of Measure from Kilogram to Gram

```
try {
   double convertFrom = 1234.21;
   IUnitOfMeasureManager uomManager =
      (IUnitOfMeasureManager) session.getManager(IUnitOfMeasureManager.class);
   IUnitOfMeasure fromUOM =
      uomManager.createUOM(0, "Kilogram");
   IUnitOfMeasure toUOM = uomManager.createUOM(0, "Gram");
   IUnit fromUnit = fromUOM.getUnit();
   IUnit toUnit = toUOM.getUnit();
   double conversionFactor = fromUnit.getConversionFactor()
      toUnit.getConversionFactor();

// This example returns 1000/1 = 1000
   double convertedValue = convertFrom*conversionFactor;
   } catch (APIException ex) {
           System.out.println(ex);
}
```

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.

> **Note:** 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 Title Block.Mass` field of an item.

Example 18–17 Getting and setting the mass (weight) value for an item

```
private IUnitOfMeasure getMassValue(IItem item) throws APIException {
   IUnitOfMeasure uom =
      (IUnitOfMeasure)item.getValue(ItemConstants.ATT_TITLE_BLOCK_WEIGHT);
   System.out.println("Value: " + uom.getValue());
   System.out.println("Unit: " + uom.getUnit().toString());
   return uom;
}
private void setMassValue(IItem item, double value, String unit)
      throws APIException {
   IUnitOfMeasure uom = null;
   IUnitOfMeasureManager uomm =
      (IUnitOfMeasureManager)m_session.getManager(IUnitOfMeasureManager.class);
   uom = uomm.createUOM(value, unit);
   item.setValue(ItemConstants.ATT_TITLE_BLOCK_WEIGHT, uom);
   System.out.println("Value: " + uom.getValue());
   System.out.println("Unit: " + uom.getUnit().toString());
}
```

If you create a query to search for items by mass, only the numeric value is searched, not the unit. The server converts mass values to the standard unit before returning query results. For example, the following query returns all items whose mass value is between1.0 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 18–18 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();
```

Adding Substance Compositions for Manufacturer Parts

With appropriate privileges, you can modify the Specifications, Compositions, and Substances tables of a manufacturer part directly without submitting a declaration. This feature is useful for manufacturing partners that want to specify composition information for their parts. To add a row to the Specifications, Compositions, and Substances tables, use the ITable.createRow(Object) method.

Note: Once a row has been added to the Compositions and Substances tables of a Manufacturer Part, you cannot update or remove it.

The procedure for adding rows to the Substances table of a Manufacturer Part is similar to the way you add rows to the composition tables for a declaration. Follow these steps to add substance compositions into a manufacturer part:

To add rows to Substances table of 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 18-9..

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 18–19 Adding specification, composition, and substances to Manufacturer Part

```
public void addMfrPartSubs(IAgileSession m_session) throws APIException {

// Create a Manufacturer Part
try {
   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");

// Add Spec
   IRow rowSpec = tblSpec.createRow(spec);

// Get the Compositions table and Set as Homogeneous Material Level
   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);
   }
}
```

Rolling Up Compliance Data

After gathering compliance data for items, manufacturer parts, and part groups, compliance managers review the completed declarations to determine if the data is ready for publication into the product record. Once declarations are published with the data written through to parts and part groups on BOMs, compliance managers must examine and test BOMs to ensure the assemblies and products are compliant. This process is called compliance validation and is fulfilled through compliance rollups. Rollups are built into the system. They are easy to use and rollup results are available on the UI. For more information on rolling up compliance data and the business logic behind this process, refer to the *Agile Product Governance & Compliance User Guide.*

The SDK supports calling the PG&C Rollup function on the server side. This is the same rollup function that is called by the UI. The IPGCRollup interface supports this feature.

Understanding the IPGCRollup Interface

The `IPGCRollup` interface provides the following methods to support rolling up compliance data:

- `rollup()`
- `rollup(Date)`

One of these methods has no parameters and the other has *Date* as a parameter. The Date parameter in the rollup API is used by the system to set the timestamp for the rollup, when it is done.

Example 18–20 IPGCRollup methods

```
public interface IPGCRollup {
   public void rollup() throws APIException;
   public void rollup(Date rollupDate) throws APIException;
}
```

Note: After invoking `rollup(Date)`, it is necessary to call `IDataObject.refresh()` to make sure the rollup function has taken effect. Otherwise, the system will display the results obtained in the previous rollup if the timestamp of the recent rollup is the same as the Date parameter.

Passing the Date Parameter

If you do not pass the date, the system will use the current time provided by the system. When a rollup is performed on a set of items, if the timestamp of the recent rollup on an item is the same as the passed Date parameter, the system will not repeat the rollup process on that item. Instead, it will display the results obtained in the previous rollup. You may want to use this date feature if there is a large number of items to rollup and you want to use the SDK to call all of them. In this case, you will get the current date first, and then the pass that date for the subsequent SDK `Rollup(Date)` call. For example, you want to use the SDK to roll up data for Assembly 1 and Assembly 2. In this case, the SDK is called twice. The first instance, to roll up data for Assembly 1, and the second instance, to rollup data for Assembly 2. With the date parameter already inside the rollup when performing the rollup on Assembly 2, the system will reuse the previous rollup data obtained for Item1.

```
Assembly 1
   Item1
   Iitem2

Assembly 2
   Item1
   Item3
```

Using the IPGCRollup Interface

The following examples roll up the assembled data on Items and Manufacturer Parts:

- Item (latest released ECO or MCO)
- MPN (latest released ECO or MCO)

Rolling Up Assembled Data on Items

This example calls an API using the SDK to identify the top level parent of a given *Item* (its latest released ECO or MCO). Next, it will call the rollup API on the top level parent returned by the previous API to ensure the assemblies and products are compliant.

Example 18–21 Identifying the top level parent for an Item

```
public void itemRollup(String itemStr) throws Exception{
   try {
   IItem item =(IItem)m_session.getObject(IItem.OBJECT_TYPE, itemStr);
   IQuery query = (IQuery)m_session.createObject
      (IQuery.OBJECT_TYPE, ItemConstants.CLASS_ITEM_BASE_CLASS);

//IQuery query = (IQuery)
   m_session.createObject(IQuery.OBJECT_TYPE, ItemConstants.CLASS_PART);
      query.setSearchType(QueryConstants.WHERE_USED_TOP_LEVEL);
      query.setCriteria("[1001] Equal To '"+item.getName()+"'");

//
```

```
   query.setCriteria("["+SDKWrapper.getString("TITLE_BLOCK") +"."+SDKWrapper
      .getString("IQuery_Number")+"] Equal To '"+item.getName()+"'");
   ITable results=query.execute();
      if (results.size() > 0) {
         Iterator it = results.getReferentIterator();
      if (it.hasNext()) {
         IItem obj = (IItem)it.next();
         IItem tlaItem = (IItem)m_session.getObject
            (IItem.OBJECT_TYPE, obj.getName());
      tlaItem.rollup();
   }
   }
      else {
      item.rollup();
   }
   } catch (APIException e) {
         throw e;
   }
   return;
}
```

Setting Values in the Calculated Compliance Field for Item Objects

Use the following API to set the value of the *Calculated Compliance* field on the Specifications table of `Item` and `ManufacturerPart` objects:

```
Public void setCalculatedComplianceForPartSpec(Object specName, Object
complianceEntryValue) throws APIException
```

In this API, the `specName` parameter is the name of the Specification object, and the complianceEntryValue parameter is the actual value of the Calculated Compliance field, which can be any entry in the Calculated Compliance list. Both parameters are of type String.

When this value is set by the SDK Client, it is never overwritten during the Rollup. This API allows users to set the calculated compliance value based on their own defined logic, instead of using the system's default logic.

Example 18–22   Setting the value of the Calculated Compliance field for Item objects

```
// COMPLIANT is the actual value of the Calculated Compliance field which shows
// the Specification is compliant or not, based on the customized calculated
// compliance result.
   String COMPLIANT = "Compliant";

// spec_num is the Specification Name in Item object's Specification Table
   String spec_num = row.getValue
      (ItemConstants.ATT_SPECIFICATIONS_SPECIFICATION).toString();
   item.setCalculatedComplianceForPartSpec(spec_num, COMPLIANT);
```

Setting Values in the Calculated Compliance Field for Declaration Objects

This is similar to the previous API that enabled setting the *Calculated Compliance* field for Item objects. You can use this API to set the value of the Calculated Compliance field in Item table and Manufacturer Part table for Declaration objects.

```
Public void setCalculatedComplianceForMDOPartSpec (Object partName, Object
partClassName, Object changeNumber, Object specName, Object
complianceEntryValue)) throws APIException
```

The system recognizes that the SDK Client has set this value and will use the new setting in the subsequent response during Rollup. In this API, the parameter `changeNumber` is optional. When the Declaration object has only one revision of an item, you can set the value of `changeNumber` to null. If the Declaration object has more than one revision of an item, you must set the value of `changeNumber` for the proper execution of the API.

Similar to the previous API, when this value is set by the SDK Client, it is never overwritten during the Rollup within the declaration. This API allows users to set the calculated compliance value based on their own defined logic, instead of using the system's default logic.

Note: If the you intends to pass the `changeNumber` field to `setCalculatedComplianceForDeclarationPartSpec()`, you must have the Discover Change privilege mask to make this change.

Example 18–23 Setting values for Calculated Compliance field of Declaration objects

```
// complianceValue -- This is the customized calculated compliance
   value and shows if the part is compliant to a Spec
      String ComplianceValue = "Compliant";

// partName is the Item/Mfr Part name in Declaration’s Item/MfrPart table.
// If this is a mfr part, it will resemble this “MfrName::MfrPartName”
      String partName =”P00001”;
      String partClassName = “Parts”;

// If the added part in Declaration is an Item, the changeName should be the
// Change number corresponding to the Item’s revision.
// If the added part in Declaration is a Mfr Part, the changeName should be “null”
   String changeName = “C00001”;

// spec_num is the Specification Name in Declaration object’s Specification Table
   String specName = “Rohs”;
   Declaration.setCalculatedComplianceForDeclarationPartSpec
      (partName, partClassName, changeName, specName, complianceValue);
```

Setting Values in Calculated Overall Compliance Fields of Item and Manufacturer Part Objects

The SDK supports this Compliance by exposing the `setCalculateOverallCompliance` API.

`public void` **`setCalculateOverallCompliance`** `() throws APIException;`

The following examples use this API to set values the Overall Compliance field of an Item and a Manufacture Part.

Example 18–24 Setting values for Calculated Overall Compliance field of an Item object

```
Private static void test_OverallComplianceofItem (IAgileSession session) {
   Try {

// Load an existing Item.
   IItem item = (IItem) session.getObject (ItemConstants.CLASS_PART, "P00007");
   item. SetCalculateOverallCompliance ();
   } catch (APIException e) {
      e.printStackTrace ();
   }
}
```

Example 18–25 Setting values for Calculated Overall Compliance field of Manufacture Part objects

```
private static void OverallComplianceOfMfrpart(IAgileSession session) {
   HashMap params = new HashMap();
   params.put(ManufacturerPartConstants.
      ATT_GENERAL_INFO_MANUFACTURER_PART_NUMBER, "mfrpart001");
   params.put(ManufacturerPartConstants.
      ATT_GENERAL_INFO_MANUFACTURER_NAME, "ACT");

   try {
   IManufacturerPart mfrPart = (IManufacturerPart) session.getObject
      (ManufacturerPartConstants.CLASS_MANUFACTURER_PART, params);
   mfrPart.setCalculateOverallCompliance();
   } catch (APIException e) {
         e.printStackTrace();
   }
}
```

Modifying the Calculated Compliance Field in Specification Table of Part, Manufacturer Part, and Part Group Objects

The SDK supports modifying the value of this Calculated Compliance field by exposing the following API:

```
public void setCalculatedComplianceForPartSpec (Object specName, Object
calComplianceEntryValue,boolean can_bemodified) throws APIException;
```

The following examples use this API to set the values of the Calculated Compliance field of an Item, Manufacturer Part, and Commodity.

Example 18–26 Modifying Calculated Compliance fields in specification table of Items

```
private static void
   set_calculated_compliance_For_Item(IAgileSession session) {
   IItem item;
      try {
         item = (IItem)session.getObject(ItemConstants.CLASS_PART,"P00007");
         item.setCalculatedComplianceForPartSpec("WEEE", "Non-Compliant",true);
      } catch (APIException e) {
         e.printStackTrace();
      }
}
```

Example 18–27 Modifying the Calculated Compliance field of specification tables of Commodities

```
private static void
      set_calculated_compliance_For_Commotidy(IAgileSession session){
   try {
      ICommodity commodity;
      commodity =
         (ICommodity)session.getObject(ICommodity.OBJECT_TYPE, "Commodity1");
         commodity.setCalculatedComplianceForPartSpec
           ("WEEE", "Non-Compliant",true);
   } catch (APIException e) {
      e.printStackTrace();
   }
}
```

Modifying Settings in the Scheduled Rollups of Declarations

Scheduled rollups, as the name implies, are run according to a schedule and Declarations are objects that enable Agile customers gather compliance information. These rollups use a server-based task that initiates a search for the set of objects that need rollup at the scheduled time. These objects are identified by a flag called *need rollup*. Once found, rollup is performed on these objects and their status is updated. The SDK exposes the `setNeedRollupForDeclaration` API to set (modify) the Need Rollup attribute to True or False.

Example 18–28 Modifying settings in the Need Rollup Flag

```
public void setNeedRollupForDeclaration(boolean complianceEntryValue)
      throws APIException
   IDeclaration subdeclaration =
      (IDeclaration) session.getObject
         (DeclarationConstants.CLASS_SUBSTANCE_DECLARATION,"SD1");
subdeclaration.setNeedRollupForDeclaration (false);
```

Performing External Compliance Rollups on Specification Objects

PG&C's Specification objects have a cover page attribute called the *Rollup Engine.* This attribute enables PG&C users to perform only External Compliance Rollups on Specification objects and avoid Internal Compliance Rollups. The Rollup Engine is linked to an 'Internal/External' switch that enables identifying Specification objects requiring a Rollup. For example, if the attribute is set to *External*, the Rollup Engine ignores performing the Rollup on the Specification and if it is set to *Internal*, the Internal Rollup Engine performs the Rollup. This new attribute can link Specification, Declaration, Item, Manufacturer Part, and Part Group objects. The following IExternalRollupManager.java interface enables performing external rollups with SDK. The related APIs included in this interface are listed below.

- `Int triggerExtractRollupDataforObject(Object partName, Object partClassName, Object changeNumber) throws APIException; - This API invokes an external Rollup against a specific Object (item, mfrpart, part group)`

  Example

  ```
  String objnumber="P00010";
  String objclass="Parts";
  int job_id=externalrollupManager.
     triggerExtractRollupDataforObject (objnumber, objclass, null);
  ```

- Integer [] triggerExternalRollup(int jobID) throws APIException; – This API triggers the external rollup engine to perform the required external rollup

  Example

  ```
  Integer[] extract_id=externalrollupManager.triggerExternalRollup (jobid);
  ```

- This API gets the Composition of the Object that is passed as the parameter in the `Integer [] triggerExternalRollup(int jobID) throws APIException;`.

  Example

  ```
  IVOExternalComposition
  objectcomposition=(IVOExternalComposition)externalrollupManager.getObjectCompos
  ition(extractid);
  ```

- IRow AppendUpdateRow (Object partName, Object partClassName, Object changeNumber, Object siteNumber, IVOExternalComposition

externalComposition) throws APIException; – This API activates the composition row placed in the archived Table by the IRow AppendUpdateRow (Object partName, Object partClassName, Object changeNumber, Object siteNumber, IVOExternalComposition externalComposition) throws APIException; API.

Example

```
String objnumber="P00010";
String objclass="Parts";
Int row_id=row. getRowId ();
ArrayList list=new ArrayList();
list.add(row_id)
externalrollupManager.activeComposition(objnumber, objclass, change_
number, list)
```

- `Boolean deleteExtractData (int jobID) throws APIException;` – This API deletes records in the `Fact_table` that were generated by the `Int triggerExtractRollupDataforObject(Object partName, Object partClassName, Object changeNumber) throws APIException;` API

  Example

  ```
  boolean flag=externalrollupManager.deleteExtractData(jobid);
  ```

Updating Values of Calculated Compliance Attributes Against External Specifications

To update the value of the calculated compliance attribute against an external specification, use the `updatePartRollupResult` API in `IExternalRollupManager` interface.

boolean **updatePartRollupResult**(String objectnumber,String objectclassname,String changenumber, int **spec ID**,int **calculatedCompliance**, Date **rolluptime**) throws APIException;

Table 18–5 Data Field descriptions

Data field	Description
specID	External specification ID
calculatedCompliance	Calculated compliance value
rolluptime	Rollup time

Example 18–29 Updating Values of Calculated Compliance Attribute Against External Specifications

```
boolean flag = externalrollupManager.updatePartRollupResult
   ("P00020","Parts","",6103610,4,new Date());
```

Creating External or Internal Specifications

The `IRollupRegulationFactory` interface uses the following API to create external or internal specifications.

```
public ISpecification createRegulationwithNameRollupEngine
   (String subclasstypename,String regulation_name,
     boolean isexternalengine) throws APIException;
```

Example 18–30 Creating Internal and External Specifications

```
IRollupRegulationFactory rollupregulation=
   (IRollupRegulationFactory)session.getManager
```

```
      (IRollupRegulationFactory.class);

ISpecification sp = regulation.createRegulationwithNumberRollupEngine
   ("Specification", "sina", true);
```

Using the IDeclaration Interface

Agile SDK supports modifying values in the Calculated Compliance Field of Part Group, Item Composition, Manufacturer Part Composition, and Part Group Composition Objects by exposing the following API.

`public void` **`setCalculatedComplianceForDeclarationPartSpecSubstance`** `(Object part_number, Object part_type, Object change_number, Object spec, Object substance, Object` **`complianceEntryValue`**`) throws APIException;`

Modifying value of the Calculated Compliance field of Declarations

These examples use this API to set values the Calculated Compliance field for Item Composition, Manufacturer Part Composition, and Part Group Composition Objects by exposing the following APIs.

Example 18–31 Modifying Values in Item Composition Table

```
private static void ItemSetValue(IAgileSession session2)
   throws APIException {
   IDeclaration dec = IDeclaration)session.getObject
      (DeclarationConstants.CLASS_SUBSTANCE_DECLARATIONS_CLASS, "MD00009");
   dec.setCalculatedComplianceForDeclarationPartSpecSubstance
      ("P00001","Parts", "", "SPEC01", "1,4-BUTANOLIDE", "Non-Compliant");
}
```

Example 18–32 Modifying Values in Manufacturer Part Composition Table

```
private static void MfrpartSetValue(IAgileSession session2)
   throws APIException {
   IDeclaration dec =(IDeclaration)session.getObject
      (DeclarationConstants.CLASS_SUBSTANCE_DECLARATIONS_CLASS, "MD00009");
   dec.setCalculatedComplianceForDeclarationPartSpecSubstance
      ("AMP::12345","Manufacturer parts", null, "", null, "Non-Compliant");
}
```

Example 18–33 Modifying Values in Part Group Composition Table

```
private static void PartFamliySetValue(IAgileSession session)
   throws APIException {
   IDeclaration dec = (IDeclaration)session.getObject
      (DeclarationConstants.CLASS_SUBSTANCE_DECLARATIONS_CLASS, "MD00009");
   dec.setCalculatedComplianceForDeclarationPartSpecSubstance
      ("CAP","Part groups", "", "WEEE", "AL", "Missing Info");
}
```

19

Managing Agile PPM Calendars

This chapter includes the following:

- About Product Portfolio Management (PPM) Calendars
- Creating, Printing and Deleting PPM Calendars
- Updating Workweek Settings
- Adding and Updating Working and Exception (non-working)) Days

About Product Portfolio Management (PPM)

Agile Product Portfolio Management is a web-based application that enables users to manage all aspects of a project or program. PPM is fully integrated with the Agile PLM suite of products to provide a centralized view of project records and associated product information within the organization.

Users of PPM includes executive staff who view portfolio data for the overall status of projects and program/project managers to create projects and manage and monitor project tasks, schedules, and budgets. For more information, refer to the Agile PLM Product Portfolio Management User Guide.

About PPM Calendar Management

The Calendar Management component of PPM is a Release 9.3.4 enhancement. Before this enhancement, you could only specify weekend days as non-working days when preparing PPM schedules, which failed to meet general business requirements. To overcome this shortcoming, this enhancement enables specifying any day of the week an exceptional day. For example, a non-working day or a national holiday unique to a given country, or a given project.

Note: Project tasks listed under "Calendar Tasks Supported by Agile SDK" are performed programmatically and in the batch mode. PPM-related Calendar tasks are performed using the Agile PLM Clients.For background information and procedures, refer to Product Portfolio Management User Guide.

Calendar Tasks Supported by Agile SDK

The PPM Calendar is a new administration data type and is processed by the SDK similar to other existing administration data. For example, Workflow, Privileges, and so on.

The exposed APIs enable performing the following PPM-related tasks programmatically and in the batch mode:

- Creating PPM calendars
- Updating workweek settings
- Adding and Updating a Calendar's Working days and Non-Working days
- Accepting and changing the status of a PPM Calendar
- Getting and printing exception days
- Deleting PPM Calendars

Performing Calendar-related Tasks using Agile SDK

Agile SDK interfaces that are exposed to support these tasks, are listed below. For additional information, you can find the Javadoc generated HTML files that document these interfaces in the Note in "Client-Side Components" on page 1-2.

- ICalendarManage
  - createCalenar, removeCalendar
  - getCalendar, getCalendars
- ICalendar
  - addExceptionalDay
  - updateExceptionalDay
  - deleteExceptionalDay

Creating PPM Calendars

Agile SDK exposes the following interface for this purpose.

```
ICalendar calendar = manager.createCalendar(param);

   ICalendarManager manager =
     (ICalendarManager)session.getManager(ICalendarManager.class);
   Map<Integer, Object> param = new HashMap<Integer, Object>();
   param.put(WorkCalendarConstants.ID_CALENDAR_NAME,"SDK1");
   param.put(WorkCalendarConstants.ID_CALENDAR_ENABLED, "Yes");

   int[] weekday = new int[]{WorkCalendarConstants.ID_WEEKEND_SAT,
            WorkCalendarConstants.ID_WEEKEND_SUN};
   param.put(WorkCalendarConstants.ID_CALENDAR_WEEK_SETTING, weekday);

   // create calendar
   ICalendar calendar = manager.createCalendar(param);
   System.out.println(calendar);
```

The following code sample is an example of creating a PPM calendar:

Updating Workweek Settings

Agile SDK exposes the following interface for this purpose.

```
calendar.setValue
       (WorkCalendarConstants.ID_CALENDAR_WEEK_SETTING, weekday);
```

The following code sample shows how to get and set an existing a PPM calendar's workweek settings:

```
ICalendarManager manager =
                (ICalendarManager) session.getManager(ICalendarManager.class);

            // get the calendar
            ICalendar calendar = manager.getCalendar("SDK1");
            System.out.println(calendar);

            // Update work week setting
            int[] weekday = new int[]{WorkCalendarConstants.ID_WEEKDAY_FRI,
                            WorkCalendarConstants.ID_WEEKEND_SUN};
            calendar.setValue
                    (WorkCalendarConstants.ID_CALENDAR_WEEK_SETTING, weekday);
```

Managing Working and Non-Working Days

Agile SDK exposes the following interface to add and update PPM Calendar working and non-working (exceptional) days.

```
          ICalendar calendar = manager.getCalendar("SDK1");
```

The following code sample shows how to add and update working and non-working days in PPM Calendars:

```
ICalendarManager manager =
             (ICalendarManager)session.getManager(ICalendarManager.class);
             // get the calendar
             ICalendar calendar = manager.getCalendar("SDK1");

             Map expday = new HashMap();
             expday.put(WorkCalendarConstants.EXCEPTION_SETTING_START,
                          "04/07/2014");
             expday.put(WorkCalendarConstants.EXCEPTION_SETTING_END,
                          "04/08/2014");
             expday.put(WorkCalendarConstants.
                           EXCEPTION_SETTING_COMMENT, "SDK_update11");

             // add exception day
             calendar.addExceptionalDay(expday);
```

Getting and Printing Exceptional Days

Agile SDK exposes the following interface to get and print a PPM Calendar's working and non-working days:

```
             ICalendar calendar = manager.getCalendar("SDK1");
                      System.out.println(calendar);
```

The following code samples get and print a PPM Calendar's exception days as well as exception days for a specific calendar year:

```
             ICalendarManager manager =
                    (ICalendarManager)session.getManager(ICalendarManager.class);
             // get the calendar
             ICalendar calendar = manager.getCalendar("SDK1");
             System.out.println(calendar);
```

```
// Get & Print exception day of one specific year
IVOCalendarDay[] voCalDays = calendar.getExceptionalDaysOfYear(2014);
for (IVOCalendarDay day : voCalDays) {
      System.out.println(day+"            "+day.getComment());
}
```

Deleting PPM Calendars

Agile SDK exposes the following interface to delete PPM Calendars.

```
manager.removeCalendar("SDK1");
```

Agile SDK exposes the following interface to delete existing PPM Calendar's

```
ICalendarManager manager =
             (ICalendarManager)session.getManager(ICalendarManager.class);

             // remove calendar
             manager.removeCalendar("SDK1");
```

Accepting Changes and Changing the Status of Calendar

Agile SDK exposes the following interfaces for this purpose.

- Accepting Changes made to PPM Calendars:

  ```
  "IProgram.acceptCalendarChange()
  ```
- Changing the Status of a PPM Calendar to Cancel or Not Started:

  ```
  "IProgram.changeStatusToCancelOrNotStarted(IStatus newStatus, String comment,
  Object[] notifyList, boolean urgent, boolean isContinue)
  ```

Examples

The following code samples show accepting changes to a PPM Calendar and changing the Status of a PPM Calendar to Cancel, or Not Started.

Accepting Changes

```
IProgram program = (IProgram) session.getObject(IProgram.OBJECT_TYPE, "PGM00011");
program.acceptCalendarChange();
```

Changing the Status of PPM Calendars

```
proposedTask = (IProgram)agileSession.getObject(IProgram.OBJECT_TYPE, "T00003");
IStatus myCurrentStatus = proposedTask.getStatus();
if(myCurrentStatus.getStatusType().equals(StatusConstants.TYPE_CANCEL)){
       cancelTask(proposedTask,getNotStartedStatusForWorkflow(proposedTask));
       System.out.println("task changed to not started");

} else {
       cancelTask(proposedTask,getCancelStatusForWorkflow(proposedTask));
       System.out.println("task changed to canceled");

}
private static IStatus getCancelStatusForWorkflow(IProgram obj){
       IWorkflow workflow = null;
       IStatus status = null;
       IStatus[] statuses = null;
```

```
        try{
                workflow = obj.getWorkflow();
                status = null;
                if(workflow!= null){
                        statuses = workflow.getStates();
                        int noOfStatuses = statuses.length;
                        for (int i = 0; i < noOfStatuses; i++) {
                                status = statuses[i];
                                if(status.getStatusType()
                                        .equals(StatusConstants.TYPE_CANCEL))break;
                        }
                }
        }catch(APIException e){
                e.printStackTrace();
        }catch(Exception e){
                e.printStackTrace();
        }
        return status;
}

private static IStatus getNotStartedStatusForWorkflow(IProgram obj)
                throws APIException{
        IWorkflow workflow = null;
        IStatus status = null;
        IStatus[] statuses = null;

        try{
                workflow = obj.getWorkflow();
                status = null;
                if(workflow!= null){
                        statuses = workflow.getStates();
                        int noOfStatuses = statuses.length;
                        for (int i = 0; i < noOfStatuses; i++) {
                                status = statuses[i];
                                if(status.getStatusType().equals
                                        (StatusConstants.TYPE_PENDING))break;
                        }
                }
        }catch(APIException e){
                e.printStackTrace();
        }catch(Exception e){
                e.printStackTrace();
        }
        return status;
}

private static void cancelTask(IProgram objTask,IStatus status)
                        throws APIException{
        try{
                objTask.changeStatusToCancelOrNotStarted(status,
true, "Cancelled by Task Creation PX", true, true, new Object[0], true, true);
                System.out.println("task status changed");
        }catch(APIException e){
                throw e ;
        }catch(Exception e){
                e.printStackTrace();
        }
}
```

20 Creating and Managing Projects

This chapter includes the following:

- About Projects and Project Objects
- Differences in the Behavior of Project Objects
- Creating Projects
- Adding Rules for PPM Objects
- Loading Projects
- Adding “FileFolder” to Project's Content Tab
- Using Project Templates
- Scheduling Projects
- Setting Start and End Timestamps for PPM Date Attributes
- Working with Project Baselines
- Delegating Ownership of a Project to Another User
- Adding Resources to a Project Team
- Substituting Project Resources Locking or Unlocking Projects
- Locking or Unlocking Projects
- Working with Discussions

Note: In Release 9.3, the name of the Program Base Class was changed to Projects and Projects to Sourcing Projects. However the interface for the Projects Base Class is still called IProgram in the SDK Guide and in Javadoc references.

About Projects and Project Objects

You can use the project management features of Agile Product Portfolio Management (PPM) to define a project and its associated elements such as activity schedules, deliverables, and discussions. These capabilities enable you to determine the availability of the required resources, assigning resources to tasks, identifying bottlenecks, and responding to over- and under-allocated resource conditions. You can also create and reuse project templates.

The Project object is used to schedule and execute projects. Each project, in addition to schedule information, contains attachments, discussions and actions items, resources and roles, and history and content of related activities. For management visibility, data is rolled up to higher levels by rules and parent-child relationships.

The Agile API provides support for creating, loading, and working with Projects. The IProgram interface represents all Project objects, including programs, phases, tasks, and gates.

Similar to other Agile PLM business objects, the IProgram interface implements IRoutable, which means it uses the same IRoutable.changeStatus() method to change a Projects' Workflow status and to route it to other users. For more information, "Changing the Workflow Status of an Object" on page 16-9.

Differences in the Behavior of Project Objects

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

- The Project object is a container of other underlying Project objects, such as Phases, Tasks, and Gates. The underlying Project objects are related to the parent object, usually the Projects, through the Schedule table.
- Projects have baselines that allow you to track changes in the schedule. Therefore, the IProgram interface provides methods that let you create, get, or remove a baseline.
- Projects can be archived. If you archive the root Projects, the entire Projects tree is soft-deleted from the system.
- Projects can be locked or unlocked.

Creating Projects

Use the IAgileSession.createObject() method to create Projects. When you specify the Project's parameters, you must specify the Project subclass (for example, Program, phase, task, or gate). For Programs, phases, and tasks, you must also specify following required Project attributes:

- General Info.Name
- General Info.Schedule Start Date
- General Info.Schedule End Date
- General Info.Duration Type

For gates, only two attributes are required, General Info.Name and General Info.Schedule End Date.

The following example shows how to create new Projects and specify the required attributes.

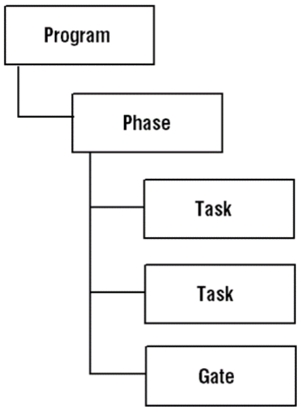
Example 20–1 Creating Projects

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

// Set Projects name
   String name = "APOLLO PROJECTS";

// Set Projects start date
```

```
   Date start = new Date();
   start.setTime(1);

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

// Set Projects duration type
   IAttribute attr = m_admin.getAgileClass(ProgramConstants.CLASS_PROGRAM).
   getAttribute(ProgramConstants.ATT_GENERAL_INFO_DURATION_TYPE);
   IAgileList avail = attr.getAvailableValues();
   avail.setSelection(new Object[] {"Fixed"});

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

// Create Projects
   IProgram program =
      (IProgram)m_session.createObject(ProgramConstants.CLASSPROGRAM, params);
} catch (APIException ex) {
      System.out.println(ex);
}
```

Projects contain other types of activities, such as phases, tasks, and gates. A gate is a special milestone-a task with an end date but no duration-that denotes the completion of a set of related phases, tasks, or Projects. The following figure shows the hierarchy of Project objects.

Figure 20–1 Program hierarchy



You can use the IAgileSession.createObject() method to create phases, tasks, and gates in the same way that you create other Project objects. Once you create these different types of activities, you can add them to the Schedule table of a Projects object. For more information, see "Scheduling Projects" on page 20-8.

Adding Rules for PPM Objects

In PLM, any object that is assigned a lifecycle phase or a workflow can be added as a deliverable. The only exceptions are Discussions, Users, and User groups.

Rules in PPM ensure an activity will not complete before the completion of the preceding activity as set in the workflow, or lifecycle phase. For example, if you want to ensure the completion of an activity before a Gate is opened, you can add that activity as a deliverable for the Gate to open. You can even restrict one Gate from opening before another by adding the prior Gate as a deliverable for the subsequent Gate to open. For more information, refer to the Agile PLM *Product Portfolio Management User Guide*.

The SDK supports this function with IProgram interface as shown in the following example.

Example 20–2 Setting rules for PPM objects

```
try{
//Get Program
   IProgram pgm =
      (IProgram)session.getObject
   (ProgramConstants.CLASS_PROGRAM,"PGM00239");

//Get Object and add as relationship under Content tab
   IChange eco = (IChange)session.getObject(ChangeConstants.CLASS_ECO,"C00060");
   ITable table = pgm.getTable(ProgramConstants.TABLE_RELATIONSHIPS);
   IRow row = table.createRow(eco);

//Get the Control object status
   IStateful state = (IStateful)pgm;
   IStatus[] statuses = state.getStates();
   IStatus ctl_status = null;
   for(int i=0; i<statuses.length; i++){
   if(statuses[i].getName().equals("In Process")){
      ctl_status = statuses[i];
   break;
   }
}

//Get the Affected object status
   state = (IStateful)eco;
   statuses = state.getStates();
   IStatus aff_status = null;
   for(int i=0; i<statuses.length; i++){
   if(statuses[i].getName().equals("Submitted")){
      aff_status = statuses[i];
      break;
   }
}

//Add Rule
   HashMap map = new HashMap();
   map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_CONTROLOBJECT, pgm);
   map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_AFFECTEDOBJECT, eco);
   map.put(CommonConstants.ATT_RELATIONSHIPS_RULE_CONTROLOBJECTSTATUS,
      ctl_status);
   map.put(CommonConstants.
         ATT_RELATIONSHIPS_RULE_AFFECTEDOBJECTSTATUS, aff_status);
   row.setValue(CommonConstants.ATT_RELATIONSHIPS_RULE, map);
   System.out.println(row.getCell
   (CommonConstants.ATT_RELATIONSHIPS_RULE));
   }catch (APIException ex) {
      System.out.println(ex);
   }
```

Loading Projects

To load Projects, use the IAgileSession.getObject() method. To uniquely identify a Project object, specify the value for the General Info.Number attribute. You can also load a Project object by searching for it by name, and then selecting it from the search results.

Note: The IProgram.getName() method actually returns the value of the General Info.Number attribute, and not that of General Info.Name.

Example 20–3 Loading Projects

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

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

Adding “FileFolder” to Project's Content Tab

You can add FileFolders to the Content tab of a Project using the IProgram API. The following example shows how to perform this task.

Example 20–4 Adding a FileFolder to the Content tab of a Project

```
private void uploadFile(IAgileSession session,String foldname)
   throws APIException {
     IProgram program =
        (IProgram) session.getObject(IProgram.OBJECT_TYPE, "PGM00041");
     IFileFolder ff =
        (IFileFolder) session.getObject(IFileFolder.OBJECT_TYPE, foldname);

//Upload filefolder to Conents table
   ITable table = program.getTable(ProgramConstants.TABLE_RELATIONSHIPS);
   table.createRow(ff);
}
```

Using Project Templates

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

This special version of the saveAs() method enables to use the SDK to:

- Create a new Project from a template and specify the tables that you want copied over
- Change the owner of the Project and the owner of the children
- Create a new Project template by saving a Project as a template

Creating New Projects Using Templates

You can use this special version of the saveAs()method to specify the Project tables that you want to copy from the original Project to the new Project. You don't need to specify all tables. The General Info, Schedule, Dependencies - Dependent Upon, Dependencies - Required For, and Workflow tables are copied automatically. The Discussion, News, and History tables cannot be copied. Generally, you should copy Page Two, Page Three (if it's used), and the Team table, as shown in the example below.

Example 20–5 Creating a new Project from a template

```
try
{
// Get the Project 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();

// Available values for Template attribute are Active, Proposed, and Template
   templateList.setSelection(new Object[] {"Active"});
   map.put(ProgramConstants.ATT_GENERAL_INFO_NAME, name);
   map.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE, new Date());
   map.put(ProgramConstants.ATT_GENERAL_INFO_TEMPLATE, templateList);

// Define the tables to copy to the new program from the template
   Integer pagetwo = ProgramConstants.TABLE_PAGETWO;
   Integer pagethree = ProgramConstants.TABLE_PAGETHREE;
Integer team = ProgramConstants.TABLE_TEAM;
   Object[] tables = new Object[]{pagetwo, pagethree, team};

// Save the template as a new program
   IProgram program =
      (IProgram)template.saveAs(ProgramConstants.CLASS_PROGRAM,tables, map);
      }
   } catch (APIException ex) {
   System.out.println(ex);
}
```

Creating Projects and Changing Ownerships

When you create a Project from a template using the saveAs() API call, you can change the ownership of the Project and propagate the change to the children of the Project. In SDK, the exposed API is used to accomplish this:

```
public IAgileObject saveAs(Object type, Object[] tablesToCopy,Object params,
boolean applyToChildren)
```

This is done by specifying a value for both the ProgramConstants and the OWNER attribute. The value for the OWNER attribute is required in order to change the Project's ownership. Set the Boolean applyToChildren to true if you want to apply the OWNER value to all children.

In the UI, when you create a Project from a template, you have the option to change ownership of the Project and apply the change to the children. In this situation, the SDK mirrors the UI. However, the original Project must be a *Template* to create a Project from a template via SDK's saveAs() API.

Note: In the SDK, a Project is a template when the value of the General Info.Template attribute in the original program is set to Template.

Example 20–6 Creating Projects from a template, change owner, and propagate change

```
public IProgram saveTemplateAndSetOwner (IProgram template, String userID, boolean
      applyToChildren) throws APIException {

/* "template" is a program template
*   userID -- The "userID" of the user that
*   is specified as the owner of the Saved program object
*   applyToChildren -- true or false.
*   If "true" the "specified owner" is the owner of the entire program tree
*   If "false", the specified owner is the owner of the Root Parent object only
*/

// Generate a random name for the Saved Program object
HashMap map = new HashMap () ;
String newPgmName =
   "PROG" + System.currentTimeMillis() ;
   IUser user =
      session.getObject(UserConstants.CLASS_USER, userID) ;
      map.put(ProgramConstants.ATT_GENERAL_INFO_NAME, newPgmName);
      map.put(ProgramConstants.ATT_GENERAL_INFO_OWNER, user);
      map.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_START_DATE, new Date());

// Define the tables to copy from the template and if
// you do not want to copy any tables, specify "null"
// for the "tables" param.

   Integer pageTwo = ProgramConstants.TABLE_PAGETWO ;
   Integer pageThree = ProgramConstants.TABLE_PAGETHREE ;
   Integer team = ProgramConstants.TABLE_TEAM ;
   Object[] tables =
         { pageTwo, pageThree, team } ;
   IProgram pgm =
         (IProgram) root.saveAs
            (ProgramConstants.CLASS_PROGRAM, tables, map, applyToChildren);
   System.out.println
      ("New Program Number = " + pgm.getName()) ;
   System.out.println
      ("Owner Value = " + pgm.getValue
         (ProgramConstants.ATT_GENERAL_INFO_OWNER).toString())
   return pgm ;
}
```

Saving Projects as Templates

When you create a Project, 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 Project or when you save it as a new Project. Existing Projects cannot be changed from the “Active” or “Proposed” state to “Template”.

The following example shows how to open a Project object and save it as a template.

Example 20–7 Saving a Projects object as a template

```
try {
// Get the program whose number is PGM00005
   IProgram program =
      (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00005");

   if (program != null) {
   // Create a hash map of program attributes for the new program
      HashMap map = new HashMap();
      String name = "Rapid Development");
   IAttribute att =
      m_admin.getAgileClass(ProgramConstants.CLASPROGRAM).getAttribute
         (ProgramConstants.ATT_GENERAL_INFO_TEMPLATE);
   IAgileList templateList =
      att.getAvailableValues();

// Available values for 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 tables to copy to the template
   Integer pagetwo = ProgramConstants.TABLE_PAGETWO;
   Integer pagethree = ProgramConstants.TABLE_PAGETHREE;
   Integer team = ProgramConstants.TABLE_TEAM;
   Object[] tables = new Object[]{pagetwo, pagethree, team};

// Save the program as a template
   IProgram program = (IProgram)template.saveAs
      (ProgramConstants.CLASS_PROGRAM, tables, map);
   }
} catch (APIException ex) {
   System.out.println(ex);
}
```

Scheduling Projects

To schedule Projects, edit the Schedule table, which lets you add, edit, and remove schedule items. To add a new row to the Schedule table, use the ITable.createRow() method and specify an IProgram object for the parameter.

Example 20–8 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 Project
   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("06/01/15"));
   info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
   df.parse("06/10/15"));
   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("06/01/15"));
   info.put(ProgramConstants.ATT_GENERAL_INFO_SCHEDULE_END_DATE,
   df.parse("06/05/15"));
   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("06/10/15"));
   IProgram gate =
      (IProgram)m_session.createObject(ProgramConstants.CLASS_GATE, info);
   row = schedule.createRow(gate);
   }
} catch (APIException ex) {
      System.out.println(ex);
}
```

Once a Project's schedule is defined, you can reschedule the project with the IProgram.reschedule() method. The reschedule() method takes a couple of parameters, the IProgram.RESCHEDULE constant and the new value for that schedule option. Here are the list of IProgram.RESCHEDULE constants you can use:

- STARTDATE - This moves the scheduled start date to the specified date.
- ENDDATE - This moves the scheduled end date to the specified date.
- BACKWARDDAYS - This moves the schedule backward by the specified number of days.
- FORWARDDAYS - This moves the schedule forward by the specified number of days.

Example 20–9 Rescheduling Projects

```
try {
// Get a Project
   IProgram program =
      (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
   if (program != null) {

// Define new start and end dates
   String startDate = "02/01/2015 GMT";
   String endDate = "06/01/2015 GMT";
   SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy z");
   Date start = df.parse(startDate);
   Date end = df.parse(endDate);

// Change the schedule start date
   program.reschedule(IProgram.RESCHEDULE.STARTDATE, start);

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

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

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

Setting Start and End Timestamps for PPM Date Attributes

Start and End timestamps are automatically set for PPM Date attributes when the end user creates or edits a scheduled PPM task. You can schedule PPM tasks in *Working Time*, which is configurable in the agile.properties file. When creating or editing PPM objects, the end user must specify a valid date for Working Time within the Start and End values set in agile.properties file for the following PPM Date attributes:

- Schedule
- Estimated

- Actual

> **Note:** If the specified time for the above Date attributes is not within the Start and End values set in agile.properties, PPM will not complete the end operation of the user. For example, if the Start and End values in agile.properties are 8:00 AM and 6:00 PM and those specified by the user are different, PPM will not complete the applicable operation.

The environment variable (flag) called ppm.date.appendtime automatically sets the appropriate time for POM's Schedule, Estimated, or Actual dates before sending these values to the Agile server to update the Date attribute. When the flag is turned on, the existing *time* in the date value for Schedule, Estimated, or Actual dates is ignored and is automatically set according to the following rules:

- If the attribute is *Schedule Start Date*, *Estimated Start Date* or *Actual Start Date*, then the start working time of the day set in agile.properties is appended. For example, if the working time inagile.properties is configured as 8:00:00-12:00:00, 13:00:00-17:00:00, then the start working time for the day is 8 AM. The time portion in the date value for start date attributes is set to 8 AM.
- If the attribute is *Schedule End Date*, *Estimated End Date* or *Actual End Date*, then the end working time of the day set in agile.properties is appended. For example, if the working time in agile.propertieses is configured as 8:00:00-12:00:00, 13:00:00-17:00:00, then the end working time for the day is 5 PM. The time portion in the date value for end date attributes is set to 5 PM.

By default, the value of the ppm.date.appendtime flag is set to True. This is to ensure backward compatibility of PPM SDK so that SDK Clients compiled in earlier releases can execute without re compilation.

To set timestamp for PPM date attributes in SDK Client

You have the following options:

- Set the ppm.date.appendtime flag to False using syntax such as java **-Dppm.date.appendtime=false** <SDK Program Name>

OR,

- Set an environment variable called ppm.date.appendtime and execute the SDK program.

> **Note:** This is a global setting and the setting will apply to all SDK programs that are running on the given platform.

Working with Project Baselines

Project baselines allow you to compare actual progress with your original plans. When you create a baseline, a snapshot of your Project'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 Project 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 20–10 Creating and retrieving baselines

```
try {
// Get a Project
   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 Project to Another User

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

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

Example 20–11 Delegating ownership of a Project 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 Project Team

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

When you add a resource to the Team table, you specify what roles the user or user group has for that Project object. The roles available are not the complete set of Agile PLM roles; they are roles specifically related to Project functionality. Here is the list of roles you can assign to team members:

- Executive
- Change Analyst
- Program Team Member
- Program Manager
- Resource Pool Owner
- Program Administrator

For a description of each of these roles, refer to the *Agile PLM Administrator Guide*.

The Team table has two attributes that require special mention:

- ProgramConstants.ATT_TEAM_NAME
- ProgramConstants.ATT_TEAM_ROLES.

These are SingleList and MultiList attributes, respectively. To get the available values for these attributes, use ITable.getAvailableValues() instead of IAttribute.getAvailableValues(). Otherwise, the IAgileList object returned by the method can contain invalid list values.

Example 20–12 Adding resources to a Project team

```
try {
// Get users
   IUser user1 = (IUser)session.getObject(UserConstants.CLASS_USER, "daveo");
   IUser user2 = (IUser)session.getObject(UserConstants.CLASS_USER, "yvonnec");
   IUser user3 = (IUser)session.getObject(UserConstants.CLASS_USER, "albertl");
   IUser user4 = (IUser)session.getObject(UserConstants.CLASS_USER, "brians");

// Get a resource pool (user group)
   IUserGroup pool =
      (IUserGroup)session.getObject(IUserGroup.OBJECT_TYPE, "Development");

// Add all four users to the resource pool
   ITable usersTable = pool.getTable(UserGroupConstants.TABLE_USERS);
      usersTable.createRow(user1);
      usersTable.createRow(user2);
      usersTable.createRow(user3);
      usersTable.createRow(user4);

// Get a Project
   IProgram program =
      (IProgram)session.getObject(IProgram.OBJECT_TYPE, "PGM02423");
   if (program != null) {
   // Get the Team table of the program
   ITable teamTable = program.getTable(ProgramConstants.TABLE_TEAM);
   // Get Roles attribute values (use ITable.getAvailableValues)
   IAgileList attrRolesValues =
      teamTable.getAvailableValues(ProgramConstants.ATT_TEAM_ROLES);

// Create a hash map to hold values for row attributes
   Map map = new HashMap();

// Add the first user to the team
   attrRolesValues.setSelection(new Object[]
      {"Change Analyst","Projects Manager"});
   map.put(ProgramConstants.ATT_TEAM_NAME, user1);
   map.put(ProgramConstants.ATT_TEAM_ROLES, attrRolesValues);
   IRow row1 = teamTable.createRow(map);

// Add the second user to the team
   attrRolesValues.setSelection(new Object[]{"Projects Administrator"});
   map.put(ProgramConstants.ATT_TEAM_NAME, user2);
   IRow row2 = teamTable.createRow(map);

// Add the resource pool to the team
   attrRolesValues.setSelection(new Object[]{"Projects Team Member"});
   map.put(ProgramConstants.ATT_TEAM_NAME, pool);
   IRow row3 = teamTable.createRow(map);
}
```

In Agile Web Client, when you add a resource pool to the Team table, you can replace the pool with one or more resources contained within it. In other words, instead of assigning the entire resource pool, you can assign select users from the pool. The IProgram.assignUsersFromPool() method reproduces this functionality. To use assignUsersFromPool(), you must specify a user group that has already been added to the Project's Team table.

Example 20–13 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 Project 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 Project. To substitute Project 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 Project's children.

Example 20–14 Substituting Project resources

```
try {
// Get a Project
   IProgram program =
      (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
   if (program != null) {
   // Get users
      IUser u1 =
         (IUser)m_session.getObject(UserConstants.CLASS_USER, "akurosawa");
      IUser u2 =
         (IUser)m_session.getObject(UserConstants.CLASS_USER, "creed");
      IUser u3 =
         (IUser)m_session.getObject(UserConstants.CLASS_USER, "dlean");
      IUser u4 =
         (IUser)m_session.getObject(UserConstants.CLASS_USER, "jford");

// Get a user group
   IUserGroup ug =
      (IUserGroup)m_session.getObject(IUserGroup.OBJECT_TYPE, "Directors");

// Substitute u1 with u3 and do not apply to children
   program.substituteResource(u1, u3, false);

// Substitute u2 with u4 and apply to children
   program.substituteResource(u2, u4, true);

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

Locking or Unlocking Projects

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

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

Example 20–15 Locking Projects

```
try {
// Get a Program
   IProgram program =
      (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
   if (program != null) {
   // Lock it
      program.setLock(true);
      }
} catch (APIException ex) {
   System.out.println(ex);
}
```

Working with Discussions

During the course of a project, issues arise that require users to collaborate and exchange information. Agile PLM provides threaded discussion functionality that allows team members to reply with their feedback, providing a record of their thoughts and ideas. Discussions are asynchronous; that is, they do not require a simultaneous connection from all discussion participants. Users can reply to any thread of the discussion independently. To close issues, action items can be assigned to team resources. The Discussion object is used to manage both threaded discussions and the action items related to them.

Discussion objects, unlike Projects, are not routable objects. Therefore, discussions do not have workflows associated with them.

Note: The Action Items, Cover Page, and Replies tables appear on the Discussion tab in Agile PLM clients. The Page Two table appears on the Details tab in Agile PLM clients. The Where Used table is not supported, its functionality is replaced by General Info.Related To field.

Creating a Discussion

To create a discussion, use the IAgileSession.createObject() method. When you specify discussion parameters, you must specify the discussion subclass and the following required discussion attributes:

- Cover Page.Number
- Cover Page.Subject

In addition, you must also specify data for the Cover Page.Notify List and Cover Page.Message attributes. Otherwise, the discussion does not have a Notification list, or a message that users can respond to.

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

Example 20–16 Creating a discussion

```
try {
// Create a hash map variable
   Map map = new HashMap();

// Set the Number field
   IAgileClass discussionClass =
      m_session.getAdminInstance().getAgileClass
         ( DiscussionConstants.CLASS_DISCUSSION);
   String number =
      discussionClass.getAutoNumberSources()[0].getNextNumber();

// Set the Subject field
   String subject = "Packaging issues";

// Make the Message field visible
   IAttribute attr =
      discussionClass.getAttribute
         (DiscussionConstants.ATT_COVER_PAGE_MESSAGE);
   IProperty propVisible =
      attr.getProperty(PropertyConstants.PROP_VISIBLE);
   IAgileList list =
      propVisible.getAvailableValues();
   list.setSelection(new Object[] { "Yes" });

// Set the Message field
   String message =
      "We still have problems with the sleeves and inserts." +
         "Let's resolve these things at the team meeting on Friday.";

// Set the Notify List field
   IUser user1 = m_session.getCurrentUser();
   IUser user2 =
      (IUser)m_session.getObject(UserConstants.CLASS_USER, "jdassin");
   attr = discussionClass.getAttribute
      (DiscussionConstants.ATT_COVER_PAGE_NOTIFY_LIST);
   list =
      attr.getAvailableValues(); list.setSelection(new Object[] {user1, user2});

// Put the values into the hash map
map.put(DiscussionConstants.ATT_COVER_PAGE_NUMBER, number);
map.put(DiscussionConstants.ATT_COVER_PAGE_SUBJECT, subject);
map.put(DiscussionConstants.ATT_COVER_PAGE_MESSAGE, message);
map.put(DiscussionConstants.ATT_COVER_PAGE_NOTIFY_LIST, list);

// Create a Discussion object
IDiscussion discussion = (IDiscussion)m_session.createObject
(DiscussionConstants.CLASS_DISCUSSION, map);

// Get a Projects
   IProgram program =
      (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
   if (program != null) {

// Get the Discussion table
   ITable discTable = program.getTable(ProgramConstants.TABLE_DISCUSSION);
// Add the new discussion to the table
   discTable.createRow(discussion);
   }
} catch (APIException ex) {
    System.out.println(ex);
}
```

Replying to a Discussion

Team members or notified users-that is, users listed in the Cover Page.Notified List field of a discussion-can reply to discussions. When you reply to a discussion, you create another nested table in the Replies table.

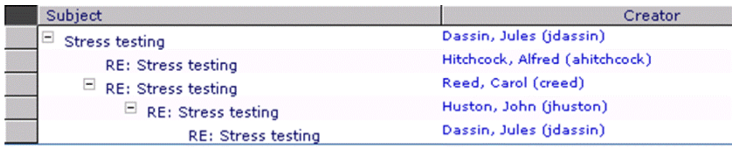
Example 20–17 Replying to a discussion

```
private void replyToDiscussion() throws Exception {
   Iterator it;
      IDiscussion discussion;

// Get a Project
   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 Agile Web Client.

Figure 20–2   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 20–18   How to expand the Replies table

```
// Read the Replies table
   public void readRepliesTable(IDiscussion discussion) throws Exception {
   ITable replies =
      discussion.getTable(DiscussionConstants.TABLE_REPLIES);
   browseReplies(0, replies);
   }

// Recursively browse through all levels of the Replies table
   void browseReplies(int indent, ITable replies) throws Exception {
   Iterator i = replies.iterator();
   while (i.hasNext()) {
      IRow row = (IRow) i.next();
      System.out.print(indent(indent*4));
      readRow(row);
      System.out.println();
   ITable followup =
      (ITable)row.getValue(DiscussionConstants.ATT_REPLIES_CHILD_TABLE);
   browseReplies(indent + 1, followup);
      }
   }
// Read each cell in the row and print the attribute name and value
```

```
   static protected void readRow(IRow row) throws Exception {
      ICell[] cells = row.getCells();
      for (int j = 0; j < cells.length; ++j) {
      Object value = cells[j].getValue();
      System.out.print( "\t" + cells[j].getAttribute().getName() + "="+ value);
      }
   }
// Indent text
   private String indent(int level) {
   if (level <= 0) {
   return "";
   }
   char c[] = new char[level*2];
   Arrays.fill(c, ' ');
   return new String(c);
   }
```

Joining a Discussion

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

Note: If you are not on the Notify List of a Discussion object, you cannot read the replies. However, anyone listed on the Team table of a Projects can join a discussion associated with that Project.

Example 20–19 Joining a discussion

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

Example 20–20 Creating an action item

```
private void replyToDiscussion() throws Exception
{

// Get a Project
   IProgram program =
      (IProgram)m_session.getObject(IProgram.OBJECT_TYPE, "PGM00012");
   if (program != null) {

   // Create a hash map for Action Item parameters
      HashMap map = new HashMap();

   // Set the Subject field
      String subj = "Update packaging requirements";
      map.put(ProgramConstants.ATT_ACTION_ITEMS_SUBJECT, subj);

// Set the Assigned To field
   IUser user1 =
      (IUser)m_session.getObject(UserConstants.CLASS_USER, "akurosawa");
   IAttribute attr =
      m_session.getAdminInstance().getAgileClass(
   ProgramConstants.CLASS_PROGRAM).getAttribute(
      ProgramConstants.ATT_ACTION_ITEMS_ASSIGNED_TO);
      IAgileList list = attr.getAvailableValues();
      list.setSelection(new Object[] {user1});
      map.put(ProgramConstants.ATT_ACTION_ITEMS_ASSIGNED_TO, list);

// Set the Due Date field
   DateFormat df = new SimpleDateFormat("MM/dd/yy");
   map.put(ProgramConstants.ATT_ACTION_ITEMS_DUE_DATE, df.parse("03/30/05"));

// Get the Action Items table
   Table table = program.getTable(ProgramConstants.TABLE_ACTIONITEMS);

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

21 Handling Exceptions

This chapter includes the following:

- About Exceptions
- Getting Error Codes
- Disabling and Enabling Error Codes with Bulk APIs
- Getting Error Messages
- Saving and Restoring State Enabled and Disabled Warnings
- Warnings on Deleting Objects Disabled Automatically by Agile API

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 on screen. To handle an exception gracefully, your program must:

- Protect code that contains a method that might throw an exception by putting it in a try block.
- Test for and handle any exceptions that are thrown inside a catch block.

The Agile API provides a subclass of Exception called APIException. This is a general-purpose exception class that is used throughout the Agile API to handle Agile PLM runtime errors. In the Agile API HTML reference, each method indicates the types of exceptions it throws. Generally, any Agile API method that requires interaction with the Agile Application Server throws APIException. The table below lists the APIException class methods for handling exceptions:

Table 21–1 API Exception Error Codes

Method	Description
getErrorCode()	Returns the number of the error code associated with the APIException.
getMessage()	Returns the error message associated with the APIException.
getRootCause()	Returns the root cause of the APIException, if any.
getType()	Returns the type of exception.

Exception Constants

The ExceptionConstants class contains String constants for all Agile Application Server and Agile API runtime error and warning codes. For a description of each of these constants, refer to the API Reference files at `http://edelivery.oracle.com/`.

Several ExceptionConstants are for exceptions that are used to display an Agile PLM warning message before completing an action. All constants for warning messages end with the suffix WARNING. If you don't want to use Agile PLM warning messages in your code, you can disable them. For more information, see "Disabling and Enabling Error Codes with Bulk APIs" on page 21-2.

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 21–1 Getting Agile PLM error codes

```
private void removeApprover
   (IChange change, IUser[] approvers, IUser[] observers, String comment) {

// Remove the selected approver
   try {
   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);
      }
   }
```

Disabling and Enabling Error Codes with Bulk APIs

The SDK supports the following bulk operations in IAgileSession to disable/enable all error codes or for a given set of error codes:

- IAgileSession.enableWarnings(Integer[])
- IAgileSession.disableWarnings(Integer[])
- IAgileSession.enableAllWarnings()
- IAgileSession.disableAllWarnings()

The process is similar to the previous example. The following example shows how to use these bulk APIs to suppress warnings while releasing a Change.

Example 21–2 Disabling and enabling error codes in the bulk mode

```
public static void releseECO(IAgileSession session, IChange change)
   throws Exception {

// Set workflow
   IWorkflow workflow = change.getWorkflows()[0];
   change.setWorkflow(workflow);
   IStatus submit = getStatus(workflow, StatusConstants.TYPE_SUBMIT);
   IStatus ccb = getStatus(workflow, StatusConstants.TYPE_REVIEW);
   IStatus released = getStatus(workflow, StatusConstants.TYPE_RELEASED);
   session.disableWarnings(new Integer[] {
      ExceptionConstants.APDM_WFL_ATLEASTONECHGANALYST_WARNING,
      ExceptionConstants.APDM_MISSINGFIELDS_WARNING });
   // instead you can use session.disableAllWarnings()

// route to SUBMIT
   change.changeStatus(submit, false, null,
      false, false, new Object[]{}, new Object[]{}, new Object[] {}, false);

// Change status to CCB
   change.changeStatus(ccb, false, null, false, false, new Object[]{}, new
   Object[]{}, new Object[]{}, false);

// route from CCB to release
   change.changeStatus(released, false, "release", false, false, new Object[]{},
      new Object[]{}, new Object[]{}, false);
   session.enableWarnings(new Integer[] {
      ExceptionConstants.APDM_WFL_ATLEASTONECHGANALYST_WARNING,
      ExceptionConstants.APDM_MISSINGFIELDS_WARNING });
   // instead you can use session.enableAllWarnings()
}

public static IStatus getStatus(IWorkflow workflow, StatusConstants status)
      throws Exception {
         IStatus[] states = workflow.getStates(status);
         IStatus state = states[0];
         return state;
}
```

Getting Error Messages

If your program throws an APIException, which indicates an Agile PLM runtime error, you may want to display an error message. You can use the getMessage() method to return the error message string and then display it in a message dialog box, as shown in the following example.

Example 21–3   Getting an error message

```
// Display an error message dialog
   void errorMessage(APIException ex) {
      try {
         JOptionPane.showMessageDialog(null, ex.getMessage(),
            "Error", JOptionPane.ERROR_MESSAGE);
      } catch (Exception e) {}
   }
```

For a list of Agile PLM error messages, refer to the API Reference files (at http://edelivery.oracle.com/) under ExceptionConstants.

Disabling and Enabling Warning Messages

Several Agile PLM error messages are warnings that give you the option to stop or continue with an operation. By default, most error messages, including warning messages, are enabled. If you try to perform an action that triggers a warning, an exception will be thrown. To avoid the exception, you can disable the warning message before performing the action.

The following example shows how to check whether attempting to release a change causes an exception to be thrown. If the error code for the exception is ExceptionConstants.APDM_UNRESPONDEDCHANGE_WARNING, the program displays a warning. The user can click **Yes** in the warning dialog box to release the change.

Example 21–4 Disabling and enabling error codes

```
private void releaseChange(IAgileSession m_session, IChange chgObj) {
   IStatus nextStatus = null;

// Get the default next status
   try {
      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 APIException is Warning and not Error

As noted above, if you try to perform an operation that triggers a warning, an exception will be thrown. Warning messages are helpful for interactive GUI clients, like Agile Web Client, but you may not want to use them in your Agile API program, particularly if it performs batch processes.

You can use APIException.isWarning() to check whether an Agile PLM exception is a warning. If true, you can disable the warning and continue the operation.

Example 21–5 Checking if an APIException is a warning

```
private void checkIfWarning(IAgileSession m_session) {
   boolean gotWarning = true;
   while (gotWarning) {
   // Add some API code here that throws an exception
      try {
      m_session.doNothing();
      gotWarning = false;
      } catch (APIException e) {
      try {
         if (e.isWarning())
            m_session.disableWarning((Integer)e.getErrorCode());
            } catch (Exception ex) {}
               continue;
      }
         break;
   }
}
```

Saving and Restoring State Enabled and Disabled Warnings

Rather than keep track of which warning messages are disabled or enabled before beginning a particular operation, you can use IAgileSession.pushWarningState() to save the current state of enabled and disabled warnings. After completing the operation, you can restore the previous state of enabled and disabled warnings using IAgileSession.pushWarningState().

Example 21–6 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();
```

Warnings on Deleting Objects Disabled Automatically by Agile API

In the Agile Web Client, when you try to delete an object a warning message appears. These warning messages are not appropriate for batch processes in an Agile API program. Therefore, the Agile API implicitly disables the following warnings, which saves you the trouble of disabling them in your code.

- ExceptionConstants.APDM_HARDDELETE_WARNING
- ExceptionConstants.APDM_SOFTDELETE_WARNING

For more information about deleting objects, see "Deleting and Undeleting Objects" on page 2-21.

22

Working with Product Cost Management

This chapter includes the following:

- About Product Cost Management
- Working with Price Objects
- Working with Suppliers
- Working with Sourcing Projects

About Product Cost Management

The Product Sourcing module of the Agile PLM supports, enhances, and simplifies the handling of all product cost-related data throughout the product lifecycle. This enables you to effectively manage and manipulate Sourcing content, collaborate with suppliers to establish new Sourcing content, and analyze the data. Product Sourcing supports the following functions:

- Create Sourcing Projects
- Gather and prepare product content
- Leverage pricing contracts and history
- Create Request for Quote (RFQ) objects
- Manage supplier RFQ responses and negotiate pricing (Not supported by PCM SDK)
- Conduct Sourcing Project analysis

The Agile API supports the following Product Sourcing objects:

- `IChange` - This interface is for the Change class, which includes Price Change Orders (PCOs).
- `IPrice` - This interface is for the Price class, which handles both published prices and historical prices.
- `IProject` - This interface is for the Sourcing Projects class, which is the container used for product Sourcing data.
- `IRequestForQuote` - This interface is for the `RequestForQuote` class, which represents an RFQ for a Sourcing Project.
- `ISupplier` - This interface is for the Supplier class.

Except for the `ISupplierResponse` object, Agile API enables you to read and modify all Product Sourcing objects. The following table lists the create, read, and modify rights for Product Sourcing objects.

Table 22–1 Create, Read, and Modify privileges for Product Sourcing objects

Object	Create	Read	Modify
`IChange` (including `PCO`)	Yes	Yes	Yes
`IPrice`	Yes	Yes	Yes
`IProject`	Yes	Yes	Yes
`IRequestForQuote`	Yes	Yes	Yes
`ISupplier`	Yes	Yes	Yes

Working with Price Objects

Agile PLM's price management solution replaces inefficient manual systems, where prices are often stored in files, spreadsheets, or databases in disparate locations. The Agile PLM system allows you to create and centrally manage prices and terms for items and manufacturer parts.

There are two out-of-the-box Price classes provided with the system:

- **Historical Quotes** - This is a historical Quote object that contains price quotes from previous Sourcing Projects or legacy data.
- **Published Prices** - This is a historical Published Price that contains published prices or contract prices on current items and manufacturer parts.

These are the basic steps used to define pricing for an item or manufacturer part:

1. Users with the appropriate role can create a new Price object, specifying the Number, Description, Item or Manufacturer Part, Supplier, Site, and Customer.
2. After creating a Price object, users can build out a price/terms matrix for each associated item or manufacturer part. The price and terms matrix includes Effectivity Dates, Quantity, Price, and Cancellation Windows.
3. The Price object is submitted and goes through a Workflow approval process. Other users can approve or reject the object.
4. Users with the appropriate role can create a Price Change Order (PCO) to modify a Price object that has been released. The updated Price object is again submitted for approval.

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 22–1 Loading a Price object

```
public IPrice getPrice() throws Exception {
    IPrice price = (IPrice)m_session.getObject(IPrice.OBJECT_TYPE, "PRICE10008");
return price;
}
```

For a list of Price object tables, refer to the Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in `SDK_samples.zip`. To access this file, see the **Note** in "Client-Side Components" on page 1-2.

Adding Price Lines

The Price Lines table of a Price object is where you define the prices and terms for the related item or manufacturer part. When you add a row to the Price Lines table, you must initialize the row with values. At a minimum, you must specify values for the following attributes:

- ATT_PRICE_LINES_SHIP_FROM
- ATT_PRICE_LINES_SHIP_TO
- ATT_PRICE_LINES_PRICE_EFFECTIVE_FROM_DATE
- ATT_PRICE_LINES_PRICE_EFFECTIVE_TO_DATE
- ATT_PRICE_LINES_QTY

If you fail to specify a value for one of these attributes, the Price Lines row is not created.

Minimum Required Attributes to Create Price Lines

The minimum required attributes to create Price Lines are:

- ATT_PRICE_LINES_SHIP_TO
- ATT_PRICE_LINES_PRICE_EFFECTIVE_FROM_DATE
- ATT_PRICE_LINES_CURRENCY_CODE

Example 22–2 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. To modify a released Price object, you must first create a Price Change Order (PCO) and adding the Price object to the Affected Prices table. The PCO is then submitted for approval. Any changes made to the Price object take effect when the PCO completes its Workflow approval process.

A PCO is similar to other Change objects, such as ECOs and ECRs. You can create a PCO using the IAgileSession.createObject() method.

Example 22–3 Creating a PCO and Adding a Price

```
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);
}
```

Creating a Price Object

There are several steps to create a Price object. First, specify the object class and the unique identifying attributes, and then use `IAgileSession.createObject()` to return the new Price object.

Price objects are more complex than other Agile API objects because they have several key attributes that must be specified. Most other Agile API objects have only one key object, such as the object's number. With a Price object, you must specify a number, customer, item or manufacturer part, revision (for items), Program, site, and supplier. If any one of these attributes is missing, an exception will be thrown and the Price object won't be created.

Note: If you are not dealing with site-specific information, specify the Global site for the Manufacturing Site attribute.

After you create a Price object, you can further define it by setting values for Cover Page, Page Two, and Page Three fields. To define prices and terms for items and manufacturer parts, add rows to the Price Lines table. If there are files or documents to attach, add them to the Attachments table.

Defaults

To create a price with `Program==All` and `Customer==All`, you do not need to pass values for `PriceConstants.ATT_GENERAL_INFORMATION_CUSTOMER` and `PriceConstants.ATT_GENERAL_INFORMATION_Program` during price creation. By default, the price is created with `Program==All` and `Customer==All`.

Specifying Item Revisions

When you specify the revision of an item during price creation, you need to pass the change number, instead of the revision number.

Example 22–4 Specifying Item Revision by Passing the Change Number

```
//Pass the change number
   params.put(PricecSpecifying Item Revisionsnstants.ATT_GENERAL_INFORMATION_ITEM_
REV, "CO-35884");

//Instead of the revision number
   params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_REV, "B")
```

Creating a published price

The following example shows how to create a published price.

Example 22–5 Creating a Published Price

```
public void createPublishedPrice
   (ICustomer customer, ISupplier supplier) throws Exception {
   HashMap params = new HashMap();
   IAgileClass cls =
      m_admin.getAgileClass(PriceConstants.CLASS_PUBLISHED_PRICE);
      IAutoNumber an = cls.getAutoNumberSources()[0];
      params.put(PriceConstants.ATT_GENERAL_INFORMATION_NUMBER, an);
      params.put(PriceConstants.ATT_GENERAL_INFORMATION_CUSTOMER, customer);
      params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_NUMBER, "1000-02");
      params.put(PriceConstants.ATT_GENERAL_INFORMATION_ITEM_REV, "CO-35884");
      params.put(PriceConstants.ATT_GENERAL_INFORMATION_PROGRAM, "PROGRAM0023");
      params.put(PriceConstants.
         ATT_GENERAL_INFORMATION_MANUFACTURING_SITE, "San Jose");
      params.put(PriceConstants.ATT_GENERAL_INFORMATION_SUPPLIER, supplier);
      IPrice price = (IPrice)m_session.createObject(cls, params);
}
```

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 22–6 Loading a supplier

```
public ISupplier getSupplier() throws APIException {

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

Note: The Agile API does not support adding new rows to Supplier tables.

Modifying Supplier Data

The Agile API enables you to 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 22–7 Modifying supplier data

```
public void updateSupplierGenInfo(ISupplier supplier) throws Exception {
   ICell cell = null;
   IAgileList listvalues = null;

//Update Name (Text field)
   cell = supplier.getCell(SupplierConstants.ATT_GENERAL_INFO_NAME);
   cell.setValue("Global Parts");

//Update URL (Text field)
   cell = supplier.getCell(SupplierConstants.ATT_GENERAL_INFO_URL);
   cell.setValue("http://wwww.globalpartscorp.com");

//Update Corporate Currency (List field)
   cell = supplier.getCell(SupplierConstants.ATT_GENERAL_INFO_CORPORATE_CURRENCY);
   listvalues = cell.getAvailableValues();
   listvalues.setSelection(new Object[] { "EUR" });
   cell.setValue(listvalues);
}

public void updateSupplierContactUsers(ISupplier supplier) throws Exception {
   ICell cell = null;
   IAgileList listvalues = null;

//Load the Contact Users table
   ITable contactusers =
      supplier.getTable(SupplierConstants.TABLE_CONTACTUSERS);

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

//Update Email (Text field)
cell = row.getCell(SupplierConstants.ATT_CONTACT_USERS_EMAIL);
cell.setValue("wangsh@globalpartscorp.com");
}
```

Working with Sourcing Projects

A Sourcing Project is where you prepare content for Sourcing tasks, such as Requests for Quotes (RFQs) and Sourcing analysis. Sourcing Project is a centralized, collaborative solution. Multiple users can add data to a Sourcing Project and perform analysis of Sourcing results. Because Sourcing Projects serve as the home for all Sourcing activities and are linked to many classes of objects, including Supplier, `RequestForQuote (RFQ)`, and `SupplierResponse`.

You can use the"Supported API Methods" to:

- Load an existing Sourcing Project
- Create Sourcing Projects by quantity breaks
- Create Sourcing Projects by price periods
- Open and close a Sourcing Project
- Add items, including AMLs to Sourcing Project items
- Access and modify objects, tables, and attributes in Sourcing Projects
- Access and modify Sourcing Project status
- Update Sourcing Project AMLs
- Update Page 1, Page 2, and Page 3 in Sourcing Projects

- Read and update a nested Pricing table in Sourcing Projects
- Set quantity for an item in Sourcing Projects
- Update the target price for items in Sourcing Projects
- Set partners for items in Sourcing Projects
- Perform quantity Rollups in Sourcing Projects
- Set a response designated as *best* in Sourcing Projects

Unlike the Web Client which provides additional functionality for Sourcing Projects, the Agile API exposes Sourcing Projects for simple data extraction and updating. Consequently, the Agile API does not support the following functions:

- Validation for items, commodities, or manufacturer parts.
- Filter Sourcing Project tables
- Modify the price scenario for Sourcing Projects (change quantity breaks and effectivity periods)

Supported API Methods

The SDK supports the following API methods for Sourcing Projects. For information on these interfaces, refer to the Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in `SDK_sample.zip`. To access this file, see the **Note** in "Client-Side Components" on page 1-2.

- `IAgileSession.createObject(Object, Object)`
- `IAgileSession.createObject(int, Object)`
- `IAgileSession.getObject(Object, Object)`
- `IAgileSession.getObject(int, Object)`
- `IProject.assignSupplier (Object partnerParams)`
- `IProject.Costrollup()`
- `IProject.lookupPrices()`
- `IProject.rollupQuantity()`
- `IProject.getName()`
- `IProject.changeStatusToOpen()`
- `IProject.changeStatusToClose()`
- `IProject.getTable(Object)`
- `IRow.getValue(Object)`
- `IRow.setValue(Object, Object)`
- `ITable.iterator()`
- `ITable.getName()`
- `ITable.getTableDescriptor()`
- `ITable.size()`
- `ITable.createRow(Object)`

Note: The PCM SDK does not support the IRow.getReferent() method.

Loading an Existing Sourcing Project

To load existing Sourcing Projects, use the IAgileSession.getObject() method. To uniquely identify the Sourcing Projects, specify the value for the Cover Page | Number attribute.

Example 22–8 Loading Sourcing Project

```
public IProject getProject() throws APIException {
   String prjnum = "PRJACME_110";
   IProject prj = (IProject)m_session.getObject(IProject.OBJECT_TYPE, prjnum);
   return prj;
}
```

Creating Sourcing Projects by Quantity Breaks

The generic `IAgileSession` method supports defining Sourcing Projects.

Creating Sourcing Projects requires specifying one of the following set of parameters:

- Sourcing Project number and quantity breaks

Or,

- Sourcing Project number, quantity breaks, and price period information

Note: Quantity breaks is a required parameter and is always specified. The example below creates a Sourcing Project using the quantity break parameter.

Example 22–9 Creating a Sourcing Project by quantity break

```
IAgileClass agClass =
      m_admin.getAgileClass(ProjectConstants.CLASS_SOURCING_PROJECT);
   HashMap map = new HashMap();
   map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER, number);
   map.put(ProjectConstants.
    ATT_GENERAL_INFORMATION_NUMBER_OF_QTY_BREAKS, new Integer(4));
      IProject prj = (IProject) m_session.createObject(agClass, map);
```

Caution: Do not pass numbers that are greater than two digits to the `QUANTITY_BREAK` attribute.

Creating Sourcing Projects by Quantity Breaks and Price Periods

Alternatively, you can create Sourcing Projects by specifying quantity breaks and price period information such as the number of periods, period type, and start date. Example below creates a Sourcing Project using these parameters.

Note: When you create a Sourcing Project with price period information set to period type, you must specify the Period Type attribute. The supported values are Monthly, Quarterly, Semi-Annually, and Yearly. However, `Period Type` is not correctly returned afterwards when you check the value of period type, for example, by invoking `getValue(ProjectConstants)`**`ATT_GENERAL_INFORMATION_PERIOD_TYPE)`**. That is, instead of returning the value that you set when creating the Sourcing Project, the future returned value is always "`Weekly`."This is not an error. It is normal SDK behavior and the specified period type value is not altered, because it is for internal use only.

Example 22–10 Creating a Sourcing Project by quantity breaks and price periods

```
IAgileClass agClass =
      m_admin.getAgileClass(ProjectConstants.CLASS_SOURCING_PROJECT);
   IAutoNumber number = agClass.getAutoNumberSources()[0];
   HashMap map = new HashMap();
      map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER, number);
      map.put(ProjectConstants.
         ATT_GENERAL_INFORMATION_NUMBER_OF_QTY_BREAKS, new Integer(4));
      map.put(ProjectConstants.
         ATT_GENERAL_INFORMATION_NUMBER_OF_PERIODS, new Integer(4));
   IAgileList list =
      agClass.getAttribute(PERIODTYPE).getAvailableValues();
   String TYPE = "Monthly";
   list.setSelection(new Object[]{TYPE});
      map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PERIOD_TYPE, list);
      map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PERIOD_START_DATE,
          (new GregorianCalendar()).getTime());
   IProject prj =
      (IProject) m_session.createObject(agClass, map);
```

Importing Data for Sourcing Projects

SDK exposes the following API to support importing Items into a Sourcing Project:

- byte[] importDataForSourcingProject (String projectNumber,
- byte[] srcFile, String srcFileType-
- byte[] mappingFile,
- byte[] transformFile, String[] operations, List options)throws Exception

Example 22–11 Importing data into a Sourcing Project

```
String srcFilePath = ".//ImportSource//Assembly_BOM_AML_PC.xls";
String srcFileType = "ExcelFile";
String mappingPath = ".//ImportSource//Assembly_BOM_AML_PC.xml";
String transformPath = null; String[] operations = new String[] { "project.item",
"project.bom", "project.aml"};
List options = new ArrayList();
options.add("BusinessRuleOptions|ChangeMode=Authoring");
options.add("BusinessRuleOptions|BehaviorUponNonExistingObjects=Accept");
String _output = ".//Log//Ass_BOM_AML_PC_log.xml";options.add
      ("Template|TemplateType=com.agile.imp.template.TemplateParentChildFilter");
FileOutputStream fop = new FileOutputStream(_output);
IImportManager imgr =
       (IImportManager)session.getManager(IImportManager.class);
```

```
 byte[] logData = null;

logData = imgr.importDataForSourcingProject(projectNumber,
        stream2byte(new FileInputStream(srcFilePath)), srcFileType,
        convertFiletoStream(mappingPath),
        convertFiletoStream(transformPath), operations, options);

byte buf[] = new byte[1024 * 4];
int n = 0;
InputStream logStream = byte2stream(logData);
       while ((n = logStream.read(buf)) != -1) {
                fop.write(buf, 0, n);
       }
fop.close();
System.out.println("Import Ass_BOM_AML with ParentChild
           Template successfully, the log file is "+ _output );
```

Understanding Nested Tables of PCM Sourcing Projects

A nested table is a table within a table. They are used to accesses and modify data in multi-level objects such as BOMs and Items with AMLs. The way the SDK fulfills this function is to treat the cell values in a nested table as a table. For example, when the SDK finds the next level in a cell in a BOM table, it treats and processes the cell as a table. Nested tables are unique to the PCM SDK.

Sourcing Projects' Parent Table and Nested Child Table Constants

The list of parent Sourcing Projects table and the corresponding nested child table constants appear in the Parent Sourcing Projects Table Constants and the Corresponding Nested Sourcing Projects Tables.

Table 22–2 Parent Sourcing Projects Table Constants and Nested Child Table Constant

Parent Table Constant	Nested Child Table Constant	Read/Write Mode
TABLE_ITEMS	ATT_ITEMS_AML	Read/Write
TABLE_ITEMS	ATT_ITEMS_PRICING	Read/Write
TABLE_AML	ATT_AML_PRICETABLE	Read/Write
TABLE_ITEM	ATT_ITEM_PRICE_TABLE	Read/Write
TABLE_ITEM	ATT_ITEM_BOM_TABLE	Read
TABLE_ANALYSIS	ATT_ANALYSIS_AML	Read
TABLE_ANALYSIS	ATT_ANALYSIS_PRICING	Read

Accessing and Modifying Nested Tables in Sourcing Projects or RFQs

The example below is a Read mode example that accesses a nested table. To modify/update a nested table, see Example 22–41, "Nested RFQ table update"

Note: The Money type attribute in nested PCM Pricing tables always uses “USD” as the default currency unit. This applies even if the buyer specifies a different currency unit. In this case, the “United State Dollar” is the default and only supported currency.

Example 22–12 Accessing a nested table

```
IRow row = (IRow) table.iterator.next();
```

```
ITable nested_table =
(ITable)row.getValue(ProjectConstants.ATT_ITEMS_AML);
```

Viewing Updates after Modifying a Nested Table

After modifying a nested table, it is necessary to reload the table as shown in the previous example for changes to take effect, otherwise, the old data will reappear and the new values are not displayed.

Accessing and Modifying the Status of Sourcing Project

Because Sourcing Projects do not have a Workflow connected to them, their status change is controlled internally. They control their status with a set of methods. This is a special case for some PCM objects such as Sourcing Projects and Requests for Quote. This release supports changing the status of a Sourcing Project from Draft to Open and Open to Close.

You can access the status of Sourcing Projects using the standard `IDataObject` method for the lifecycle phase field on the Cover Page (Page 1). You can modify the status of Sourcing Projects with `IProject` methods which enable you to open, modify, and close a Sourcing Project. You must set the Ship to Location parameter in order to open a Sourcing Project, as shown in the following example.

Example 22–13 Setting values for Sourcing Project's Cover Page and Open and then Close the Project

```
// add Ship To //
String sj = "San Jose";
IAgileList ship2List =
(IAgileList)prj.getValue(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_
LOCATION);
ship2List.setSelection(new Object[]{sj});
prj.setValue(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,
ship2List);
// open Sourcing Project //
prj.changeStatusToOpen();
// close Sourcing Project //
prj.changeStatusToClose();
```

Managing Data in Sourcing Projects

The following paragraphs provide descriptions and examples to prepare a Souring Project to issue an RFQ. You can then use the SDK to complete the RFQ-related tasks.

Note: The *Sourcing Project Start* date that you specify is converted to the GMT format for storage in the PLM database. Due to this conversion, the date value returned by

```
IProject.getValue(ProjectConstants.ATT_GENERAL_INFORMATION_
PERIOD_START_DATE)
```

is not guaranteed to be the same that the user may expect.

Setting Quantity for Items in Sourcing Projects

You can use the SDK to set the required quantity for the an Item object in a Sourcing Project. The code sample below sets this value in the Item table, under the Items tab, for a single price target. The end user can specify the target price using the displayed name which is `QuantityBreak2`

Example 22–14 Setting quantity for Items

```
/ Setting Quantity for an Item //
   ITable tab_item = dObj.getTable(ProjectConstants.TABLE_ITEM);
   IRow row = (IRow) tab_item.iterator().next()
   ITable priceTable =
      row.getValue(ProjectConstants.ATT_ITEM_PRICE_TABLE); XXXXX
   for (Iterator iterator =
      priceTable.iterator(); iterator.hasNext();) {
   Row row = (IRow) iterator.next();
   String name = row.getName();
      if(name.equals("QuantityBreak2")){
      row.setValue(ProjectConstants.ATT_PRICEDETAILS_QUANTITY,
         new Double(123));
      }
   }
```

Note: For items, quantity is only set at the root level. Thus, if an item is not a root, the exception `ExceptionConstants.PCM_PROJECT_ITEM_IS_NOT_ROOT` is thrown.

In addition, because `priceTable` is a nested table, you must reload the table to get the updated value of Quantity. This is shown in the following example.

Example 22–15 Reloading a nested table to get an updated value

```
// Getting the updated value //
   priceTable =
      row.getValue(ProjectConstants.ATT_ITEM_PRICE_TABLE); XXXXX
   for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
   IRow iRow = (IRow) iterator.next();
   String name = iRow.getName();
      if(name.equals("QuantityBreak2")){
         Object qty =
            row.getValue(ProjectConstants.ATT_PRICEDETAILS_QUANTITY));
      }
   }
```

Adding Items to Sourcing Projects Using BOM Filters

The PLM Web Client supports setting up BOM filters to selectively add Items to Sourcing Projects. This filtering applies to all enabled attributes on the Cover page, P2, and BOM tabs of the Item object and includes the Parts and Documents fields in the Cover page. These attributes are enabled using the Java Client. For background information and procedures on BOM filtering, refer to *Agile PLM Product Cost Management User Guide.* To enable Item object attributes, refer to *Agile PLM Administrator Guide*.

The SDK enables replicating this Web Client feature programmatically. SDK filter operators that support this feature are implemented in the `OperatorConstants` class. The following code samples show how BOM filters are applied to Item object's *numeric, multilist, money*, and *mass* attributes. As indicated above, these attributes must be enabled in the Java Client by an Admin user.

Example 22–16 Applying BOM filter operators located in the OperatorConstants class

```
IProject prj =
   (IProject)m_session.getObject(IProject.OBJECT_TYPE, "PRJ00001");
IItem assembly =
   (IItem)m_session.getObject(IItem.OBJECT_TYPE, "P00001");

//Applying BOM filter to numeric attributes
   ProjectItemFilter itemfilter = new ProjectItemFilter();
   itemfilter.addCriteria(ItemConstants.ATT_BOM_BOM_NUMERIC03,
   OperatorConstants.RELOP_EQ, new Integer(10));
   itemfilter.addCriteria(ItemConstants.ATT_BOM_BOM_NUMERIC04,
   OperatorConstants.RELOP_GE, new Integer(100));
   Map params = new HashMap();
   params.put(ProjectConstants.ATT_ITEM_NUMBER, assembly);
   params.put(ProjectConstants.ATT_ITEM_FILTER, itemfilter);
   ITable ITEM = prj.getTable(ProjectConstants.TABLE_ITEM);
   IRow row = ITEM.createRow(params);

//Applying BOM filter to multilist attributes
   throws APIException, RemoteException, Exception;
   ProjectItemFilter itemfilter = new ProjectItemFilter();
IAttribute list03 =
      m_session.getAdminInstance().getAgileClass(ItemConstants.CLASS_PARTS_
   CLASS).getAttribute(ItemConstants.ATT_PAGE_TWO_MULTILIST01);
   IAgileList list3 = list03.getAvailableValues();
   list3.setSelection(new Object[]{"Austria","India"});
   itemfilter.addCriteria(ItemConstants.ATT_PAGE_TWO_MULTILIST01,
   OperatorConstants.RELOP_CONTAINS_ALL_VALUE, list3);
   Map params = new HashMap();
   params.put(ProjectConstants.ATT_ITEM_NUMBER, assembly);
   params.put(ProjectConstants.ATT_ITEM_FILTER, itemfilter);
   ITable ITEM = prj.getTable(ProjectConstants.TABLE_ITEM);
   IRow row = ITEM.createRow(params);

//Applying BOM filter to money attributes
   Money mny = new Money(new Double(15.3), "USD");
   ProjectItemFilter itemfilter = new ProjectItemFilter();
   itemfilter.addCriteria(ItemConstants.ATT_PAGE_TWO_MONEY01,
   OperatorConstants.RELOP_EQ, mny);
   Map params = new HashMap();
   params.put(ProjectConstants.ATT_ITEM_NUMBER, assembly);
   params.put(ProjectConstants.ATT_ITEM_FILTER, itemfilter);
   ITable ITEM = prj.getTable(ProjectConstants.TABLE_ITEM);
   IRow row = ITEM.createRow(params);

//Applying BOM filter to mass attributes
   IUnitOfMeasureManager uomm =(IUnitOfMeasureManager)m_
   session.getManager(IUnitOfMeasureManager.class);
   IUnitOfMeasure uom = uomm.createUOM(10.1,"Gram");
   ProjectItemFilter itemfilter = newProjectItemFilter();
   itemfilter.addCriteria(ItemConstants.ATT_TITLE_BLOCK_
      MASS,OperatorConstants.RELOP_NEQ, uom);
   Map params = new HashMap();
   params.put(ProjectConstants.ATT_ITEM_NUMBER, assembly);
   params.put(ProjectConstants.ATT_ITEM_FILTER, itemfilter);
   ITable ITEM = prj.getTable(ProjectConstants.TABLE_ITEM);
   IRow row = ITEM.createRow(params);
```

Updating the QPA Attribute on the Item Table of Sourcing Projects

SDK supports updating the Quantity per Assembly (QPA) attribute on the Item Table of Open and Draft PCM Sourcing Projects. You can set the value of the QPA attributes directly in the Sourcing Project, or by using the values from the Item Master file. Although you can pass Sourcing Projects that their status is are neither Draft or Open, but the Agile code will automatically determine whether to update, or not update the QPA attribute

Example 22–17 Updating the QPA Attribute on the Item Table of Sourcing Projects

```
public static void main(String[] args) {
   try {

   IProject prj = (IProject)session.getObject(IProject.OBJECT_TYPE,"PRJ00011");
   ITable items = prj.getTable(ProjectConstants.TABLE_ITEM);
   Iterator iter = items.iterator();
   while (iter.hasNext()){
      IRow itemRow = (IRow)iter.next()
      ITable bom = (ITable) itemRow.getValue(ProjectConstants.ATT_ITEM_BOM_TABLE);
      Iterator bomIter = bom.iterator();
      while (bomIter.hasNext()){
         IRow bomRow = (IRow) bomIter.next()

         // This example sets the Sourcing Project's QPA attribute to 5.
         bomRow.setValue(ProjectConstants.ATT_ITEM_QPA, 5);
         }
      }
   }catch (Exception e) {
      e.printStackTrace();
   } finally {
      session.close();
      }
}
```

Example 22–18 Updating Selected Attributes on Item Table of Sourcing Projects from Item Master File

```
public static void updateContentFromItemMaster() throws Exception {
   IProject prj =
      (IProject)session.getObject(IProject.OBJECT_TYPE, "PRJ00011);
   Map params = new HashMap();
   params.put(ProjectConstants.FLAG_CONTENT_UPDATE_MODE_OPTION_ALL, false);
   params.put(ProjectConstants.FLAG_CONTENT_UPDATE_MAINTAIN_AML_CHANGES, false);
   params.put(ProjectConstants.
      FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_BOM_AND_AML, true);
         params.put(ProjectConstants.
            FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_ITEM, true);
   params.put(ProjectConstants.
      FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_MANUFACTURER_PART, true);
   List itemAttributes = new ArrayList();
   itemAttributes.add(ProjectConstants.ATT_ITEM_QPA);

// choose QPA attribute to update
   params.put(ProjectConstants.
      FIELD_CONTENT_UPDATE_ITEM_ATTRIBUTES, itemAttributes);
   List mfrPartAttributes = new ArrayList();
   mfrPartAttributes.add("description");
   params.put(ProjectConstants.
      FIELD_CONTENT_UPDATE_MFR_PART_ATTRIBUTES, mfrPartAttributes);
         System.out.println("Updating Content from Item Master....");
   prj.updateContentFromItemMaster(params);
      System.out.println("Finish updating Content from Item Master....");
}
```

Updating PLM content from the Item Master

The following API supports updating PLM content including the Universal Unit of Measure Attribute (UOM) from the Item master file:

```
void IProject.updateContentFromItemMaster (Object contentParams) throws
APIException, RemoteException, Exception
```

Example 22–19   Updating content from Item Master file

```
   IProject prj =
      (IProject)session.getObject(IProject.OBJECT_TYPE, "PRJ000XX");
   Map params = new HashMap();
   params.put(ProjectConstants.FLAG_CONTENT_UPDATE_MODE_OPTION_ALL, false);
   params.put(ProjectConstants.FLAG_CONTENT_UPDATE_MAINTAIN_AML_CHANGES, false);
   params.put(ProjectConstants.
      FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_BOM_AND_AML, true);
   params.put(ProjectConstants.
      FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_ITEM, true);
   params.put(ProjectConstants.
      FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_MANUFACTURER_PART, true);
   List itemAttributes = new ArrayList();
   itemAttributes.add(ProjectConstants.ATT_ITEMS_DESCRIPTION);
   itemAttributes.add(ProjectConstants.ATT_ITEMS_ATTACHMENTS);
   itemAttributes.add(ProjectConstants.ATT_ITEMS_UOM); //'UOM' attribute

   params.put(ProjectConstants.
      FIELD_CONTENT_UPDATE_ITEM_ATTRIBUTES, itemAttributes);
   List mfrPartAttributes = new ArrayList();
   mfrPartAttributes.add("description");
   mfrPartAttributes.add("attachments");
   params.put(ProjectConstants.FIELD_CONTENT_UPDATE_MFR_PART_ATTRIBUTES,
      mfrPartAttributes)
   prj.updateContentFromItemMaster(params);
```

Updating Selected Attributes on Item Table of Sourcing Projects from Item Master

Use public static void updateContentFromItemMaster API as shown in the following example.

Example 22–20   Updating Item Table of Sourcing Project's attributes from Item Master

```
public static void updateContentFromItemMaster() throws Exception {
IProject prj = (IProject)session.getObject(IProject.OBJECT_TYPE, "PRJ00011");
Map params = new HashMap();
params.put(ProjectConstants.FLAG_CONTENT_UPDATE_MODE_OPTION_ALL, false);
params.put(ProjectConstants.FLAG_CONTENT_UPDATE_MAINTAIN_AML_CHANGES, false);
params.put(ProjectConstants.
          FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_BOM_AND_AML, true);
params.put(ProjectConstants.
          FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_ITEM, true);
params.put(ProjectConstants.
          FLAG_CONTENT_UPDATE_SELECTED_CONTENT_OPTION_MANUFACTURER_PART, true);
List itemAttributes = new ArrayList();
itemAttributes.add(ProjectConstants.ATT_ITEM_QPA);

// choose QPA attribute to update
params.put(ProjectConstants.FIELD_CONTENT_UPDATE_ITEM_ATTRIBUTES, itemAttributes);
List mfrPartAttributes = new ArrayList();
mfrPartAttributes.add("description");
params.put(ProjectConstants.FIELD_CONTENT_UPDATE_MFR_PART_ATTRIBUTES,
      mfrPartAttributes);
     System.out.println("Updating Content from Item Master....");
prj.updateContentFromItemMaster(params);
System.out.println("Finish updating Content from Item Master....");
}
```

Performing Quantity Rollups in Sourcing Projects

Quantity rollups generate data related the quantity values for the selected Item in a Sourcing Project. In the SDK, you can use the following API to invoke a Quantity rollup in a Sourcing Project.

```
public void rollupQuantity() throws APIException, RemoteException,
Exception;
```

This code sample uses `rollupQuantity()` to perform the Quantity rollup.

Example 22–21 Quantity Rollup

```
IProject prj =
(IProject)m_session.getObject(ProjectConstants.CLASS_SOURCING_PROJECT,"PRJ00001");
prj.rollupQuantity();
```

Note: To get the updated value of Quantity, it is necessary to invoke rollupQuantity() on the Sourcing Project similar to the example in "Modifying Supplier Data" on page 22-6. This is necessary because `getValue()` does not return the updated value of the affected item after setting Quantity.

Performing Cost Rollup in Sourcing Projects

Cost Rollup (Rollup cost) generates an Assembly Cost Report (ACR) based on available prices. In this process, it picks up the lowest costs from filtered data, performs Set as Best (on user defined or default parameters) and costed BOM rollup (aggregation) to generate the ACR. In the UI, Rollup cost provides an intuitive mechanism for non PCM users to cost a BOM without going through PCM steps.

Note: Cost Rollup runs on existing Sourcing Project prices. If Cost Rollup needs to run on looked up prices, `lookupPrices()` must be invoked prior to running `costRollup()`. If there are no assemblies in the Sourcing Project, the `ExceptionConstants.PCM_NO_ASSEMBLY_IN_PROJECT` is thrown.

The PCM SDK supports the Cost Rollup function with the following API.

```
public void costRollup()
throws APIException, RemoteException, Exception;
```

Example 22–22 Using costRollup

```
IProject prj =
(IProject) m_session.getObject(ProjectConstants.CLASS_SOURCING_PROJECT,
"PRJ0001");
prj.costRollup();
```

Note: If you need to run quantity rollup immediately after cost rollup, be sure to provide some delay (For example as in `Thread.currentThread().sleep(10000);`) to allow the results of the cost rollup to be refreshed in the database.

Performing Price Lookup in Sourcing Projects

You can use the SDK to verify the existence of a price scenario for a specified period and quantity in the Item Master. You can either use the price information of the Item, or modify the price information and send the RFQ to suppliers for requote. This is shown in the following code samples.

In Agile PCM, there are three types of price objects:

- **Contracts** - Predefined agreements with suppliers for Item prices over a specified time period
- **Published Prices** - The Item price information that has been published from other Sourcing Projects
- **Quote Histories** - Quoted prices that were previously received for an Item

For information about price objects and price lookups in Sourcing Projects, refer to the Agile PLM *Product Cost Management User Guide*.

Price Lookup API and Price Options

- **Supported API** - The SDK supports price lookups with this IProject API:

  public void lookupPrices(Object lookupParams)   throws APIException, RemoteException, Exception;

- **Price lookup options** - This API performs price lookups from Price history and price lookups from another Sourcing Project

---

Note: `lookupPrices()looks` for an Item or an MPN one object at a time. To run lookup for multiple Items/MPNs, you must run the API one Item or one MPN at a time.

The following examples show price lookups from the Sourcing Project History and from another Sourcing Project. In addition, applicable parameters are grouped and listed as those that are specific to the price lookup type and necessary in the price lookup type.

This example shows a price lookup from Sourcing Project History and from another Sourcing Projects. It provides a list of specific and required parameters for the two price lookups.

Example 22–23 Price lookup from History and another Sourcing Projects

```
ArrayList priceTypes = new ArayList();
priceTypes.add(PriceConstants.CLASS_PUBLISHED_PRICE);
priceTypes.add(PriceConstants.CLASS_QUOTE_HISTORY);
priceTypes.add(PriceConstants.CLASS_CONTRACT);

//supplier1, supplier2 are objects of ISupplier or String
ArrayList suppliers = new ArrayList();
suppliers.add(supplier1);
suppliers.add(supplier2);
```

```
//customer1, customer2 are objects of ICustomer or String
ArrayList customers = new ArrayList();
customers.add(customer1);
customers.add(customer2);

//program1, program2 are objects of IProgram or String
ArrayList programs = new ArrayList();
programs.add(program1);
programs.add(program2);

//program1, program2 are objects of  IProgram or String
items[0] = "P00001";
String shipTo = "berlin";
HashMap itemMap = new HashMap();
itemMap.put("IPN1","REV1");
//   itemMap.put("IPN1", null) if no revision
//or
//   itemMap.put(item); //item is an object of IItem
HashMap mpnMap = new HashMap();
mpnMap.put("MPN1","MFR1");
//or
mpnMap.put(mfrPart); //mfrPart is and object of IManufacturerPart

IProject srcPrj = (IProject)
m_session.getObject(ProjectConstants.CLASS_SOURCING_PROJECT, "PRJ_SRC");
Boolean isLookupFromPrice = new Boolean(false);
String priceScenario = null;
Map priceScenarios = new HashMap();

//if lookup from price history
priceScenario = "QuantityBreak1";

//if lookup from Sourcing Project
String destPricePoint1 = "QuantityBreak1";
String destPricePoint2 = "QuantityBreak2";
String srcPricePoint1 = "QuantityBreak1";
String srcPricePoint2 = "QuantityBreak2";
priceScenarios.put(destPricePoint2,srcPricePoint1);
priceScenarios.put(destPricePoint1,srcPricePoint2);
Boolean ignoreQtyRange = new Boolean(true);
Double qtyPercentRange = new Double(15);
Boolean ignoreDateRange = new Boolean(true);
Integer dateRange = new Integer(20);
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipTo);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER,srcPrj);
map.put(LookupConstants.FLAG_IGNORE_ITEM_REVISION,ignoreItemRev);
map.put(LookupConstants.FLAG_CONSIDER_BEST_PRICES,considerBestPrices);
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FIELD_PRICE_SCENARIOS,priceScenarios);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.FIELD_SELECT_RESPONSE_BY,
LookupConstants.OPTION_LOWEST_PRICE);
map.put(LookupConstants.FIELD_LOOKUP_TYPE,
LookupConstants.OPTION_LOOKUP_FROM_PRICE);
prj.lookupPrices(map);
```

Parameters specific to price lookups from price history

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER
ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM
ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION
LookupConstants.FLAG_ALL_PRICE_SCENARIOS
LookupConstants.FIELD_PRICE_SCENARIO
LookupConstants.FLAG_IGNORE_QUANTITY
LookupConstants.FIELD_QUANTITY_RANGE
LookupConstants.FLAG_IGNORE_DATE_RANGE
LookupConstants.FIELD_DATE_RANGE
LookupConstants.FIELD_SELECT_RESPONSE_BY
```

Parameters specific to price lookups from Sourcing Projects

```
ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER
LookupConstants.FLAG_ALL_PRICE_SCENARIOS
LookupConstants.FLAG_IGNORE_ITEM_REVISION
LookupConstants.FLAG_CONSIDER_BEST_PRICE
```

Note: The remaining parameters are common to both cases.

Required parameters for price lookups from price history

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
LookupConstants.FIELD_QUANTITY_RANGE if LookupConstants.FLAG_IGNORE_QUANTITY is
'false'
LookupConstants.FIELD_DATE_RANGE if LookupConstants.FLAG_IGNORE_DATE_RANGE is
'false'
LookupConstants.FIELD_PRICE_SCENARIO if LookupConstants.FLAG_ALL_PRICE_SCENARIOS
is 'false'
```

Setting the price lookup from Sourcing Projects

```
ProjectConstants.ATT_GENERAL_INFORMATION_NUMBER
LookupConstants.FLAG_ALL_PRICE_SCENARIOS
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
```

Note: Parameters that are not required in one of the price lookups, for example, History, may be optional in price lookup from another Sourcing Project. Comparing the list of required parameters above, the `LookupConstants.FLAG_ALL_PRICE_SCENARIOS` parameter is optional when performing a price lookup from History. You can either omit the optional parameters, or set them to null.

Setting the price lookup from History or Sourcing Projects

Set `LookupConstants.FIELD_LOOKUP_TYPE` for lookup from history or Sourcing Projects as follows:

- For lookup from price history - `LookupConstants.OPTION_LOOKUP_FROM_PRICE`
- For lookup from an existing Sourcing Project - `LookupConstants.OPTION_LOOKUP_FROM_PROJEct`

Settings for Quantity Breaks in price lookups

You can set quantity breaks in a price lookup by cost, date, or leadtime by setting `LookupConstants.FIELD_SELECT_RESPONSE_BY` as follows:

- For break tie by cost - `LookupConstants.OPTION_LOWEST_PRICE`
- For break tie by date - `LookupConstants.OPTION_MOST_RECENT_RESPONSE`
- For break tie by leadtime- `LookupConstants.OPTION_SHORTEST_LEAD_TIME`

Impact of improper parameter settings

If the following parameters are not set, or are improperly set, APIs will take the following actions:

- `LookupConstants.FIELD_LOOKUP_TYPE` will default to `LookupConstants.OPTION_LOOKUP_FROM_PRICE` which corresponds to the lookup from price history
- `LookupConstants.FLAG_IGNORE_QUANTITY` or `LookupConstants.FLAG_IGNORE_DATE_RANGE` will default to true.
- `LookupConstants.FIELD_SELECT_RESPONSE_BY` will default to `LookupConstants.OPTION_LOWEST_PRICE` which corresponds to the break tie by cost
- `LookupConstants.FLAG_IGNORE_ITEM_REVISION` or `LookupConstants.FLAG_CONSIDER_BEST_PRICES` will default to false
- `LookupConstants.LookupConstants.FLAG_ALL_PRICE_SCENARIOS` if not set, will default to true.
- `ExceptionConstants.APDM_ADMIN_MISSINGREQUIREDFIELD` exception is thrown when a required parameter is missing
- `ExceptionConstants.API_INVALID_PARAM` exception is thrown when the datatype or the value of a parameter is incorrectly set

Settings for RFQ lookup

The settings are similar to that of Sourcing Project lookups from price history. Following is a sample code.

```
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipTo);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);-
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.
          FELD_SELECT_RESPONSE_BY,LookupConstants.OPTION_LOWEST_PRICE);
map.put(LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER,excludeAuthSupplier);
rfq.lookupPrices(map);
```

Settings for RFQ lookup

The settings are similar to that of Sourcing Project lookups from price history. Following is a sample code.

```
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipTo);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);-
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.
          FELD_SELECT_RESPONSE_BY,LookupConstants.OPTION_LOWEST_PRICE);
map.put(LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER,excludeAuthSupplier);
rfq.lookupPrices(map);
```

Required parameters for an RFQ lookup

```
- PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
- ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers
- ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
- LookupConstants.FIELD_QUANTITY_RANGE if
- LookupConstants.FLAG_IGNORE_QUANTITY is 'false'
- LookupConstants.FIELD_DATE_RANGE if
- LookupConstants.FLAG_IGNORE_DATE_RANGE is 'false'
- LookupConstants.FIELD_PRICE_SCENARIO if
- LookupConstants.FLAG_ALL_PRICE_SCENARIOS is 'false'
```

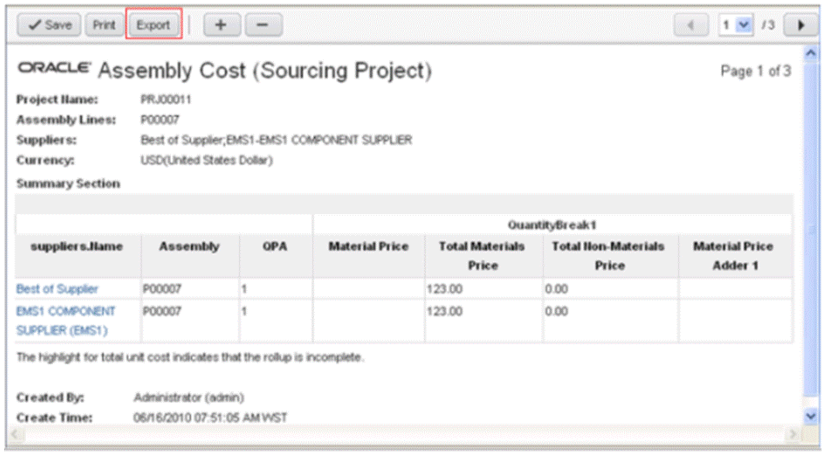
Note: If you do not set the value of LookupConstants.FLAG_ EXCLUDE_AUTH_SUPPLIER, it will default to false.

Generating the Assembly Cost Report for Sourcing Projects

The PLM Web Client supports generating the Assembly Cost Report to help understand the total cost of a BOM, including the material and non-material costs. To simplify this operation, the Web Client provides a wizard to locate and select the required Sourcing Project, including the necessary parameters. See Understanding Assembly Cost Report Parameters for a description of these parameters.

A sample report is shown in the following screen shot. Once the report is prepared, the PLM user can invoke the **Export** command and send the data from the PLM to an external device in a supported format, for example, to an Excel worksheet.

Figure 22–1 Sample Assembly Cost Report



ORACLE Assembly Cost (Sourcing Project)

Page 1 of 3

Project Name:	PRJ00011
Assembly Lines:	P00007
Suppliers:	Best of Supplier;EMS1-EMS1 COMPONENT SUPPLIER
Currency:	USD(United States Dollar)

Summary Section

			QuantityBreak1			
suppliers.Name	**Assembly**	**QPA**	**Material Price**	**Total Materials Price**	**Total Non-Materials Price**	**Material Price Adder 1**
Best of Supplier	P00007	1		123.00	0.00	
EMS1 COMPONENT SUPPLIER (EMS1)	P00007	1		123.00	0.00	

The highlight for total unit cost indicates that the rollup is incomplete.

Created By:	Administrator (admin)
Create Time:	06/16/2010 07:51:05 AM WST

Agile SDK supports this Web Client feature by enhancing the ProductReport.execute() API. The extent of the SDK operation does not include invoking the Export command. The following is a code sample that uses the `ProductReport.execute()` API and the required parameters to programmatically generate this report.

Example 22–24 Generating Assembly Cost Reports

```
String[] items = new String[1];
String[] suppliers = new String[2];
suppliers[0] = "-101";  //to include data for best suppliers
suppliers[1] = "EMS1 COMPONENT SUPPLIER";
String[] priceScenarios = new String[1];
priceScenarios[0] = "QuantityBreak1";
Boolean doCostRollup = new Boolean(true);
HashMap map = new HashMap();
   map.put(ProductReportConstants.REPORTPARAM_REPORT_
   TYPE,ProductReportConstants.REPORT_PROJECT_ASSEMBLY_COST);
   map.put(ProductReportConstants.REPORTPARAM_REPORT_CATEGORY, "data");
   map.put(ProductReportConstants.PROJECT_NUMBER, "PRJ00001");
   map.put(ProductReportConstants.PROJECT_ITEMS, items);
   map.put(ProductReportConstants.PROJECT_SUPPLIERS, suppliers);
   map.put(ProductReportConstants.PROJECT_PRICEPOINTS, priceScenarios);
   map.put(ProductReportConstants.PROJECT_DO_COST_ROLLUP, doCostRollup);
try{
   IProductReport report = (IProductReport)
      m_session.createObject(IProductReport.OBJECT_TYPE, "My Reports");
          String result =report.execute(map);
   }
      catch (Exception e){
         e.printStackTrace();
}
```

Note: The returned value of the `ProductReport.execute()` API represents either the XML data of the Assembly Cost Report or the XML schema. To get the schema, you must set the values of the first two parameters in the map where the value of the parameter `REPORTPARAM_REPORT_CATEGORY` is set to `schema`.

Understanding Assembly Cost Report Parameters

The Assembly Cost Report parameters supported by `ProductReport.execute()` are defined as follows:

- `items` - The array of top level assembly items: "`<itemNumber>::<revNumber>`"if there is a revision, and "`<itemNumber>`"if there are no revisions.
- `suppliers` - The array of supplier names along with the optional indicators for the "`Best of Suppliers": "-101`"or the "`Best Of Suppliers/Partners`"
- `pricescenarios` - The array of price scenarios names
- `doCostRollup` - The flag to run the cost Rollup option for the report

If you want to generate a report for all assemblies, all suppliers/partners, or all price scenarios without specifying their names, you must provide the following corresponding parameters:

-
  ```
  String[] items = new String[1];
  items[0] = "all";
  ```

-
  ```
  String[] suppliers = new String[2];
  suppliers[0] = "101";
  suppliers[1] = "all"
  ```

-
  ```
  String[] priceScenarios = new String[0];
  priceScenarios[0] = "all";
  ```

Modifying the Target Price for Items in Sourcing Projects

Target Price is the market cost per unit of the item or the manufacturer part. It is specified when items are ordered. For each Item and for each Pricepoint, Target Price is set in the Items table, under the AML tab. A Pricepoint is the Target price quoted for a given quantity for an Item. For example, price quoted for X number of tires, which can be different for Y number of the same tires.

Note: The Target price is always a positive number. Setting a negative value for Target price will throw the `ExceptionConstants.PCM_NEGATIVE_TARGET_PRICE` exception.

Target Price is set at the Item level only. You cannot set a Target Price the AML level. The end user specifies a Pricepoint using the name displayed for the Pricepoint. In the following example, `QuantityBreak2` is the Pricepoint.

Example 22–25 Setting the Target price in Sourcing Projects

```
ITable tab_item = dObj.getTable(ProjectConstants.TABLE_ITEMS);
IRow row = (IRow) tab_item.iterator().next();
ITable priceTable = row.getValue(ProjectConstants.ATT_ITEMS_PRICING);
for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
IRow row = (IRow) iterator.next();
String name = row.getName();
   if(name.equals("QuantityBreak2")){
     row.setValue(ProjectConstants.ATT_PRICEDETAILS_TARGET_COST,
         new Money(new Double(1.23), "USD") );
   }
}
```

Because `priceTable` is a nested table, you must reload this table to get the updated value of the Target price as shown in the following example. This is similar to the example in "Setting Quantity for Items in Sourcing Projects" on page 22-12.

Example 22–26 Reloading the priceTable to get the updated value of the target price

```
priceTable = row.getValue(ProjectConstants. ATT_ITEMS_PRICING);
for (Iterator iterator = priceTable.iterator(); iterator.hasNext();) {
IRow iRow = (IRow) iterator.next();
String name = iRow.getName();
    if(name.equals("QuantityBreak2")){
       Object qty = row.getValue(ProjectConstants.ATT_PRICEDETAILS_TARGET_COST));
    }
}
```

Setting the Best Response for Items in Sourcing Projects

The Best Response is set in the Analysis table under the Analysis tab for both the Item and Manufacturer Part number objects. The end user specifies three of these parameters: Lowest Cost, Lowest Cost Within Lead Time Constraint, Shortest Lead Time, Supplier Rating, and AML Preferred status. For more information, refer to *Agile Product Lifecycle Management - Product Cost Management Supplier Guide*.

You can use the SDK to find the best response for an Item Part Number (IPN), a Manufacturer Part Number (MPN), and for an IPN and an MPN as shown in the following code samples.

Example 22–27 Setting the Best Response for an IPN

```
ITable table_analysis = prj.getTable(ProjectConstants.TABLE_ANALYSIS);
   Iterator it = table_analysis.iterator();
while(it.hasNext()) {
   IRow row = (IRow) it.next();
   String itemName = row.getValue(ProjectConstants.ATT_ANALYSIS_NUMBER);
   String suppName = row.getValue(ProjectConstants.ATT_ANALYSIS_SUPPLIER);
   if (itemName.equals("IPN1") && suppName.equals("suppName1 (suppNumber1)")) {
      row.setValue(ProjectConstants.ATT_ANALYSIS_BEST_RESPONSE, "Yes");
   }
}
```

Example 22–28 Setting the Best Response for an MPN

```
ITable table_aml =
(ITable) row.getValue(ProjectConstants.ATT_ANALYSIS_AML);
Iterator _it = table_aml.iterator();
while(it.hasNext()){
   String itemName = row.getValue(ProjectConstants.ATT_ANALYSIS_NUMBER);
   String suppName = row.getValue(ProjectConstants.ATT_ANALYSIS_SUPPLIER);
   String mfrName = row.getValue(ProjectConstants. ATT_ANALYSIS_MANUFACTURER);
   if (itemName.equals("MPN1") && suppName.equals("suppName1
      (suppNumber1)" && mfrName.equals("MFR1"))) {
row.setValue(ProjectConstants.ATT_ANALYSIS_BEST_RESPONSE, "Yes");
      }
}
```

Note: Because you can only set the Best Response to Yes, if you pass any value other than Yes, the ExceptionConstants.API_INVALID_ PARAM exception is thrown.

Example 22–29 Getting the Best Response for an IPN and an MPN

```
String bResp =
row.getValue(ProjectConstants.ATT_ANALYSIS_BEST_RESPONSE).toString();
```

Setting Partners in Sourcing Projects

Partners can view complete Sourcing Project BOMs in RFQs. You can assign partners to an item in the Sourcing Project when you add the item to the RFQ that will be sent to the partners. If multiple partners are selected, you can split the quantity among the partners by specifying what percentage of an item you want to receive from each supplier. For example, if two partners supply the same item, you can add both partners to the list and then assign a certain percentage to each, for example, 50%-50%, or 60%-40%, and so on.

In the SDK, the following API is used to set partners for an item in a Sourcing Project and split the percentage among the partners.

The behavior of this API and its use cases are similar to those of `IRequestForQuote.assignSupplier()`. However, when you add new partners for an item with this API, you will override the existing ones. Thus, to avoid removing existing partners, it is necessary to once again add the existing partners and set the split (Percentage for each) level for each one. This only occurs in the SDK, the GUI does not require adding the existing partners when you add new partners. You cannot remove a partner for an item, but assigning a split = 0, (Percentage of ownership/participation) will remove the partner. For more information on the GUI behavior, refer to the latest release of *Agile Product Lifecycle Management - Product Cost Management Supplier Guide*.

The following code sample sets partners and splits the percentages among the assigned partners.

Example 22–30 Setting partners and splitting percentages among partners

```
HashMap map = new HashMap();
HashMap supplierSplit = new HashMap();
HashMap partnerMap = new HashMap();
map.put(ProjectConstants.ATT_ITEM_NUMBER, item);
Double split1 =
new Double(55);
Double split2 =
new Double(75);
supplierSplit.put(supplier1, split1);
supplierSplit.put(supplier2, split2);
partnerMap.put(ProjectConstants.ATT_PARTNERS_PARTNER, supplierSplit);
map.put(ProjectConstants.ATT_ITEM_PARTNER_TABLE, partnerMap);
prj.assignSupplier(map);
```

An `item` or `supplier` can be an `IItem` object, a `ISupplier` object, or a String object. Partners can be assigned to any `Item Part Number (IPN)`, but not a `Manufacturer Part Number (MPN)`. If the item is not in the Sourcing Project, the `ExceptionConstants.PCM_ERROR_INVALID_PROJECT_ITEM is thrown.`

Split percentages can be any object representing a number. If it is not a number, the `ExceptionConstants.API_INVALID_PARAM` exception is thrown.

To get data about a given partner, you can use the Items or AML tabs as shown below.

Example 22–31 Getting partner data using Item or AML

```
ITable tab_item =
          prj.getTable(ProjectConstants.TABLE_ITEMS);
IRow row = (IRow) tab_item.iterator().next();
ITable partnerTable =
         (ITable) row.getValue(ProjectConstants.ATT_ITEMS_PARTNERS);

Or,

ITable tab_item =
prj.getTable(ProjectConstants.TABLE_ITEM);
IRow row =
(IRow) tab_item.iterator().next();
ITable partnerTable =
(ITable) row.getValue(ProjectConstants.ATT_ITEM_PARTNER_TABLE);
for (Iterator iterator =
partnerTable.iterator(); iterator.hasNext();) {
IRow iRow =
(IRow) iterator.next();
String partner =
iRow.getValue(ProjectConstants.ATT_PARTNERS_PARTNER).toString();
String split =
iRow.getValue(ProjectConstants.ATT_PARTNERS_PARTNER_SPLIT).toString();
}
}
```

Managing Sourcing Project Attachments for Items and AMLs

The following paragraphs provide descriptions and examples to perform the following operations:

- Loading Items and AMLs on Sourcing Projects attachments tables
- Adding Items on Sourcing Project attachment tables
- Removing Items on Sourcing Project attachment tables

Adding Sourcing Project Attachments by File Type

These examples demonstrate adding attachments by specifying the supported file types.

Example 22–32 Adding string file attachments

```
attachments.createRow("D:\\test1.txt");
```

Example 22–33 Adding MAP file attachments

```
Map map = new HashMap();
map.put(FileFolderConstants.ATT_FILES_CONTENT,"D:\\test2.txt");
map.put(FileFolderConstants.ATT_FILES_FILE_NAME, "test2.txt");
IRow row2 = attachments.createRow(map);
```

Example 22–34 Adding new attachment files

```
IRow row3 = attachments.createRow(new File("D:\\test3.txt"));
```

Example 22–35 Adding URL attachments

```
IRow row4 = attachments.createRow("http://www.oracle.com");
```

Deleting Sourcing Project Table Rows for Items

The following APIs support deleting a single or a collection of rows in Sourcing Project Attachment tables:

- Deleting a single Table row – `ITable.removeRow(IRow row)`
- Deleting all Table rows – `ITable.clear ()`

  Deleting multiple Table rows – `ITable.removeAll (ListRow row)`

Example 22–36 Removing Rows in Sourcing Project Attachment tables

```
// Remove attachment by IRow
   attachments.removeRow(row1);
```

Importing Data into Sourcing Projects

Agile exposes the following APIs to support deleting different combinations of Table rows for Items and AML tabs of a Sourcing Project:

```
String srcFilePath = ".//ImportSource//9312HF54//Assembly_BOM_AML_PC.xls";
String srcFileType = "ExcelFile";
String mappingPath = ".//ImportSource//9312HF54//Assembly_BOM_AML_PC.xml";
String transformPath = null
String[] operations = new String[] { "project.item", "project.bom","project.aml"};
List options = new ArrayList();
options.add("BusinessRuleOptions|ChangeMode=Authoring");
options.add("BusinessRuleOptions|BehaviorUponNonExistingObjects=Accept");
String _output = ".//Log//9312HF54//Ass_BOM_AML_PC_log.xml";
Options.add("Template|TemplateType=com.agile.imp.template.TemplateParentChildFilte
r");
FileOutputStream fop = new FileOutputStream(_output);
IImportManager imgr = (IImportManager)session.getManager(IImportManager.class);
byte[] logData = null;
logData = imgr.importDataForSourcingProject(projectNumber,
stream2byte(new FileInputStream(srcFilePath)), srcFileType,
convertFiletoStream(mappingPath),options);
convertFiletoStream(transformPath), operations, byte buf[] = new byte[1024 * 4];
   int n = 0;
   InputStream logStream = byte2stream(logData);
   while ((n = logStream.read(buf)) != -1) {
     fop.write(buf, 0, n);
   }
   fop.close();
System.out.println("Import Ass_BOM_AML with ParentChild Template successfully,
        the log file is "+ _output );
```

Importing Data for Sourcing Projects

The SDK exposes the `importDataForSourcingProject` API to enable this feature.

Example 22–37 Importing data for a Souring Project

```
importDataForSourcingProject (String projectNumber, byte[] srcFile, String
srcFileType, byte[] mappingFile, byte[] transformFile, String[] operations, List
options) throws APIException
```

Validating Items on the AML Tab of Sourcing Projects

The following API supports validating Items, Manufacturer sand Manufacturer parts on the AML tab of a Sourcing Project:

```
void IProject.validateItems (List itemRows, List amlRows) throws APIException,
RemoteException, Exception
```

Example 22–38 Validating Items on AML tab of a Sourcing Project

```
IProject prj = (IProject)session.getObject(IProject.OBJECT_TYPE, "PRJ000XX");
ITable prjTableAML = prj.getTable(ProjectConstants.TABLE_AML);
Iterator itr = prjTableAML.iterator();
List itemRows = new ArrayList();
List amlRows = new ArrayList();
   while (itr.hasNext()) {
    IRow row = (IRow)itr.next();
  if
("Yes".equals(row.getValue(ProjectConstants.ATT_ITEMS_ITEM_HAS_MFRS).toString()))
{
ITable tableAML = (ITable)row.getValue(ProjectConstants.ATT_ITEMS_AML);
Iterator itrAML = tableAML.getTableIterator();
    while (itrAML.hasNext()) {
      IRow amlRow = (IRow)itrAML.next();
     amlRows.add(amlRow);
    }
   }
itemRows.add(row);
}
prj.validateItems(itemRows, amlRows);
```

Working with RFQs

Requests for Quotes (RFQs) allow users to request pricing information from suppliers. RFQs serve as the instrument to negotiate pricing and terms for items or manufacturer parts. RFQs are defined for Sourcing Projects. Thus, to define an RFQ, you must first create the Sourcing Project and then create the required RFQs for that Sourcing Project.

A single Sourcing Project can generate several RFQs. RFQs support a one-to-many relationship with suppliers. That is, one RFQ may generate several responses from suppliers.

The Agile API supports the following RFQ-related tasks.

- Create an RFQ for Sourcing a Project
- Load and modify RFQ objects, tables, and attributes
- Access and modify the Page 1, Page 2, and RFQ Response tables
- Add items to the RFQ Response table from the RFQ's Sourcing Project
- Read and update the nested tables in the Page 1, Page 2, and RFQ Response table
- Assign suppliers to items or manufacturer parts in the RFQ Response table

For a list of API methods that support these RFQ functions, see "Supported API Methods" on page 22-8.

Note: The PCM SDK RFQ objects do not have a Page three, and no Page three RFQ constant is supported. Do not invoke these constants because the RFQ will not produce the expected result.

Supported API Methods

The SDK supports the following APIs for RFQs. For information on these interfaces, refer to the Javadoc generated HTML files that document the SDK code. You can find them in the HTML folder in SDK samples (ZIP file). To access this file, see the "Client-Side Components" on page 1-2.

```
- IAgileSession.createObject(Object, Object)
- IAgileSession.createObject(int, Object)
- IAgileSession.getObject(Object, Object)
- IAgileSession.getObject(int, Object)
- IRequestForQuote.getName()
- IRequestForQuote.assignSupplier(Object)
- IRequestForQuote.getTable(Object)
- IRequestForQuote.lookupPrices(Object)
- ITable.iterator()
- ITable.getTableDescriptor()
- ITable.size()
- ITable.createRow(Object)
- IRow.getValue(Object)
- IRow.setValue(Object, Object)
```

Creating RFQs for Sourcing Projects

RFQs are defined for a specific Sourcing Project. Creating an RFQ uses the generic `IAgileSession` method. You can use `IDataObject` with standard calls such as `getObject`, `getTable`, `getValue`, `setValue` to access and modify objects, tables, and attributes as follows:

- Read Page 1 or Cover Page and Page 2 tables
- Update Page 1 or Cover Page and Page 2 table

To create an RFQ, you must open the Sourcing Project. However, to open the Sourcing Project, it is necessary to first set the ship to location. See the code example in "Accessing and Modifying the Status of Sourcing Project" on page 22-12.

You cannot create an RFQ by specifying the Sourcing Project number only. You must also specify the related Sourcing Project as this is a required parameter. This is shown in the example below.

Example 22–39 Creating an RFQ for a Sourcing Project

```
IProject pnumber =
    (IProject)session.getObject(IProject.OBJECT_TYPE, "Taurus Analysis Project");
IAgileClass rfqClass =
    m_admin.getAgileClass(RequestForQuoteConstants.CLASS_RFQ);
IAutoNumber rfqNumber =
    rfqClass.getAutoNumberSources()[0];HashMap
map = new HashMap();
map.put(RequestForQuoteConstants.ATT_COVERPAGE_RFQ_NUMBER, rfqNumber);
map.put(RequestForQuoteConstants.ATT_COVERPAGE_PROJECT_NUMBER, pnumber);
IRequestForQuote rfq =
    (IRequestForQuote) m_session.createObject(rfqClass, map);
```

Loading RFQs from Sourcing Project's RFQ Table

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

Example 22–40 Loading an RFQ from the Sourcing Project RFQ table

```
ITable table = prj.getTable(ProjectConstants.TABLE_RFQS);
Iterator it = table.iterator();
IRow row1 = (IRow) it.next();
IDataObject obj1 = (IDataObject)
   m_session.getObject(IRequestForQuote.OBJECT_TYPE,row1.getValue
      (ProjectConstants.ATT_RFQS_RFQ_NUMBER));
```

Note: The `getReferent()` method does not support the PCM SDK, including the RFQ tables. A list of supported RFQ tables appears in the table below.

Supported RFQ Tables

Supported RFQ tables and their respective constants are listed in " Supported RFQ tables and respective constants".

Table 22–3 Supported RFQ tables and respective constants

Table	Constant	Read/Write Mode
Cover Page	`TABLE_COVERPAGE`	Read/Write
Page Two	`TABLE_PAGE`TWO	Read/Write
Responses	`TABLE_RESPONSES`	Read/Write

Note: The Agile API does not support adding new rows to RFQ tables. However, you can add new rows to the RFQ response table.

Accessing and Modifying RFQ Objects, Tables, Nested Tables, and Attributes

You can access RFQ objects, tables, and attributes using the generic `IAgileSession` and `IDataObject` methods and standard calls such as `getObject, getValue, setValue`. Information about these classes, tables, and attributes is provided in the `com.agile.api.RequestForQuoteConstants.java file` in the API HTML reference files. To access these files, see "Client-Side Components" on page 1-2.

RFQ Parent Table and Nested Child Table Constants

The list of parent RFQ table and the corresponding nested child table constants appears in the table below.

Table 22–4 RFQ Parent Table and Nested Child Table Constants

Parent Table Constant	Nested Child Table Constant	Read/Write Mode
`TABLE_RESPONSES`	`ATT_RESPONSES_AML`	Read/Write
`TABLE_RESPONSES`	`ATT_RESPONSES_PRICING`	Read/Write

Similar to Sourcing Projects, nested RFQ tables are accessed by treating their cell value as a table. See "Understanding Nested Tables of PCM Sourcing Projects" on page 22-11. The following example updates a nested table.

Note: Do not use `Project.ATT_RFQ_RFQ_STATE` to get the status of an RFQ, because it is not exposed to the SDK and will not render the correct value of the RFQ row. To get the status of an RFQ, you must first load the RFQ, and then get the status from the RFQ itself.

Example 22–41 Nested RFQ table update

```
ITable subtab1 =
(ITable)row.getValue(RequestForQuoteConstants.ATT_RESPONSES_PRICING);
IRow pricing1 =
(IRow)subtab1.iterator().next();
Integer nest =
ProjectConstants.ATT_PRICEDETAILS_MATERIAL_PRICE;
Object nre =
pricing1.getValue(nest);
Money tc =
new Money(new Integer(100), "USD");
pricing1.setValue(nest, (Object)tc);
```

Note: You must assign the supplier before updating an RFQ response table entry.

Performing Price Lookup in RFQs

Similar to Performing Price Lookup in a Sourcing Project, you can verify the existence of a price scenario for a specified period and quantity for RFQs. If they exist, you do not have to create an RFQ for the specified item. You can either use the price information of the item, or you can modify the price information and send the RFQ to suppliers for requote. This is shown in the following code sample.

Example 22–42 Price lookup from history and from another Sourcing Project

```
HashMap map = new HashMap();
map.put(PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE,priceTypes);
map.put(ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_CUSTOMER,customers);
map.put(ProjectConstants.ATT_GENERAL_INFORMATION_PROGRAM,programs);
map.put(ProjectConstants.
ATT_GENERAL_INFORMATION_SHIP_TO_LOCATION,shipTo);
map.put(ProjectConstants.ATT_ITEMS_NUMBER,itemMap);
map.put(ProjectConstants.ATT_ITEMS_AML,mpnMap);
map.put(LookupConstants.FLAG_ALL_PRICE_SCENARIOS,allPriceScenarios);
map.put(LookupConstants.FIELD_PRICE_SCENARIO,priceScenario);
map.put(LookupConstants.FLAG_IGNORE_QUANTITY,ignoreQtyRange);
map.put(LookupConstants.FIELD_QUANTITY_RANGE,qtyPercentRange);
map.put(LookupConstants.FLAG_IGNORE_DATE_RANGE,ignoreDateRange);
map.put(LookupConstants.FIELD_DATE_RANGE,dateRange);
map.put(LookupConstants.
FIELD_SELECT_RESPONSE_BY,LookupConstants.OPTION_LOWEST_PRICE);
map.put(LookupConstants.FLAG_EXCLUDE_AUTH_SUPPLIER,excludeAuthSupplier);
     rfq.lookupPrices(map);
```

Example 22–43 Using the required parameters to perform RFQ price lookups

```
PriceConstants.ATT_GENERAL_INFORMATION_PRICE_TYPE
ProjectConstants.ATT_ANALYSIS_SUPPLIER,suppliers
ProjectConstants.ATT_ITEMS_NUMBER or ProjectConstants.ATT_ITEMS_AML
```

```
LookupConstants.FIELD_QUANTITY_RANGE
if LookupConstants.FLAG_IGNORE_QUANTITY is 'false'
LookupConstants.FIELD_DATE_RANGE
if LookupConstants.FLAG_IGNORE_DATE_RANGE is 'false'
LookupConstants.FIELD_PRICE_SCENARIO
if LookupConstants.FLAG_ALL_PRICE_SCENARIOS is 'false'
```

Working with RFQ Responses

The PCM SDK supports the following operations for RFQ responses, nested table of items responses, and Child AML responses:

- Read RFQ tables with different views (price scenarios, currency modes)

This is supported through generic SDK API.

- Add items to RFQs
- Add Response lines (Assign Suppliers)

PCM RFQ provides the following method for assigning suppliers to items or manufacturer parts.

```
public void assignSupplier(Object supplierParams)
throws APIException, RemoteException, Exception;
```

You can assign supplier data such as Manufacturer Part Number (`mpn`) or supplier name as a String or an Object to the RFQ response shown in the following example.

Example 22–44 Adding supplier data as String constants

```
IRequestForQuote dObj =(IRequestForQuote)
   m_session.getObject(RequestForQuoteConstants.CLASS_RFQ, number);
ITable tab =
   dObj.getTable(RequestForQuoteConstants.TABLE_RESPONSES);
Map mp =
   new HashMap();
mp.put(ProjectConstants.ATT_RESPONSES_NUMBER, "P00007");
```

You can also add an item as an `IItem` object as shown below.

Example 22–45 Adding supplier data as Objects

```
mp.put(ProjectConstants.ATT_RESPONSES_NUMBER, itemObject);
```

If you are assigning a supplier to an `mpn`, you must specify the `mpn` as an **`IManufacturerPart`** Object, or as a pair of Objects. One for the `mpn` name and one for `mfr` name.

```
mp.put(RequestForQuoteConstants.ATT_RESPONSES_NUMBER, mpnObject);
```

Or,

```
mp.put(RequestForQuoteConstants.ATT_RESPONSES_NUMBER, mpnName);
mp.put(RequestForQuoteConstants.ATT_RESPONSES_MANUFACTURER, mfrName);
```

Caution: When you invokeRequestForQuote.TABLE_RESPONSE to assign suppliers to item components, the table size may change if there is more than one supplier for that item component. That is, if an item has a single supplier, each item and the corresponding supplier will occupy their own distinct separate rows in the TABLE_RESPONSE table. However, if the item has more than one supplier, then the row for this item component is split into the number of suppliers, thus changing TABLE_RESPONSE by increasing the number of rows in the table. It is therefore necessary to immediately reload the ITERATOR to reflect the change in TABLE_RESPONSE table. This is not an SDK defect and is due to SUN J2SE ITERATOR behavior.

- Update Response Lines

The PCM SDK supports the RFQ response table through generic SDK API. It does not support the RFQ Response Class or Supplier Response.

Note: The response currency in the RFQ response line is determined by the response currency attribute. This causes the server to ignore the currency parameter in the material price. Buyers can modify the response currency in the response line and it will be applied to all pricing attributes in the response line. The supplier RFQ response currency is set to your RFQ currency preference and cannot be modified in the supplier responses. Once the response line is opened to suppliers, the response line must be locked before buyers can modify them.

Locking and Unlocking RFQ Responses

The following APIs support these two functions:

- `void IRequestForQuote.lockresponses (List rows, boolean background) throws APIException, RemoteException, Exception`
- `void IRequestForQuote.unlockresponses (List rows) throws APIException, RemoteException, Exception`

Example 22–46 Locking and unlocking RFQ responses

```
ITable response = rfq.getTable(RequestForQuoteConstants.TABLE_RESPONSES);
ArrayList <IRow> respRow = new ArrayList<IRow>();
Iterator it = response.iterator();
while(it.hasNext()){
IRow row = (IRow)it.next();
respRow.add(row);
}

//lock the response rows
rfq.lockResponses(respRow,false);

//unlock the response rows
rfq.unlockResponses(respRow);
```

Editing RFQ Responses Submitted by Suppliers

The `submitAll()` API implemented on the `ISupplierResponse` API enables suppliers to submit their RFQ responses. In addition, a PCM server API enables editing the response and prices tables in the supplier's response.

These APIs support the following functions:

- Editing response tables
- Editing price tables
- Submitting the edited data in response and price tables

Example 22–47 Editing response tables

```
//Editing of Item/AML rows for Response table data
Map responseMap = new HashMap();

//Bid Decision
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_BID_DECISION, "Bid");

//Currency -read only from Supplier Response
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_CURRENCY, "FRF");

//Lead-Time
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_LEADTIME, new Integer(3));

//Inventory Available
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_INVENTORY_AVAILABLE, new
Integer(222));

//Min
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_MIN, 1);

//Mult
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_MULT, 10);

//Transportation Terms
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_TRANSPORTATION_TERMS, "EXW
- EX WORKS");

//Country of Origin
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_COUNTRY_OF_ORIGIN,
"Canada");

//EOL Date
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_EOL_DATE, eolDate);

//NCNR
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_NCNR, "Yes");

//Valid From
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_VALID_FROM, validFrom);

//Valid until - is required only for Quantity Break type project.
 // Not valid for price period project
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_VALID_UNTIL, validUntil);

//resp Money 1
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_MONEY_1, new
Money(value, "INR"));
```

```
//resp Date 1
SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy");
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_DATE_1,
df.parse("12/29/2010"));

//resp Text 1
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_TEXT_1, new
String("EMS1 Supplier Edited from SDK - Text1"));

//resp Number1
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_NUMBER_1, 11);

//resp list 1
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_LIST_1, "USD");

//resp MultiText 1
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_MULTITEXT_1, new
String("EMS1 Supplier Edited from SDK - MultiText1"));

//resp MultiList 1
responseMap.put(SupplierResponseConstants.ATT_RESPONSES_RESP_MULTILIST_1, "USD");
ISupplierResponse rfq =
(ISupplierResponse)agileSession.getObject(ISupplierResponse.OBJECT_TYPE, params);
ITable tableResp = rfq.getTable(SupplierResponseConstants.TABLE_RESPONSES);
for (Iterator rowIter = tableResp.iterator(); rowIter.hasNext(); ) {
IRow rowResp = (IRow)rowIter.next();
rowResp.setValues(responseMap);
}

//Editing Item/AML rows for Price table data
Map responsePriceMap = new HashMap();
responsePriceMap.put(ProjectConstants.ATT_PRICEDETAILS_MATERIAL_PRICE, new
Money(new Double(227.50), "USD"));
responsePriceMap.put(ProjectConstants.ATT_PRICEDETAILS_NRE, new Money(new
Double(0.85), "USD"));

// Non Material prices for assembly
responsePriceMap.put(ProjectConstants.ATT_PRICEDETAILS_NONMATERIAL_PRICE_1, new
Money(new Double(33.5), "USD"));
responsePriceMap.put(ProjectConstants.ATT_PRICEDETAILS_NONMATERIAL_PRICE_25, new
Money(new Double(1.55), "INR"));
responsePriceMap.put(ProjectConstants.ATT_PRICEDETAILS_MATERIAL_PRICE_ADDER_6, new
Money(new Double(10.50), "USD"));
responsePriceMap.put(ProjectConstants.ATT_PRICEDETAILS_MATERIAL_PRICE_ADDER_7, new
Money(new Double(222.50), "INR"));

ISupplierResponse rfq =
(ISupplierResponse)agileSession.getObject(ISupplierResponse.OBJECT_TYPE, params);
ITable tableResp =
rfq.getTable(SupplierResponseConstants.TABLE_RESPONSES);
for (Iterator rowIter = tableResp.iterator(); rowIter.hasNext(); ) {
IRow rowResp =
(IRow)rowIter.next();
ITable pricing_table =
(ITable)rowResp.getValue(SupplierResponseConstants.ATT_RESPONSES_PRICING);
Iterator priceIter = pricing_table.iterator();
while (priceIter.hasNext()) {
IRow priceRow =
(IRow)priceIter.next();
priceRow.setValues(responsePriceMap);
}
}
//New API for Submit All action from ISupplierResponse
ISupplierResponse rfqResponse =
(ISupplierResponse)agileSession.getObject(ISupplierResponse.OBJECT_TYPE, params);
rfqResponse.submitAll();
```

23 Performing Administrative Tasks

This chapter includes the following:

- About PLM Administration Tasks
- 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 Group

About PLM Administration

The Agile API provides read/write access to all nodes of Agile PLM's administrative functionality. This means you can create Agile API programs that let users read and modify Agile PLM subclasses, and add, modify, or delete Agile PLM users. The Agile API does not allow you to create new nodes in the administrative tree structure. Therefore, you can't create workflows, criteria, and roles. However, you can create users and user groups because those objects have been implemented as data objects; IUser and IUserGroup both extend IDataObject.

About Agile Java Client

Agile Java Client provides administrative functionality that enables users with Administrative privileges to manage the Agile Application Server. It enables these users to quickly and easily configure the Agile PLM system to support the way they use the system. They 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

Privileges Required to Administer Agile PLM

Before you can administer the Agile Application Server, you must have proper privileges. For access to administrative functionality, you should have the Administrator privilege. The Administrator role grants the Administrator privilege to all administrative functionality available on the server. The User Administrator role grants the Administrator privilege for functionality related to users and user groups.

Without the Administrator privilege, you cannot modify administrative nodes, users, and user groups. If you have not yet been granted Administrator rights to the Agile PLM system, contact the Agile PLM administrator.

To create users and user groups, you need the Create privilege for those objects. Several roles supplied with the Agile PLM system, such as the Administrator, User Administrator, and Change Analyst roles, include the Create privilege for users and user groups.

Administrative Interfaces

The following table lists interfaces related to Agile PLM's administrative functionality.

Table 23–1 Agile PLM's 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 an Admin node in the Agile Java Client
IProperty	A property of an Agile PLM administrative node
IRoutableDesc	Metadata that describes any object that implements the IRoutable interface, you can use IRoutableDesc to get the workflows for a class without instantiating an object of that class
ITableDesc	Metadata that describes an Agile PLM table, you can use ITableDesc to get table attributes without loading a table
ITreeNode	A generic node in a hierarchical tree structure. Several administrative interfaces, such as INode and IFolder, are subinterfaces of ITreeNode and therefore inherit its functionality. (There is a deprecated ITree interface that provides similar functionality to ITreeNode, but be sure to use ITreeNode.)
IUser	An Agile PLM user
IUserGroup	A user group. Use user groups to define project teams, site-related groups, departments, or 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 ITreeNode interface, you first get an instance of ITreeNode from the current session. The following example shows how to log in to the Agile Application Server and get an ITreeNode instance.

Example 23–1 Getting an instance of IAdmin

```
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 instance of IAdmin, you can:

- Traverse the server nodes
- Traverse the folder hierarchy
- Get Agile PLM classes and subclasses

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 Smart Rules.

This table shows the mapping of Agile Java Client nodes to Agile API administrative functionality.

Agile Java Client node	Agile API equivalent
Data Settings	
Classes	NodeConstants.NODE_AGILE_CLASSES
Character Sets	NodeConstants.NODE_CHARACTER_SETS
Lists	Not supported
Process Extensions	Not supported
AutoNumbers	NodeConstants.NODE_AUTONUMBERS
Criteria	NodeConstants.NODE_CRITERIA_LIBRARY
Workflow Settings	
Workflows	NodeConstants.NODE_AGILE_WORKFLOWS

Agile Java Client node	Agile API equivalent
User Settings	
Account Policy	Not supported
Users	Create a query of users
User Groups	Create a query of user groups
Supplier Groups	Not supported
Roles	NodeConstants.NODE_ROLES
Privileges	NodeConstants.NODE_PRIVILEGES
User Monitor	Not supported
Deleted Users	Not supported
Deleted User Groups	Not supported
System Settings	
Smart Rules	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
Project Portfolio Management	
Projects Health	Not supported
Cost Status	Not supported
Quality Status	Not supported
Resource Status	Not supported
Dashboard Management	Not supported
Default Role	Not supported
Agile Content Service	
Subscribers	NodeConstants.NODE_SUBSCRIBERS
Destinations	NodeConstants.NODE_DESTINATIONS
Events	NodeConstants.NODE_EVENTS
Filters	NodeConstants.NODE_FILTERS
Package Services	Not supported
Response Services	Not supported
Product Governance & Compliance	
Sign Off Message	Not supported
Server Settings	

Agile Java Client node	Agile API equivalent
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
Examples	
Example Roles	Not supported
Example Privileges	Not supported
Example Criteria	Not supported

Agile Web Client allows you to view and edit system and user settings by choosing Admin and Settings from the menu, respectively. The following table identifies how Agile Web Client administrative functionality maps to the Agile API.

Agile Web Client Node	Agile API equivalent
Tools > My Settings	
User Profile	User.General Info page
Change Password	IUser.changeLoginPassword() and IUser.changeApprovalPassword()
Transfer Authority	Not supported
Organize Bookmarks	My Inbox folder
Organize Searches	Searches folder
Organize Reports	Not supported
Personal Groups	My Inbox folder
Deleted Personal Groups	Not supported
Personal Criteria	Not supported
Personal Supplier Groups	Not supported
Tools > Administration > Web Client Settings	
Themes	Not supported
Tools > Administrator > User Settings	
Users	Create a query of users
User Groups	Create a query of user groups
Supplier Groups	Not supported
Deleted Users	Not supported
Deleted User Groups	Not supported
Dashboard Configuration	Not supported

Admin nodes in Agile PLM Clients do not have names that match up identically to their respective NodeConstants. For example, the Notifications node in Agile Java Client is equivalent to NodeConstants.NODE_NOTIFICATION_TEMPLATES. Similarly, the hierarchy of nodes that are represented in the Agile PLM database does not exactly match Agile Java Client node hierarchy.

If your Agile API program provides a tree view of the Agile PLM administrative nodes, you can use the view to interactively retrieve INode objects. From each INode object you can get the child nodes. If you continue to traverse the administrative node hierarchy, you can reach all node levels.

The following example shows how to retrieve the root node and its children, thus displaying the top-level nodes on the Agile Application Server.

Example 23–2 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 23–3 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 supports specifying 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 23–4 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 similar to the IAgileClass objects that are returned by the IAdmin.getAgileClasses() method. The difference is that getAgileClasses() returns several virtual classes, such as Item and Change, that are not represented as nodes. To modify the properties of the attribute of a particular node, Agile recommends using the IAdmin.getAgileClasses() or IAdmin.getAgileClass() methods. Although it's possible to modify a subclass by traversing the Classes node and its subnodes, it is much easier to work with getAgileClasses() objects. For more information, see "Managing Agile PLM Classes" on page 23-7.

Managing Agile PLM Classes

The Agile Classes node provides a framework for classifying Agile PLM objects, such as parts, changes, and packages. Using Agile Java Client, you can define new subclasses for your organization. Although you can't use the Agile API to create new subclasses, you can read or modify any of the existing subclasses. For example, you can customize a subclass by defining the attributes that are visible in each table or on each page.

The Agile PLM classes framework is based on the types of objects that are created in Agile PLM. The objects that are available on your Agile PLM system depend on the Agile PLM agreement with Oracle.

Each Agile PLM class has at least one subclass. The following table lists Agile PLM base classes, classes, and Agile-supplied subclasses. Your Agile PLM system may include other user-defined subclasses.

Table 23–2 Agile PLM Class 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

Table 23–2 (Cont.) Agile PLM Class and Subclasses

Base Class	Classes	Predefined Subclasses
	IPC 1752-2 Declarations	IPC 1752-2 Declaration
	JGPSSI Declarations	Japan Green Procurement Survey Standardization Initiative Declaration
	Part Declarations	Part Declaration
	Substance Declarations	Substance Declaration
	Supplier Declarations of Conformance	Supplier Declaration of Conformance
Discussions	Discussions	Discussion
File Folders	File Folders	File Folder
	Historical Report File Folders	Schedule Generated
		User Saved
Items	Documents	Document
	Parts	Part
Manufacturer Parts	Manufacturer Parts	Manufacturer Part
Manufacturers	Manufacturers	Manufacturer
Packages	Packages	Package
Prices	Published Prices	Contract
		Published Price
	Quote Histories	Quote History
Product Service Requests	Non-Conformance Reports	NCR
	Problem Reports	Problem Report
Projects	Activities	Phase
		Program
		Task
	Gates	Gate
Quality Change Requests	Audits	Audit
	Corrective and Preventive Actions	CAPA
Reports1	Custom Reports	Custom Report
	External Reports	External Report
	Standard Reports	Administrator Report
		Standard Report
Requests for Quote	Requests for Quote	RFQ
RFQ Responses	RFQ Responses	RFQ Response
Sites	Sites	Site
Sourcing Projects	Sourcing Projects	Sourcing Project

Table 23–2 (Cont.) Agile PLM Class and Subclasses

Base Class	Classes	Predefined Subclasses
Specifications	Specifications	Specification
Substances	Materials	Material
	Subparts	Subpart
	Substance Groups	Substance Group
	Substances	Substance
Suppliers	Suppliers	Broker
		Component Manufacturer
		Contract Manufacturer
		Distributor
		Manufacturer Representative
Transfer Orders	Automated Transfer Orders	ATO
	Content Transfer Orders	CTO
User Groups	User Groups	User Group
Users	Users	User

Note: Report objects are not supported by the Agile API.

Concrete and Abstract Classes

Agile PLM super classes, such as 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 23–5 Example: Loading an object using abstract or concrete classes

```
try {
   IItem item;
// Load a part using the Item base class
   item =(IItem)m_session.getObject
      (ItemConstants.CLASS_ITEM_BASE_CLASS, "1000-02");
// Load a part using the Parts class
   item =(IItem)m_session.getObject
      (ItemConstants.CLASS_PARTS_CLASS, "1000-02");
// Load a part using the Part subclass
   item =(IItem)m_session.getObject
      (ItemConstants.CLASS_PART, "1000-02");
} catch (APIException ex) {
    System.out.println(ex);
}
```

To get an array of classes, use the IAgileClass.getAgileClasses() method. You can specify a range of classes to return. For example, specify IAdmin.CONCRETE for the range parameter to return only concrete classes or IAdmin.ALL to return all classes.

Example 23–6 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 23–7 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 23–8 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 you created or opened in your program. For more information about ICell objects, see Chapter 6, "Working with Folders."

Referencing Attributes

You can reference Agile PLM attributes in the following ways:

- by object (an IAttribute)
- by attribute ID constants

All Agile API constants, including attribute ID constants, are contained in classes that have the suffix Constants. For example, ItemConstants contains all constants related to IItem objects.

- by fully qualified name, such as Title Block.Number or Cover Page.Change Category.

- by short name, such as Number. However, attribute short names are not unique in Agile PLM. If you are referencing multiple attributes, you may run into a conflict if two different attributes have the same short name.

Note: Because attribute names can be modified and referencing attributes by ID number or constant is difficult to identify or remember, Agile recommends using the APIName field for this purpose. For information and procedures, see Chapter 9, "Accessing PLM Metadata with APIName Field." Many of the examples in this manual reference attributes by name because they were constructed before the introduction of this field.

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

Example 23–9 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 this format TableName.AttributeName where TableName is the name of the table in which the attribute appears. AttributeName is the current value for the Name property of an attribute. All attributes have default names, but the names can be changed. In particular, Page Two and Page Three attributes that have been made visible in your Agile PLM system are likely to have been assigned more meaningful names than Text01, List01, and Date01.

For example, 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 23–10 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.

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

The following example shows how to retrieve BOM table attributes.

Example 23–11 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 23–12 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 Code" Code attribute for an ECO.

Example 23–13 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 Agile Java Client. You can also write a procedure to programmatically retrieve the ID for a user-defined attribute at run time, or use the API name.

Working with Properties of Administrative Nodes

If you use the Agile API to retrieve an INode object, you can also view property values for INode. 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 23–14 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;
   }
```

The SingleList and MultiList properties are different from other types of properties. You cannot use the IProperty.getValue() and IProperty.setValue() methods to directly modify a property that contains a list of values. Instead, you use the IAgileList.setSelection () method to select a list node, and then use the IProperty.setValue() method to set the value. For more information about how to modify SingleList and MultiList properties, see "Getting and Setting List Values" on page 5-7.

Managing Users

Users are data objects that you can create, like items and changes. Consequently, you can work with users directly without traversing the administrative node hierarchy. If you have the proper Agile PLM privileges, you can create, modify, and delete users. For example, you could create a program that periodically synchronizes Agile PLM users with data available from a corporate directory.

Getting all Users

To retrieve all Agile PLM users, run a query for User objects. The following example retrieves all users and prints the user ID, first name, and last name for each user.

Example 23–15 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 data objects that you can create with the Agile API. To create a user, you define the user's parameters and pass them to the IAgileSession.createObject() method. The required parameters you must specify are user ID 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 through the SDK that have restricted access to the Agile PLM system. You must create and maintain other users on the directory server.

Passwords that you specify for new users become 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 always change the password later.

Example 23–16 Creating a user

```
private void userTest() {
   try {

//Add code here to log in to the Agile Application Server
//After logging in, create a new user by invoking the createUser method below
   IUser user = createUser("akurosawa");
   } catch (APIException ex) {
      System.out.println(ex);
   }
}

//Create the new user
   private IUser createUser(String newUser) throws APIException {

   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 only view their own profile. To create and modify objects, the user must be assigned roles with the appropriate create and modify privileges. For an example showing how to change a user's Role settings, see "Configuring User Settings" on page 23-19.

Creating a User Group

A user group, like a user, is a dataobject and not an administrative node on the Agile Application Server. To create a user group, you define the user group's parameters, such as its name, and pass the parameters to the IAgileSession.createObject() method. The only required parameter you must specify is the name, whose attribute ID is UserGroupConstants.ATT_GENERAL_INFO_NAME. You can also specify other user attributes, which are listed in the `UserGroupConstants` class. To enable a user group, make sure the Enabled cell is set to Yes.

After creating a user group, you need to add users to the Users table to make the group meaningful. To create a new row in the Users table, use the ITable.createRow(java.lang.Object) method.

Example 23–17 Creating a user group

```
private void userGroupTest() throws APIException {
   //Add code here to log in to the Agile Application Server
   //After logging in, create a new user group
   IUserGroup group = createGroup("Swallowtail Project");

//Add users to the Western project group
   IUser[] selUsers = new IUser[] {
      m_session.getObject(IUser.OBJECT_TYPE, "jford"),
      m_session.getObject(IUser.OBJECT_TYPE, "hhawkes"),
      m_session.getObject(IUser.OBJECT_TYPE, "speckinpah")
   };
      addUsers(group, selUsers);
}

private IUserGroup createGroup(String groupName) throws APIException {
//Create the user group
   IUserGroup group =(IUserGroup)m_session.createObject
      (UserGroupConstants.CLASS_USER_GROUP, groupName);

//Enable the user group
   ICell cell = group.getCell(UserGroupConstants.ATT_GENERAL_INFO_STATUS);
   IAgileList list = cell.getAvailableValues();
   list.setSelection(new Object[] { "Active" });
   cell.setValue(list);
   return group;
}

private void addUsers(IUserGroup group, IUser[] seUsers) 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 23–18 Creating a Global User Group

```
private void setGlobal(IUserGroup group) throws APIException {
//Get the Global/Personal cell
   ICell cell = group.getCell(UserGroupConstants.
      ATT_GENERAL_INFO_GLOBAL_PERSONAL);
//Get the available values for the cell
   IAgileList values = cell.getAvailableValues();

//Set the selected value to "Global"
   values.setSelection(new Object[] { "Global" });

//Change the cell value
   group.setValue(UserGroupConstants.
      ATT_GENERAL_INFO_GLOBAL_PERSONAL, values);
}
```

Creating Users and Requiring Password Modification at Login

When creating a user, you can require the new user to change the assigned password, which is usually temporary, for a new and more secure one at login. To create such a user, it is necessary to define the user's parameters as explained in "Creating a User" on page 23-16. and pass a flag to force password change at login. This is illustrated in the following code sample.

Example 23–19 Creating a user and requiring password modification at login

```
String username = "USER" + System.currentTimeMillis();
HashMap params = new HashMap();
params.put(UserConstants.ATT_GENERAL_INFO_USER_ID, username);
params.put(UserConstants.ATT_GENERAL_INFO_FIRST_NAME, username);
params.put(UserConstants.ATT_GENERAL_INFO_LAST_NAME, username);
params.put(UserConstants.ATT_LOGIN_PASSWORD, "agile");
params.put(UserConstants.ATT_APPROVAL_PASSWORD, "agile2");
params.put(UserConstants.ATT_MUST_CHANGE_PWD_AT_LOGON, "true");
IUser user = (IUser)  session.createObject(UserConstants.CLASS_USER, params);
System.out.println("Created user: " + user.getName());
```

Creating a Supplier User

Supplier users are assigned to the Restricted user category by default, which restricts their access to the Agile PLM system. The Restricted user category allows supplier users to respond to RFQs and use other features of Agile Product Cost Management (PCM).

To create a supplier user, define the user's parameters and pass them to the IAgileSession.createObject() method. You must specify the user ID, login password, and supplier name. You can also specify other user attributes, which are listed in the UserConstants class.

Example 23–20 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 as a New User

You can use the IDataObject.saveAs() method to save an existing user to a new user. The saveAs()method serves as a handy shortcut because it allows you to assign a new user the same roles, privileges, and sites as an existing user. When you use the saveAs() method to save a user, you must specify parameters for the new user's user name and login password.

Example 23–21 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);
   }
}
```

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 23–22 Getting user cells for General Info and Page Two

```
private void getUserCells(IUser user) throws APIException {
   ICell[] cells = user.getCells();
      for (int i = 0; i < cells.length; i++) {
         System.out.println(cells[i].getName() + " : " + cells[i].getValue());
      }
}
```

Two important settings for a user are User Category and Roles. The User Category setting defines the broad range of actions a user can perform on the Agile PLM system. Select from one of the following User Category values:

- **Power** - User can log in to the server at any time with unrestricted use of the Agile PLM system. Power users are not subject to the limited number of concurrent users.
- **Concurrent** - User can log in to the server only if a concurrent user agreement is available.

> **Note:** Licenses are controlled by your agreement with Oracle.

- **Restricted** - User has restricted access to the Agile PLM system. Supplier users are by default assigned the Restricted category, which allows them to respond to RFQs and use other features of Agile Product Cost Management (PCM). Restricted users are not subject to the limited number of concurrent users.

The Roles setting further defines the capabilities of a user, assigning roles and privileges. A user won't be able to create objects without the proper roles and privileges. For more information about Agile PLM user roles, and privileges, refer to the *Agile PLM Administrator Guide*.

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

Example 23–23 Setting the User Category and Roles settings for a user

```
private void setCategory(IUser user) throws APIException {

//Get the User Category cell
   ICell cell = user.getCell(UserConstants.ATT_GENERAL_INFO_USER_CATEGORY);

//Get the available values for the cell
   IAgileList license = cell.getAvailableValues();

//Set the selected value to "Concurrent"
   license.setSelection(new Object[] { "Concurrent" });

//Change the cell value
   cell.setValue(license);
}
private void setRoles(IUser user) throws APIException {

//Get the Role cell
   ICell cell = user.getCell(UserConstants.ATT_GENERAL_INFO_ROLES);

//Get the available values for the cell
   IAgileList roles = cell.getAvailableValues();

//Set the selected roles to Change Analyst and Administrator
   roles.setSelection (new Object[]
      {"Change Analyst","Administrator","My User Profile"});

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

Resetting User Passwords

Administrators with User Administrator privileges can reset the password of other users to a new value. Users without this privilege are not able to reset user passwords. This feature enables resetting large numbers of passwords in the batch mode, which is preferable to manually changing them one at a time using the UI.

The changeLoginPassword() method which supports this feature allows passing a null value instead of the current password value. The following example shows how to use this method to reset a user's password using null instead of the current password.

Example 23–24 Resetting a password to a new value

```
public void changeLoginPassword(null, String newPassword)
      throws APIException;
```

Deleting a User

To delete a user, use the IDataObject.delete() method. Like other data objects, 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 23–25 Deleting a user

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

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

Managing User Groups

A user group is an object that contains a list of Agile PLM users. You can use user groups to define project teams, departments, and global groups and their assigned users. User groups are not site-related, unlike items and changes, but you can create groups of users based on their location. Whenever you add a user to a user group, that change is reflected in the user's Groups setting, whose attribute ID is UserConstants.ATT_GENERAL_INFO_GROUPS.

Note: In Agile Clients such as Agile Web Client, you can send an object, such as a change, to a user group. The Agile API does not support sending objects to user groups. However, you can retrieve users from the Users table of a User Group object and then send them an object.

Getting All User 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 23–26 Example: Getting all user groups

```
private void getAllUserGroups() throws APIException {
   IQuery q = (IQuery)m_session.createObject
      (IQuery.OBJECT_TYPE, "select * from [User Groups]");
   ArrayList groups = new ArrayList();
   Iterator itr = q.execute().getReferentIterator();
   while (itr.hasNext()) {
      groups.add(itr.next());
   }
   for (int i = 0; i < groups.size(); i++) {
    IUserGroup ug =(IUserGroup)groups.get(i);
    System.out.println(
    ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_NAME) + ", " +
    ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_DESCRIPTION) + ", " +
    ug.getValue(UserGroupConstants.ATT_GENERAL_INFO_MAX_NUM_OF_NAMED_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.

After creating a user group, you need to add users to the Users table to make the group meaningful. To create a new row in the Users table, use the ITable.createRow(java.lang.Object) method.

Example 23–27 Creating a user group

```
private void userGroupTest() throws APIException {
//Add code here to log in to the Agile Application Server
//After logging in, create a new user group
   IUserGroup group =
   createGroup("Swallowtail Project");
//Add users to the Western project group
   IUser[] selUsers = new IUser[] {
      m_session.getObject(IUser.OBJECT_TYPE, "jford"),
      m_session.getObject(IUser.OBJECT_TYPE, "hhawkes"),
      m_session.getObject(IUser.OBJECT_TYPE, "speckinpah")
   };
   addUsers(group, selUsers);
}

private IUserGroup createGroup(String groupName) throws APIException {
//Create the user group
   IUserGroup group =(IUserGroup)m_session.
      createObject(UserGroupConstants.CLASS_USER_GROUP, groupName);

//Enable the user group
   ICell cell = group.getCell(UserGroupConstants.ATT_GENERAL_INFO_STATUS);
   IAgileList list = cell.getAvailableValues();
   list.setSelection(new Object[] { "Active" });
      cell.setValue(list);
      return group;
}

private void addUsers(IUserGroup group, IUser[] seUsers) 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 23–28 Making a user group global

```
private void setGlobal(IUserGroup group) throws APIException {

//Get the Global/Personal cell
   ICell cell = group.getCell
```

```
      (UserGroupConstants.ATT_GENERAL_INFO_GLOBAL_PERSONAL);

//Get the available values for the cell
   IAgileList values = cell.getAvailableValues();

//Set the selected value to "Global"
   values.setSelection(new Object[] { "Global" });

//Change the cell value
   group.setValue(UserGroupConstants.ATT_GENERAL_INFO_GLOBAL_PERSONAL, values);
}
```

Adding a User Group to the User Group Table of the User

IUserGroup cannot be passed to createRow() to add a user group to the user's user group table. You must use a Map as shown in the following example.

Example 23–29 Using a Map to add a User Group to a User Group Table

```
ITable ugTable = user.getTable(UserConstants.TABLE_USERGROUP);

Map map = new HashMap();
map.put(UserConstants.ATT_USER_GROUP_GROUP_NAME, ug.getName());
ugTable.createRow(map);
```

Listing Users in a User Group

The users contained within a user group are listed on the Users table. Therefore, to get the list of users in the user group, use the IDataObject.getTable() method and then iterate over the table rows to access data for each user. The following example shows how to list the users in a user group.

Example 23–30 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));
   }
}
```

A

Mapping PLM Client Features to Agile API

This appendix includes the following:

- Login Features
- General Features
- Search Features
- Workflow Features
- Manufacturing Site Features
- Folder Features
- Project Features
- Administrative Features

Login Features

The following table lists general features for logging in to the Agile Application Server.

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

General Features

The following table lists the General features that apply to all Agile PLM business objects.

Feature	Equivalent Method(s)
Create a new object	IAgileSession.createObject()
Load an existing object	IAgileSession.getObject()
Save an object as another object	IDataObject.saveAs()
Delete an object	IDataObject.delete() IFolder.delete() IQuery.delete()

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

Search Features

The table below lists the supported Search (Query) features.

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

Attachment Features

The table below lists features for working with attachments and file folders.

Feature	Equivalent Method(s)
Download all files contained in a file folder	IFileFolder.getFile()
Download a single file listed on the Attachments tab	IAttachmentFile.getFile()
Check out a file folder	IFileFolder.checkOut()
Check in a file folder	IFileFolder.checkIn()
Cancel checkout	IFileFolder.cancelCheckOut()

Feature	Equivalent Method(s)
Incorporate or unincorporate an item, thereby locking or unlocking its attachments	IAttachmentContainer.setIncorporated()

Workflow Features

The table below lists Workflow features for routable Agile PLM objects.

Feature	Equivalent Method(s)
Audit a routable object	IRoutable.audit()
Change the status of a routable object	IRoutable.changeStatus()
Send an object to another Agile PLM user(s)	IDataObject.send()
Approve a routable object	IRoutable.approve()
Reject a routable object	IRoutable.reject()
Comment on a routable object	IRoutable.comment()

Add or remove approvers and observers for a routable object	IRoutable.addApprovers()
	IRoutable.removeApprovers()

Manufacturing Site Features

The table below lists features for working with manufacturing sites.

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

Folder Features

The following table lists the Folder features for working with folders.

Feature	Equivalent Method(s)
Add an item (such as a query) to the folder	IFolder.addChild()
Set the type of folder (public or private)	IFolder.setFolderType()
Set the folder name	IFolder.setName()
Get a folder of the current user	IUser.getFolder()
Remove an item from the folder	IFolder.removeChild()

Feature	Equivalent Method(s)
Clear all objects from the folder	IFolder.clear()
Delete a folder	IFolder.delete()

Project Features

The following table lists features for working with Projects.

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

Administrative Features

The following table provides the list of features for working with Administration nodes and properties in Agile Java Client.

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

Feature	Equivalent Method(s)
Get an Agile PLM list	IListLibrary.getAdminList()
Get all Agile PLM users	Create a query of users
Get all Agile PLM user groups	Create a query of user groups
Create a user or user group	IAgileSession.createObject()
Set properties of a user or user group	IProperty.setValue()

Change user passwords	IUser.changeApprovalPassword()
	IUser.changeLoginPassword()

B Migrating Table Constants to Release 9.2.2

This appendix includes the following:

- Mapped Pre-Release 9.2.2 Table Constants to 9.2.2 Table Constants
- Removed Pre-Release 9.2.2 Table Constants

Overview

This appendix describes how to migrate Table Contents from pre-Release 9.2.2 (Release 9.2.1 and earlier releases) to post-Release 9.2.2 (Release 9.2.2 or later releases). Information about merging and replacing the Relationship tables first appeared in "Accessing the New and Merged Relationships Tables" on page 4-3. Tables in this appendix list the Release 9.2.2 table constants and table constants that were either merged and mapped into a single table constant, or mapped into a new table constant.

Mapped Pre-Release 9.2.2 Table Constants to 9.2.2 Table Constants

This table lists the pre-release 9.2.2 table constants and the new table constants that are either merged and mapped into Release 9.2.2, or mapped into later releases of the SDK.

Pre 9.2.2 Table Constants	9.2.2 Table Constants
TABLE_RELATIONSHIPSAFFECTEDBY TABLE_RELATIONSHIPSAFFECTS TABLE_REFERENCES	TABLE_RELATIONSHIPS
ATT_RELATIONSHIPS_AFFECTED_BY_CRITERIA_MET ATT_RELATIONSHIPS_AFFECTS_CRITERIA_MET	ATT_RELATIONSHIPS_CRITERIA_MET
ATT_RELATIONSHIPS_AFFECTED_BY_CURRENT_STATUS ATT_RELATIONSHIPS_AFFECTS_CURRENT_STATUS	ATT_RELATIONSHIPS_CURRENT_STATUS
ATT_RELATIONSHIPS_AFFECTED_BY_DATE01 ATT_RELATIONSHIPS_AFFECTS_DATE01	ATT_RELATIONSHIPS_DATE01
ATT_RELATIONSHIPS_AFFECTED_BY_DATE02 ATT_RELATIONSHIPS_AFFECTS_DATE02	ATT_RELATIONSHIPS_DATE02
ATT_RELATIONSHIPS_AFFECTED_BY_DATE03 ATT_RELATIONSHIPS_AFFECTS_DATE03	ATT_RELATIONSHIPS_DATE03

Pre 9.2.2 Table Constants	9.2.2 Table Constants
ATT_RELATIONSHIPS_AFFECTED_BY_DATE04 ATT_RELATIONSHIPS_AFFECTS_DATE04	ATT_RELATIONSHIPS_DATE04
ATT_RELATIONSHIPS_AFFECTED_BY_DATE05 ATT_RELATIONSHIPS_AFFECTS_DATE05	ATT_RELATIONSHIPS_DATE05
ATT_REFERENCES_DATE01	ATT_RELATIONSHIPS_DATE06
ATT_REFERENCES_DATE02	ATT_RELATIONSHIPS_DATE07
ATT_REFERENCES_DATE03	ATT_RELATIONSHIPS_DATE08
ATT_REFERENCES_DATE04	ATT_RELATIONSHIPS_DATE09
ATT_REFERENCES_DATE05	ATT_RELATIONSHIPS_DATE10
ATT_RELATIONSHIPS_AFFECTED_BY_ DESCRIPTION ATT_RELATIONSHIPS_AFFECTS_DESCRIPTION ATT_REFERENCES_DESCRIPTION	ATT_RELATIONSHIPS_ DESCRIPTION
ATT_RELATIONSHIPS_AFFECTED_BY_LIST01, ATT_RELATIONSHIPS_AFFECTS_LIST01	ATT_RELATIONSHIPS_LIST01
ATT_RELATIONSHIPS_AFFECTED_BY_LIST02, ATT_RELATIONSHIPS_AFFECTS_LIST02	ATT_RELATIONSHIPS_LIST02
ATT_RELATIONSHIPS_AFFECTED_BY_LIST03 ATT_RELATIONSHIPS_AFFECTS_LIST03	ATT_RELATIONSHIPS_LIST03
ATT_RELATIONSHIPS_AFFECTED_BY_LIST04 ATT_RELATIONSHIPS_AFFECTS_LIST04	ATT_RELATIONSHIPS_LIST04
ATT_RELATIONSHIPS_AFFECTED_BY_LIST05 ATT_RELATIONSHIPS_AFFECTS_LIST05	ATT_RELATIONSHIPS_LIST05
ATT_REFERENCES_LIST01	ATT_RELATIONSHIPS_LIST06
ATT_REFERENCES_LIST02	ATT_RELATIONSHIPS_LIST07
ATT_REFERENCES_LIST03	ATT_RELATIONSHIPS_LIST08
ATT_REFERENCES_LIST04	ATT_RELATIONSHIPS_LIST09
ATT_REFERENCES_LIST05	ATT_RELATIONSHIPS_LIST10
ATT_RELATIONSHIPS_AFFECTED_BY_ MULTITEXT01 ATT_RELATIONSHIPS_AFFECTS_MULTITEXT01	ATT_RELATIONSHIPS_ MULTITEXT01
ATT_RELATIONSHIPS_AFFECTED_BY_ MULTITEXT02 ATT_RELATIONSHIPS_AFFECTS_MULTITEXT02	ATT_RELATIONSHIPS_ MULTITEXT02
ATT_RELATIONSHIPS_AFFECTED_BY_ MULTITEXT03 ATT_RELATIONSHIPS_AFFECTS_MULTITEXT03	ATT_RELATIONSHIPS_ MULTITEXT03
ATT_RELATIONSHIPS_AFFECTED_BY_ MULTITEXT04 ATT_RELATIONSHIPS_AFFECTS_MULTITEXT04	ATT_RELATIONSHIPS_ MULTITEXT04

Pre 9.2.2 Table Constants	9.2.2 Table Constants
ATT_RELATIONSHIPS_AFFECTED_BY_ MULTITEXT05	ATT_RELATIONSHIPS_ MULTITEXT05
ATT_RELATIONSHIPS_AFFECTS_MULTITEXT05	
ATT_REFERENCES_MULTITEXT01	ATT_RELATIONSHIPS_ MULTITEXT06
ATT_REFERENCES_MULTITEXT02	ATT_RELATIONSHIPS_ MULTITEXT07
ATT_REFERENCES_MULTITEXT03	ATT_RELATIONSHIPS_ MULTITEXT08
ATT_REFERENCES_MULTITEXT04	ATT_RELATIONSHIPS_ MULTITEXT09
ATT_REFERENCES_MULTITEXT05	ATT_RELATIONSHIPS_ MULTITEXT10
ATT_RELATIONSHIPS_AFFECTED_BY_TEXT01	ATT_RELATIONSHIPS_TEXT01
ATT_RELATIONSHIPS_AFFECTS_TEXT01	
ATT_RELATIONSHIPS_AFFECTED_BY_TEXT02	ATT_RELATIONSHIPS_TEXT02
ATT_RELATIONSHIPS_AFFECTS_TEXT02	
ATT_RELATIONSHIPS_AFFECTED_BY_TEXT03, ATT_RELATIONSHIPS_AFFECTS_TEXT03	ATT_RELATIONSHIPS_TEXT03
ATT_RELATIONSHIPS_AFFECTED_BY_TEXT04, ATT_RELATIONSHIPS_AFFECTS_TEXT04	ATT_RELATIONSHIPS_TEXT04
ATT_RELATIONSHIPS_AFFECTED_BY_TEXT05	ATT_RELATIONSHIPS_TEXT05
ATT_RELATIONSHIPS_AFFECTS_TEXT05	
ATT_REFERENCES_TEXT01	ATT_RELATIONSHIPS_TEXT06
ATT_REFERENCES_TEXT02	ATT_RELATIONSHIPS_TEXT07
ATT_REFERENCES_TEXT03	ATT_RELATIONSHIPS_TEXT08
ATT_REFERENCES_TEXT04	ATT_RELATIONSHIPS_TEXT09
ATT_REFERENCES_TEXT05	ATT_RELATIONSHIPS_TEXT10
ATT_RELATIONSHIPS_AFFECTED_BY_NOTES	ATT_RELATIONSHIPS_NOTES1
ATT_RELATIONSHIPS_AFFECTS_NOTES	
ATT_REFERENCES_NOTES	ATT_RELATIONSHIPS_NOTES2
ATT_RELATIONSHIPS_AFFECTED_BY_NUMBER	ATT_RELATIONSHIPS_NAME
ATT_RELATIONSHIPS_AFFECTS_NUMBER	
ATT_REFERENCES_NUMBER	

Removed Pre-Release 9.2.2 Table Constants

The following pre-release 9.2.2 table constants are no longer available and should not be used in later releases of the SDK:

- ATT_RELATIONSHIPS_AFFECTED_BY_EVENT
- AT_TRELATIONSHIPS_AFFECTS_TRIGGER_EVENT
- ATT_RELATIONSHIPS_AFFECTED_BY_TRIGGER_EVENT
- ATT_RELATIONSHIPS_AFFECTS_EVENT

- ATT_RELATIONSHIPS_AFFECTS_RESULT
- MaterialDeclarationConstants.TABLE_RELATIONSHIPSAFFECTEDBY
- MaterialDeclarationConstants.TABLE_RELATIONSHIPSAFFECTS
- MaterialDeclarationConstants.TABLE_REFERENCES