# Agile Product Lifecycle Management

SDK Developer Guide - Developing PLM Extensions

Release 9.3.5

**E61165-02**

January 2016

ORACLE®

Agile Product Lifecycle Management ACS User Guide, Release 9.3.5

E61165-02 ,Oracle and/or its affiliates. All rights reserved.



Primary Author: Oracle Corporation

Contributing Author: F. Tabibzade

Contributor:

# Contents

## 3 Developing Process Extensions

# 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 `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 website, 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 - Developing PLM Extensions
- SDK Components
- SDK Architecture
- System Requirements
- Java Requirements
- Agile SDK Installation Folders
- Checking Your Agile PLM System
- Agile PLM Business Objects

## About this Guide - Developing PLM Extensions

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 *SDK Developer Guide - Using Agile APIs*.

- **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 this book.

### PLM Extensions

This component of the SDK Developer Guide documents the following extension frameworks that are developed with the Java APIs, and tools such as Java or Groovy scripts:

- **Process extensions (Custom PXs)** - This framework enables Agile PLM customers to extend the functionality of Agile PLM. The functionality can be server-side extensions, such as custom workflow actions and custom autonumbering, or extensions to client-side functionalities, such as external reports, or new commands added to the Actions menu or the Tools menu. These PXs are implemented using the Java programming language.
- **Event Framework** - This framework supports Java process extensions (Java PXs), and Script process extensions (Script PXs). Similar to Custom PXs, they help PLM customers to extend the functionality of PLM clients to manage events by extending the function of an action taken by a user, an interface, or the system when the Event is triggered. Java PXs are implemented using Java and Script PXs are implemented using a scripting language called Groovy. Groovy is an object-oriented programming language for the Java Platform as an alternative to the Java programming language.
- **Web service extensions (WSX)** - This is a framework that allows Agile PLM customers to extend the functionality of the PLM system and expose the customer-specific solutions as a Web service.
- **Dashboard Management extensions (DX)** - DXs extend the functionality of the Agile PLM system. They provide the data, Dashboard Tabs, and the required formats to display the data (tables, charts, and URLs) that are configured in Agile Java Client and in Agile Web Client for authorized users.

## 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:

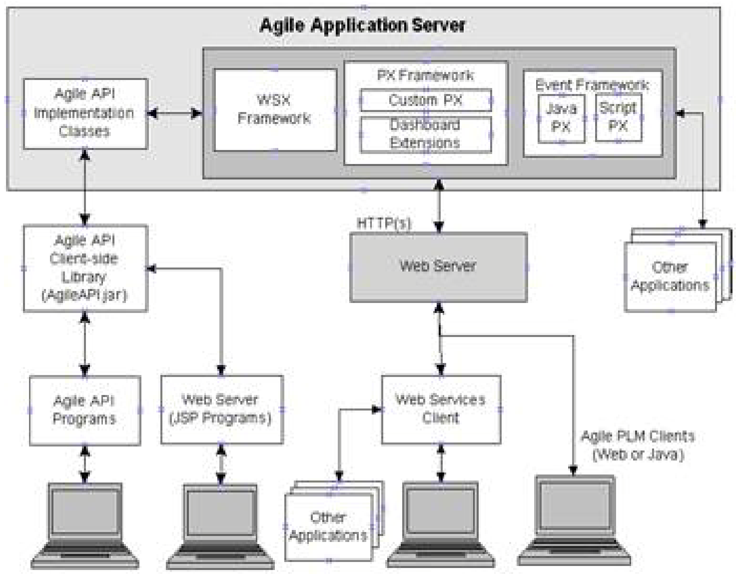
- Agile API implementation classes
- Java and Scripting process extensions framework
- Web service extensions frameworks

## 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.




**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 application server is using. Oracle Application Server 10g must use Sun Java Runtime Environment (JRE) 1.5.0_06 and Oracle WebLogic Server 10.3 must use Sun Java Runtime Environment (JRE) 1.6 for interoperability and 2007 Daylight Saving Time compliance.

**Important:** SDK code running under JRE 7 cannot connect to a Proxy URL protected by Oracle Application Server Single Sign-On (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 JRE Requirements for Agile API Clients***

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

## 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 as shown below:
  `java -Ddisable.agile.sessionID.generation=true pk.sample`
- If the Client is a PX and out of memory occurs in Agile Server, add the JVM option in: `<OAWorkaroundS_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:

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

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

## Checking Your Agile PLM System

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

## Agile PLM Business Objects

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

| **Object** | **Related Agile API Interface** |
|---|---|
| Changes | `IChange` |
| Customers | `ICustomer` |
| Declarations | `IDeclaration` |
| Design | `IDesign` |
| Discussions | `IDiscussion` |
| File Folders | `IFileFolder` |
| Items | `IItem` |
| Manufacturer parts | `IManufacturerPart` |
| Manufacturers | `IManufacturer` |
| Packages | `IPackage` |
| Part Groups (Commodity or Part Family) | `ICommodity` |
| Prices | `IPrice` |
| Product Service Request | `IServiceRequest` |
| Projects | `IProgram` |
| Sourcing Project | `IProject` |
| Quality Change Request | `IQualityChangeRequest` |

| Object | Related Agile API Interface |
|---|---|
| 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 through the Agile API.

# 2

# Developing Web Service Extensions

This chapter includes the following:

- About Web Services Extensions
- Web Services Technology
- Developing and Deploying a Web Service
- Using a Web Service
- Authenticating Users
- Preparing the Environment to Implement a WebService Sample
- About Web Service Clients Creating a
- Creating a Web Service Client
- Microsoft.NET Interoperability
- Web Service Extensions FAQs

## About Web Service Extensions

Web service extensions (WSX) is a Web service engine enabling communication between Agile PLM and disparate systems both internal and external including Enterprise Resource Planning (ERP) systems, Customer Resource Management (CRM) systems, Business-to-Business Integration systems (B2Bi), other Agile PLM systems, and supply chain partners. WSX can streamline the process for new product introduction (NPI), product changes, and rapid ramp-up of manufacturing resources. It can also simplify the process for aggregating raw product content and making critical product content available in real time to other core systems. WSX contains the tools and framework to develop new Agile PLM Web services.

You can use WSX to:

- Make product content available to Enterprise Application Integration (EAI) systems, which can then feed the data to a broad array of internal applications.
- Share product content with product design, manufacturing planning, shop floor, Enterprise Resource Planning (ERP), and Customer Relationship Management (CRM) applications.
- Make product content available to Business-to-Business (B2B) systems, which can transfer Agile Application server data across corporate boundaries to a wide range of external applications.
- Provide content to exchanges, reports, and custom applications and import Product content data from ERP and other supply chain applications.

**Note:** Agile Integration Services (AIS) is a set of Web services that is built with WSX technology to provide programmatic import and export capabilities for the Agile PLM system. AIS is a separately licensed product. For more information about AIS, refer to the *Agile Integration Services Developer Guide*.

## Key Features

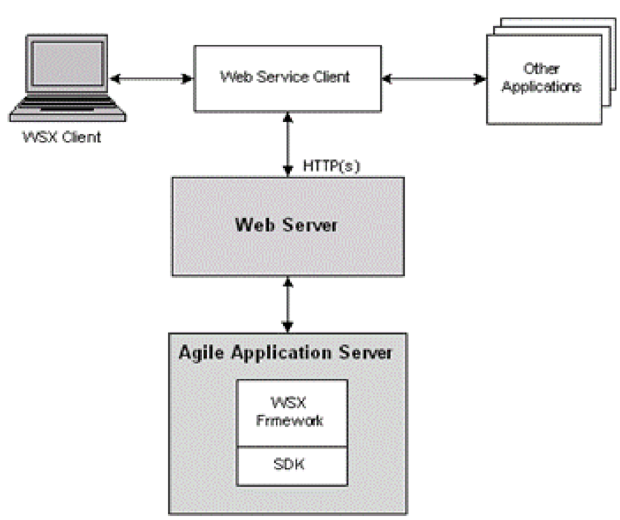
WSX includes the following key features:

- **Programmatic access to data** - WSX provides programmatic access to data stored in Agile PLM systems and other data resources, allowing you to create custom applications to automate content transfer.
- **Accessibility** - WSX provides accessibility of Agile PLM product content outside the corporate firewall using standard HTTP(S) technology.
- **Multiple programming language support** - WSX supports any language that can create and understand both Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL).
- **Multiple output format support** - WSX supports aXML and PDX 1.0. You can also use XSL to transform XML data into any format, or develop Web services that return data in any format.
- **Security** - WSX communicates with XML-compliant applications using Internet-standard communication and security protocols (HTTP and SSL), so the interface is both firewall-friendly and secure.

## WSX Architecture

To connect to Agile PLM and the WSX framework, you use standard Web service invocation methodologies.

*Figure 2–1 WSX architecture*



# Web Services Technology

Web services is a technology for building distributed applications. These services, which can be made available over the Internet, use a standardized XML messaging system and are not tied to any one operating system or programming language. Through Web services, companies can encapsulate existing business processes, publish them as services, search for and subscribe to other services, and exchange information throughout and beyond the enterprise. Web services are based on universally agreed upon specifications for structured data exchange, messaging, discovery of services, interface description, and business process design.

A Web service makes remote procedure calls across the Internet. It uses HTTP(S) or other protocols to transport requests and responses and the Simple Object Access Protocol (SOAP) to communicate request and response information.

The key benefits provided by Web services are:

- **Service-oriented Architecture** - Unlike packaged products, Web services can be delivered as streams of services that allow access from any platform. Components can be isolated; only the business-level services need be exposed.
- **Interoperability** - Web services ensure complete interoperability between systems.
- **Integration** - Web services facilitate flexible integration solutions, particularly if you are connecting applications on different platforms or written in different languages.
- **Modularity** - Web services offer a modular approach to programming. Each business function in an application can be exposed as a separate Web service. Smaller modules reduce errors and result in more reusable components.
- **Accessibility** - Business services can be completely decentralized. They can be distributed over the Internet and accessed by a wide variety of communications devices.
- **Efficiency** - Web services constructed from applications meant for internal use can be used externally without changing code. Incremental development using Web services is relatively simple because Web services are declared and implemented in a human readable format.

Like any technology, Web services have some limitations. When developing Web services, you should consider the following:

- SOAP is a simple mechanism for handling data and requests over a transport medium. It is not designed to handle advanced operations such as distributed garbage collection, object activation, or call by reference.
- Because Web services are network-based, they are affected by network traffic. The latency for any Web service invocation can often be measured in hundreds of milliseconds. Thus, the amount of functionality provided by the service should be significant enough to warrant making a high-latency call.
- Web services are not good at conversational programming. Thus, when designing services to be exposed, you should try to make the service as independent as possible.

## Web Services Architecture

You can view Web services architecture in terms of roles and the protocol stack:

- Web service roles:
  - **Service provider** - This provides the service by implementing it and making it available on the Internet.
  - **Service requestor** - This is the user of the service who accesses the service by opening a network connection and sending an XML request.
  - **Service registry** -This is a centralized directory of services where developers can publish new services or find existing ones.
- Web services protocol stack:
  - **Service transport layer** - uses HTTP to transport messages between applications. Other transports will be supported in future AIS releases.
  - **XML messaging layer** - encodes messages in XML format by using SOAP, a platform-independent XML protocol used for exchanging information between computers. It defines an envelope specification for encapsulated data being transferred, the data encoding rules, and Remote Procedure Call (RPC) conventions.
  - **Service description layer** -describes the public interface to a specific Web service by using the Web Service Description Language (WSDL) protocol. WSDL defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages, which contain either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a network protocol and message format. WSDL enables describing the endpoints and their messages independent of the message formats or network protocols. A WSDL document defines services as collections of network endpoints (called ports). A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.
  - **Service discovery layer** - centralizes services into a common registry by using the Universal Description, Discovery, and Integration (UDDI) protocol.

**Note:** WSX does not currently support UDDI or other service discovery layers.

## Security

WSX communicates with XML-compliant applications using Internet-standard communication and security protocols (HTTP and SSL). Communication between WSX and its clients (through the Web server) may be encrypted through Secure Sockets Layer (SSL) and a server-side certificate, thus providing authentication, privacy, and message integrity. Using standard Java cryptography libraries, you can encrypt and decrypt files, create security keys, digitally sign a file, and verify a digital signature.

The Web service extensions framework forces any invocation request received from outside the firewall to be secure. In other words, all external requests to WSX must be secured using HTTPS or an equivalent protocol. Internal requests to WSX can be conducted insecurely, that is, using HTTP.

There are several ways to enforce username and password security when invoking a Web service. If you are using the Agile API to develop your Web service, you can specify the username and password in the `createSession()` parameters just as you would with any API program.

For more information about Java security and cryptography support, see http://docs.oracle.com/javase/1.3/docs/guide/security/index.html.

## Tools

There is no single set of tools needed to access Web services. The tools you choose depend very much on the environment you use to develop clients. Basically, you'll need tools that enable you to generate and process XML, and process HTTP request/responses messages.

The WSX framework is based on the Apache eXtensible Interaction System (AXIS), which is a SOAP processor. However, you can use other implementations of SOAP tools, regardless of source language, to build Web service clients.

---

**Note:** The WSX Java samples included with the Agile SDK show how to use AXIS.

---

## Finding Additional Information About Web Services

This is a list of some websites to explore:

- **WebServices.Org** - http://www.webservices.org/
- **Web Services Architect** - http://www.webservicesarchitect.com/
- **Web Services Journal** - http://www.informatik.uni-trier.de/~ley/db/journals/jwsr/jwsr8.html
- **webservices.xml.com** - http://www.xml.com/pub/a/ws/2001/04/04/webservices/index.html
- **Apache Axis** - http://ws.apache.org/axis/
- **Java Web Services Developer Pack** - http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-jwsdp-419428.html
- **Sun ONE Web Services Platform Developer Edition** - http://dsc.sun.com/appserver/reference/techart/webservice.html
- **Microsoft .Net Framework** - http://msdn.microsoft.com/netframework/
- **SOAP::Lite for Perl** - http://www.soaplite.com/
- **Soap Tutorial** - http://www.w3schools.com/soap/default.asp

# Developing and Deploying a Web Service

Writing your own Web service is a simple task, consisting of a few steps:

1. Define your Web service's entry point(s). A Web service entry point (or operation) corresponds to a public method in a Java class.
2. Code your Web service operation's logic. You need not follow any special rules when coding the logic for your Web service operation. You can also use third party code libraries in addition to the Agile-provided libraries, including the Agile API.
3. Compile your Java code as you normally would.
4. Copy the compiled JAR file(s) to `AGILE_HOME\integration\sdk\extensions` on the Agile Application Server computer. The deployment descriptor for the Web service should also be in the JAR file(s) in a file named `META-INF/services/com.agile.wsx.Deployment.wsdd`.

> **Note:** If you have several application servers in a clustered environment, you must deploy Web service files on each server in the cluster.

The Agile Application Server automatically deploys all Web services listed in the deployment descriptor, ensuring that your latest changes have been applied.

## Reserved Web Service Names

The following Web service names are reserved for use by the Agile Integration Services (AIS). Do not use them to name a Web service that you've created.

- Export
- Importer
- Reserved Service names:
  - `FSHelper, DmsService` (File manager and Viewer)
  - `Export, Importer` (AIS)
  - `ResponseService, PackageService, AcsStatusService` (ACS)

# Using a Web Service

Once you have developed and deployed your custom Web service, you will want to use it. You can access your Web service using a URL of the form `http://hostname:port#/virtualPath/integration/ws/WebServiceName`

> **Note:** You must use the Agile-modified axis.jar file that is included with the Agile API.This file gets installed in the following location when you install Agile's API component: `agile_Home\integration\sdk\lib\axis.jar`

## Defining a Web Service Entry Point

A Web service entry point (or operation) corresponds to a public method in a Java class. Not all public methods in a class need be exposed as an operation, but all operations correspond to public methods. Thus, if you have a Java class (such as `MyClass`), that exposes two public methods (such as `methodOne` and `methodTwo`), it is possible for you to expose either or both methods as Web service operations.

As a general rule, the simpler the datatypes used for your parameter and return types, the more interoperable your Web service operation will be. More complex datatypes will require either custom serializers/deserializers or additional support from the Web service framework.

> **Note:** As a rule, do not try to return an Agile API object, such as `IAgileSession` or `IItem`, from a Web service. Web services should only return data structures.

# Authenticating Users

All default out-of-box Web services and user customized versions are protected by the application server. To access a protected Web service, add the following lines in your Web service client stub code:

***Example 2–1 Accessing a protected Web Service***

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

To remove the Web container protection for a specific Web service, add the lines in the following applications:

```
application.ear#integration.war/WEB-INF/web.xml
```

and

```
application.ear#integration.war/WEB-INF/web.xml files:
<security-constraint>
<web-resource-collection>
<web-resource-name>Unprotect web services</web-resource-name>
<url-pattern>/ws/<web service name></url-pattern>
<url-pattern>/services/<web service name></url-pattern>
</web-resource-collection>
</security-constraint>
```

## Using Single Sign-On Cookies for Client-Server Access

After a user on the WSX client is authenticated by the Agile 9.X server which is protected by third party single sign-on products, the browser is granted a Single sign-on cookie. This cookie is sent to the custom j2ee Web application, provided this application is in the same DNS domain as the Agile 9.X server. Now, to invoke the Web service deployed on Agile 9.X server, you can pass the single sign-on cookie instead of username and password as a valid credential.

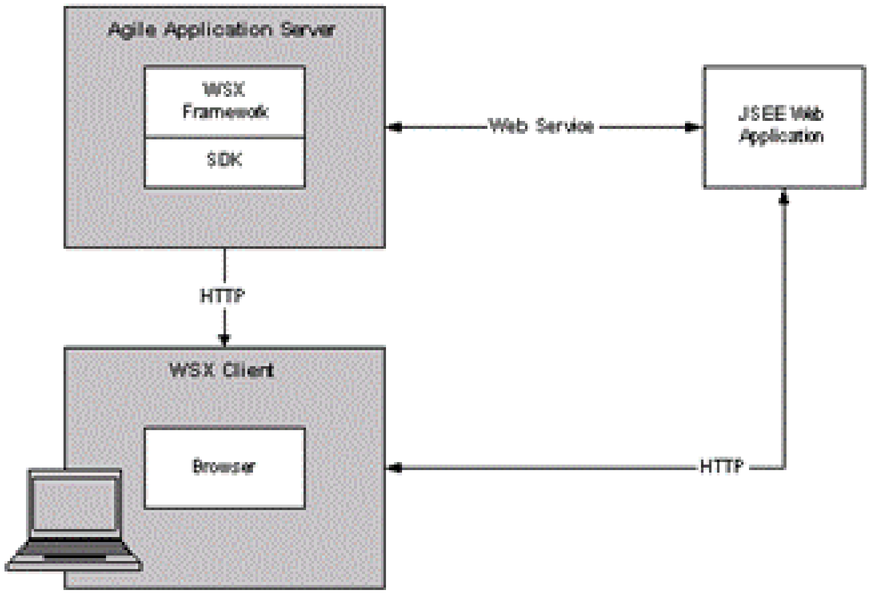
---

**Note:** If you are using both username and or password and single sign-on cookies, the single sign-on cookie has precedence over username or password.

---

### Deployment Architecture

Interactions and the request flow between the Agile server and WSX client is summarized in the following illustration.




### Invoking the Web Service Client with a Single Sign-on Cookie

This is accomplished by first, retrieving the single sign-on cookie from the HTTP request followed by modifying the SOAP binding stub code.

#### *Retrieving the Single Sign-On Cookie*

Before invoking the Web service client stub, you must retrieve the single sign-on cookie in the HTTP request. By default, the single sign-on cookie provided by SiteMinder is called SMSESSION. Modify the cookie to the format specified in RFC2965 available at `http://www.ietf.org/rfc/rfc2965.txt`. The simplest format is `name=value` where you can access both name and value by calling the `javax.servlet.http.Cookie object` method.

## Preparing the Environment for MyFirstWebService

To explain developing a Web service, a sample that highlights the development process is provided. The sample, called MyFirstWebService, is a simple example that demonstrates how to create a Web service that can use the Agile SDK to retrieve information about a particular Item and return the Item as the result of the Web service operation.

To support the desired operation, the following entry point is defined:

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

MyFirstWebService sample uses a third party library called Jakarta Commons CLI and parses args as a set of command line arguments. Based on these arguments, the results are returned as a String. You can find more information on implementation details in the SDK_samples.zip folder described in "Client-Side Components" on page 1-2. The path to MyFirstWebService is in `samples\wsx\src\first`. The remaining paragraphs in this section describe the deployment process and do not address implementation details.

## Downloading Tools to Build the Sample

To build and deploy the MyFirstWebService sample, you must download the following tools:

| Tool | Download Site |
|---|---|
| Java 8 SDK SE Version 1.7 | `http://www.oracle.com/technetwork/java/archive-139210.html` |
| Apache Project's Ant build tool, version 1.9.4 | `http://archive.apache.org/dist/ant/binaries/` |

## Installing the Java SDK

This section provides the instructions to install the Java SDK on Windows and on Solaris platforms. You can skip this section if you already have the proper version of Java installed.

**To install the Java SDK on Windows:**

1. Double-click the distribution and follow the installation procedures.
2. Set the system variable JAVA_HOME to point to the home directory of your Java SDK installation (for example, D:\j2sdk150).

**To install the Java SDK on Solaris:**

1. Execute the distribution (for example, $ ./ j2sdk-1_5_0-solaris-sparc.sh) and follow the installation procedures.
2. Set the environment variable *JAVA_HOME* to point to the home directory of your Java SDK installation (for example, */home/user*/j2sdk150).
3. Execute your .profile or .cshrc (depending on your shell) file to re-initialize your environment settings.

## Installing Ant

This section provides the necessary instructions to install Ant on Windows and Solaris environments. Use the information in "Downloading Tools to Build the Sample" on page 2-9. to download the Ant tool.

**To install the Ant on Windows:**

1. Extract the contents of the Zip archive to a local directory and follow the installation procedures.

   The Ant distribution for Windows is a zip file (for example, `apache-ant-1.6.5-bin.zip`).

2. Open a command prompt window and verify that Ant can be invoked by typing this command **%ANT_HOME%\bin\ant -version**.

   The following message is displayed:

   Apache Ant version 1.9.4 compiled on <date>

**To install Ant on Solaris:**

1. Extract the contents of the tar archive to a local directory (for example, `/home/user/ant`).

   The ANT distribution for UNIX is a tar file (for example, `apache-ant-1.9.4-bin.tar.gz`).

2. Execute your `.profile` or `.cshrc` (depending on your shell) file to set up your environment.

3. From a command prompt, verify that Ant can be invoked by typing this command **$ANT_HOME/bin/ant -version**.

   The following message appears:

   Apache Ant version 1.9.4 compiled on *compilation_date*

## Building MyFirstWebService Sample

Agile provides several sample programs for the SDK, including a sample Web service program called MyFirstWebService. To download the sample program, refer to the **Note** in "Client-Side Components" on page 1-2. The MyFirstWebService sample is in the *wsx* folder in SDK Samples.

The Ant tool reads the build.xml script and builds all targets in the following sequence on the server that is running the WSX samples:

- Compiles the Java code for the Web service into MyFirstWebService.jar.
- Copies the resulting MyFirstWebService.jar file, which includes the `commons-cli.jar` file into the`../sdk/extensions` folder.
- Generates a script (either runner.bat or runner.sh) to run the client. (It conveniently sets the CLASSPATH needed to run the client.)
- Generates client-side stub files and places them in the following folder:

  `sdk\samples\wsx\built\src\client` located in Oracle Agile PLM Event and Web Services Samples. To access this folder, see the **Note** in"Client-Side Components" on page 1-2.
- Compiles the client classes and places them in your local server.

**To build the WSX sample on the server platform:**

1. Copy the SDK_samlpes (ZIP) file. For information to access this file, see the **Note** in "Client-Side Components" on page 1-2.
2. Go to `$AGILE_HOME/sdk/samples/wsx` on the server.
3. Build the MyFirstWebService sample using the sample's build.xml file:
   - **On Windows** - `%ANT_HOME%/bin/ant`
   - **On Solaris/Linux** - `$ANT_HOME/bin/ant`

---

**Important:** If you are not building the Web service sample under `$AGILE_HOME/sdk/samples/wsx`, then upload `wsx/built/MyFirstWebService.jar` into `$AGILE_HOME/integration/sdk/extensions`. This directory is configurable in agile.properties on the server. Because the SDK will not generate WSDL files or WSXs when you invoke `http://hostname:port/virtualPath/services/MyFirstWebService?wsdl`, it will not return the required WSDL file. To generate these files, do as shown in the next step.

---

4. In the WSX folder, invoke the applicable command below to generate the WSX stub.

   - **On Windows** - `%ANT_HOME%/bin/ant-Dwsx.url=http://webserver/virtualPath/services -Dusername=`*`username`* `-Dpassword=`*`password`*
   - **On Solaris/Linux** - `$ANT_HOME/bin/antDwsx.url=http://webserver/virtualPath/services -Dusername=username -Dpassword=password`

# About Web Service Clients

This section describes the tools that you can use to develop client applications and languages that can generate and process XML files and HTTP request/response messages.

## Client Programming Languages

Although Agile tests and certifies Java for use in developing AIS clients, SOAP messages are platform- and language-independent, which means you can use virtually any client programming language that can generate and process XML and process HTTP request/response messages. For example, you can develop clients in Java, Visual Basic.Net, C++, C, or Perl.

There are helpful libraries for Java, .Net, Perl, Python, C++, and C, and for other environments as well. Here are some websites where you can find more information:

- **Apache Axis** - Open source SOAP implementation for Java: `http://ws.apache.org/axis/`
- **Java Web Services Developer Pack (JWSDP)** - Sun's Java implementation of the SOAP protocol:`http://www.oracle.com/technetwork/java/index-jsp-136025.html`
- **Microsoft .Net** - An XML Web services platform for Microsoft Windows that you can use to create Web service clients: `http://msdn.microsoft.com/net`
- **SOAP::Lite for Perl** - A Perl implementation of the SOAP protocol: `http://www.soaplite.com/`

---

**Note:** For a comprehensive list of other SOAP implementations, refer to the following website: `http://www.soapware.org/`

---

## Accessing a Web Service

In general, to access a Web service, you need to do the following:

1. **Generate a SOAP request** - In many cases, a Web-service-aware code library will be able to generate client-side stubs that generate an appropriately formatted SOAP request.
2. **Submit that request to WSX through HTTP or HTTPS** - Once an appropriate set of client-side stubs has been generated, a client application can use these stubs to submit a request.
3. **Process the SOAP response** - The client-side stubs usually are responsible for processing the SOAP response and converting the response into an appropriate set of return objects.

The WSX samples provide examples of how SOAP and Web service-related libraries can make this process simple. The following sections illustrate, using the MyFirstWebService sample, the above steps in greater detail.

# Creating a Web Service Client

When you build and deploy MyFirstWebService, you also automatically generate the client-side stubs and the client classes. This section uses MyFirstClient as an example to describe some general aspects of how to create a Web service client.

## Generating the SOAP Request

In most cases, generating an appropriate SOAP request is as simple as making use of client-side stubs. Many Web-service-aware code libraries are able to generate client-side stubs for you. This entails using a code generation utility along with the WSDL for the desired Web service.

Axis provides a WSDL2Java utility that you can use to generate client-side stubs. Other Web-service-aware libraries have their own client-side stub generation facility. Microsoft.Net includes a wsdl.exe utility. In the case of the WSX samples, the client-side stub generation occurs during the sample's build process.

This Ant target is responsible for generating the client-side stubs for MyFirstWebService. This invocation retrieves the MyFirstWebService WSDL from `${ws.url}/MyFirstWebService?wsdl`, generates Java code in the client Java package, and places the source code within the `${built.dir}/src` directory.

Within the `build.xml` file, you will find the following Ant target:

```
<target name="generate-stubs-file" depends="init">
   <clientgen wsdl="${wsx.url}/${serviceName}?wsdl"
             destDir="${built.dir}/src"
             type="JAXWS">
   </clientgen>
  <jar jarfile="${built.dir}/MyFirstWebServiceStub.jar">
     <fileset dir="${built.dir}/src" includes="**/*.class"/>
   </jar>
</target>
```

Once the client-side stubs have been generated, the user can use the generated object definitions to easily generate the appropriate SOAP request. Rather than requiring the user to understand how to construct a valid SOAP request, these stubs allow the user to focus on the capabilities of the target Web service operation. Looking at the

MyFirstClient.java sample found within `..\samples\wsx\src\client`, note that the main method contains all the code used to generate the SOAP request.

## Submitting the SOAP Request

The next step in a Web service operation is to properly submit the generated SOAP request to the Web service engine. When dealing with generated client-side stubs, this step is usually as simple as pointing the stubs to the desired server and invoking a method on the stubs. You do not need to worry about opening a connection. Instead, the generated stubs handle these details for you.

The `MyFirstClient.java` sample found within `..\samples\wsx\src\client` illustrates how to submit the SOAP request in two places:

- The `getStub()` method is responsible for pointing the client-side stubs to the desired Web service engine.
- The `stub.getItemField()` method invocation found within the main method is responsible for submitting the request to the Web service engine. That is, stubs themselves manage the submission of requests and

  The submitting of the request is managed by the stubs themselves; you do not need to worry about the `.Stubs` themselves manage the submission of the request, and you do need to perform the connection or marshaling particulars

The details on how you point the stubs to the desired Web service engine and submit the request varies for different code libraries. To this end, Oracle recommends that you consult the documentation for your `Web-service-aware` code library for the necessary information.

## Processing the SOAP Response

The processing of the SOAP response is usually handled through the generated client-side stubs. Without these generated stubs, you would be responsible for parsing the XML-based SOAP response and dealing with the many formatting and un-marshaling issues that arise. However, when dealing with generated stubs, all of these details are taken care of for you, allowing you to receive properly typed Java objects. Rather than require you to parse an XML document and discern what the returned data is, the stubs automatically do this for you.

The details on how SOAP responses are processed will vary from code library to code library. Some SOAP servers expect the client to know the datatype through some other means (perhaps WSDL). Consult the documentation for your `Web-service-aware` code library for more information.

## Running the Sample on the Web Service Client

To build and deploy the MyFirstWebService sample, use the CLASSPATH initialization information in the runner.sh (UNIX) or runner.bat (Windows) to run sample on the Web service client. You can find these files in the `SDK_samples.zip`. To access `SDK_samples.zip`, see the **Note** in "Client-Side Components" on page 1-2.

To print a usage statement for MyFirstClient, type the following command:

```
> runner client.MyFirstClient
```

The following usage statement returns the “Title Block.Description“field for part 1000-02:

**> runner client.MyFirstClient -T 15000 -a "***attribute_name*"
-e *virtual_path* -h *host* -l *port_no.* -n *item_number* -p *password* -u *username*

**> runner client.MyFirstClient -T 15000 -a "Title Block.Description"**
-e Agile -h localhost -l 80 -n 1000-02 -p agile -u jeffp

## Creating an Agile Session inside WSX

By default, the Web container protects the WSX. Therefore, you must specify user credentials when creating an Agile Session inside the WSX. The following example creates an Agile session within a protected WSX.

***Example 2–2 Setting up a session inside a WSX***

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

---

**Note:** Do not override the implicit session.

---

To have a different user, you must make an explicit SDK session as if connecting from a remote client. That is, provide an argument to the AgileSessionFactory.getInstance() method as shown in the following example.

***Example 2–3 Creating an explicit session independent of the implicit session***

```
AgileSessionFactory factory = AgileSessionFactory.getInstance ("http://...");
    params.put(AgileSessionFactory.USERNAME, ...);
    params.put(AgileSessionFactory.PASSWORD, ...);
    IAgileSession session = factory.createSession(params);
```

# Microsoft .NET Interoperability

Microsoft's .NET framework technology is a development framework that provides an application programming interface (API) to the services and APIs of classic Windows operating systems, while bringing together several disparate technologies that emerged from Microsoft in the late 1990s: ASP, COM+, XML, SOAP, to name a few.

`.NET` also brings together all the languages provided by the Visual Studio environments provided by Microsoft such as Visual Basic, J++, and C++. Also, new languages have been developed - such as C# (read C Sharp) and the relatively new language to the `.NET` family, J# (read J Sharp). J# is actually Java in Microsoft disguise providing integration of Java into the `.NET` framework. Yet, J# will not work with the Java `VM. J#`, in essence, acts as a wrapper to contain Java-enabled code to be executed by the `.NET` Common Language Runtime (CLR), Microsoft's own virtual machine (VM).

The CLR is probably the most important component to the `.NET` framework. The CLR provides for the activation of objects, security checks, memory management, object execution, and memory cleanup (garbage-collection) when objects are no longer being used.

Another factor behind `.NET` is that it not only provides for the writing of Windows-based applications or Web-based applications (`throughASP`) by using any of the languages mentioned, it can also integrate these languages into one common API. Thus, developers can write language independent code that inherit from classes, catch exceptions, and take full advantage of polymorphism across differing languages across the `.NET` framework.

---

**Important:** Although the WSX framework (the AXIS SOAP processor) works fine with AXIS Web service clients, it is not completely compatible with `.NET`. Neither Microsoft nor the Apache group have conducted interoperability tests for AXIS and .Net. For simple data types, AXIS-based Web services should work fine with .Net-based Web service clients. For some complex data types (such as binary attachments), you may experience interoperability problems. For interoperability information about non-AXIS Web service implementations deployed outside of the Agile Application Server, contact the specific Web service vendor.

---

## Web Service Extensions FAQs

This section answers common questions about Web service extensions.

**What is Web service extensions (WSX)?**

WSX is a framework for Agile customers to extend the functionality of the Agile PLM server using Web services.

**What are Web services?**

Web services use the SOAP messaging protocol to provide software services over the Internet, allowing software components to interact with each other around the world. Web services are not tied to any one operating system or programming language. They use WSDL to describe a service's public interface, essentially making Web services self-describing and therefore relatively easy to use.

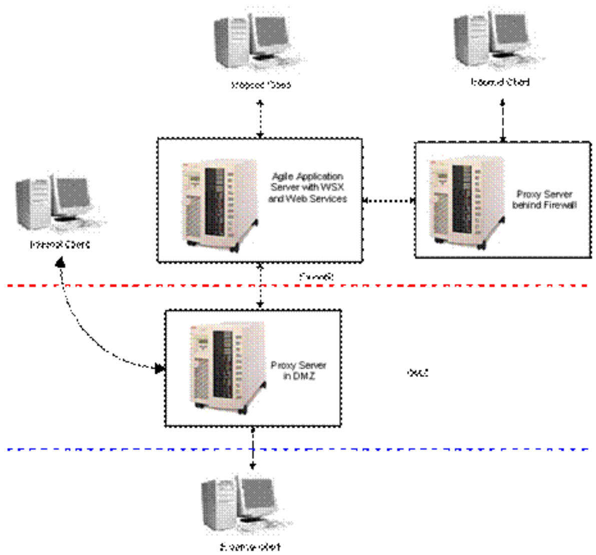
**What can I do with WSX that I cannot do solely with Agile's Java API?**

WSX provides firewall-friendly, XML-based integration with Agile PLM data using the standard HTTP(S) protocol. It supports any SOAP-compliant programming language. For example, you can create Perl or `.NET` clients for a Web service. WSX enables systems in different companies to interact with each other easily and securely. Services deployed within WSX take advantage of all the scalability, failover, and clustering features provided by the application server. There are also compelling performance benefits to services that run on the application server.

**Does WSX support both secured and unsecured connections?**

Yes. Requests that come to a Web service from outside the firewall are subject to different security requirements from requests that originate within the firewall. Two separate entry points are provided for each WSX, external (outside the firewall) or internal. External requests are made against a proxy server and then forwarded to the application server. The proxy server resides in the DMZ. Internal requests can be made against the same secure proxy server, another proxy server that does not reside in the DMZ, or directly against the application server, as shown in the following figure.

***Figure 2–2 How Web service clients connect to the Agile PLM server***




**What user authentication services are provided by WSX?**

By default, WSX is protected by the application server. Username and password security is enforced whenever a WSX client invokes a service that is protected. For more information, see "Authenticating Users" on page 2-7.

**What SOAP engine does WSX use?**

WSX is based on Apache Axis, an open-source implementation of SOAP. For more information about Axis, refer to the Axis website at `http://ws.apache.org/axis/`.

**Does WSX handle SOAP attachments?**

Yes. In fact, Agile Integration Services provides export Data and import Data services that let you export and import binary attachment files.

**Does WSX support protocols other than HTTP?**

No. WSX supports only HTTP-related protocols. For additional security, you can connect to a Web service using HTTPS and SSL. Over time, WSX may support additional protocols as needed.

**Does WSX support Perl, Python, PHP, or other Web scripting languages?**

WSX supports any client programming language that can send a SOAP message. Although the Agile SDK does not provide WSX client examples in Perl, Python, or PHP, those scripting languages are certainly capable of sending SOAP messages.

**Does WSX support UDDI?**

No. UDDI is a specification for a universal business registry of Web services that's designed to enable software to automatically discover and integrate with other services. It's currently unnecessary to register Agile PLM Web services on the Internet using UDDI. Agile PLM Web services are typically created for integration with internal software systems or to exchange data with partners or suppliers. However, Agile may consider supporting UDDI as the technology matures.

**How do I deploy a Web service?**

Place the service's JAR files in the *agile_home*/integration/sdk/extensions folder on the application server computer. Included with the Web service's JAR file(s) is a deployment descriptor file called:
META-INF/services/com.agile.wsx.Deployment.wsdd.

The deployment descriptor file is an XML file formatted according to Axis' Web Service Deployment Descriptor (WSDD) format. It declares and describes the set of Web services and Web service operations that are exposed throughWSX. The WSDD file also defines any additional behavior that is necessary when processing incoming SOAP requests (such as user authentication) or responses (such as reformatting outgoing data). For more information about WSDD format, refer to the *Axis Reference Guide* at http://ws.apache.org/axis/.

**When I deploy a Web service and its JAR file(s), do I need to update the application server classpath?**

No. The classpath is updated automatically by a special-purpose classloader. The classloader extends the application server classpath with any classes located in *agile_home*/integration/sdk/extensions (or the location specified for the sdk.extensions property in the agile.properties file).

**If I make changes to a Web service and redeploy it, do I need to restart the application server?**

No. A special-purpose handler ensures that the Web services stack is updated with the latest files that have been deployed. Whenever a Web service request is made, the handler checks whether any JAR files located in *agile_home*/integration/sdk/services is updated, added, or removed. If so, the entire Web services stack is reset. This feature enables you recompile your code and redeploy a Web service without having to restart the application server, saving you precious development time.

**Are there any Agile products that use the WSX framework?**

Yes. Agile Content Service (ACS) and Agile Integration Services (AIS) both rely on WSX framework to exchange data with the Agile PLM server.

**What are Agile Integration Services?**

Agile Integration Services (AIS) are Web services that provide import, export, and partlist functionality. Included with these Web services are sample Java Clients, but you can create other SOAP-compliant AIS clients in other programming languages.

**What is basic authentication?**

Basic authentication is a simple method of authentication. It allows a client program to provide credentials in the form of an un-encrypted user name and password when making a request. There is a new Web module that uses basic authentication for deploying Web service listeners. You can find information to access Web services with basic authentication at:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=%2Fcom.ibm.websphere.express.doc%2Finfo%2Fexp%2Fae%2Ftwbs_auwschta.html

For example, use this URL for the MyFirstWebService sample files:

http://hostname/Agile/integration/ws/MyFirstWebService?wsdl

# 3

# Developing Process Extensions

This chapter includes the following:

- About Process Extensions
- Developing Custom Autonumber Sources
- Configuring Custom Autonumber Sources in Java Client
- Developing Custom Actions
- Working with AutoView Extensions
- Defining and Deploying URL-Based Process Extensions
- Creating an External Report
- Deploying Process Extensions in Clustered Environments
- Best Practices for Copying third Party JAR Files
- Process Extensions FAQs

## About Process Extensions

Process extension (PX) is a framework for extending the functionality of the Agile PLM system. The functionality can be server-side extensions, such as custom workflow actions and custom auto numbering, or extensions to client-side functionality, such as external reports or new commands added to PLM's Actions menu or the Tools menu. Regardless of the type of functionality that a process extension provides, all custom actions are invoked on the Agile Application Server rather than the local client.

---

**Note:** In addition to server-side functionalities that you can develop in the PX framework, Agile PLM's Event framework also supports developing extensions using Java and Groovy Script, called Java PXs and Script PXs respectively. Although you can a subset of the custom actions developed in the PX framework to the Events framework, the two frameworks have their own unique interfaces and are different.

---

A process extension is either a Java class deployed on the Agile Application Server, or a link to a URL. The URL can be a simple website or the location of a Web-based application. Process extensions enable the Agile PLM server and Agile PLM users to connect to external systems. You can also use process extensions to add functionalities that are not provided by the standard Agile PLM client. Using a simple yet powerful approach, process extensions open the Agile PLM system, allowing you to tailor the application to your specific business requirements.

You can use process extensions to create:

- Custom reports
- User-driven and workflow-triggered custom actions
- Custom tools accessible through Agile PLM clients
- Custom auto numbering

Multiple process extensions can be linked together within a workflow, with each process extension performing a discrete business function. Process extensions can also be used to make requests to Web services, such as services built with Agile's Web service extensions framework.

There are five integration points for process extensions available in Agile PLM clients. You can invoke process extensions from the following areas:

- External reports
- Actions menu

> **Note:** The user can not select the PX from the Action menu in the Web Client when the current object is in the edit mode.

- Tools menu
- Workflow Status
- Autonumber sources

The additive value, extensibility, and increased usability enabled by custom actions are innumerable. Oracle Consulting Services (OCS) can help your organization develop process extensions, including reporting, data import/export, process automation, validation, and other areas across different Agile modules (PC, PG&C, PPM, PCM, EC). For more information and to purchase these PXs, please contact your Oracle representative, Oracle Agile Consulting Services, or you can attend Process Extension training through Oracle University.

## Developing Custom Autonumber Sources

Most Agile PLM object classes have at least one default autonumber source that lets you create a new object and automatically number it with the next number in the sequence. Autonumbers can have both an alphanumeric prefix or suffix. You can also specify the length of the autonumber (a string) and which alpha-numeric characters to use.

Despite the flexibility that autonumbers provide, some companies have specific numbering requirements that are not accommodated by Agile PLM's standard autonumbering capabilities. Such companies can define custom autonumber sources and add them to the Agile PLM system using the process extensions framework.

If you have administrator privileges, you can define autonumber sources in Agile Java Client. An autonumber source can use the client's standard numbering capabilities, or it can be associated with a custom autonumber source. When an Agile PLM client uses a custom autonumber source to create a new object, the Agile Application Server invokes the custom Java code to generate the number.

## Defining a Custom Autonumber Source

To define a custom autonumber source, create a Java class that implements the `ICustomAutoNumber` interface, a server-side API in the com.agile.px package. The code should define the autonumbering logic, for example, prefix, suffix, number of digits, character set, and so on, and the persistence mechanism. Regarding persistence, the location where your custom autonumber source stores numbers is entirely up to your program. For example, you can store numbers in a SQL database like Oracle or in a file.

The Agile PLM server gets the next number from the custom autonumber source by calling the `getAutoNumber()` method, which must be provided in your class. The following example shows how to implement a Java class for a custom autonumber source.

***Example 3–1 Defining the class for a custom autonumber source***

```
package autonumbers;

import com.agile.px.*;
import com.agile.api.*;

public class ResistorNumber implements ICustomAutoNumber{
     public ActionResult getAutoNumber(IAgileSession session, INode actionNode)
     {
       String num;
     // Write your code here to define the custom autonumber source for
        Resistors
     return new ActionResult(ActionResult.STRING, num);
    }
}
```

## Packaging and Deploying a Custom Autonumber Source

After you develop classes for a custom autonumber source, follow these instructions to properly package and deploy them.

**To package and deploy a custom autonumber source:**

1. Use your Java development environment or the Java Archive tool (or JAR tool) to create one or more JAR files for the custom autonumber source. Make sure the JAR file(s) includes a `META-INF/services` directory that contains a file named `com.agile.px.ICustomAutoNumber` which is a text file that lists the fully qualified Java class names, one class per line, for the custom autonumber source.

   > **Note:** When creating the Jar files, make sure you are using the Java version that is running on the Agile server.

   Multiple custom autonumber sources can be included in one package. For example, the `com.agile.px.ICustomAutoNumber` file could look like this:

   ```
   autonumbers.ResistorNumber
   autonumbers.CapacitorNumber
   autonumbers.DiodeNumber
   ```

---

**Note:** Paths within a JAR file are case-sensitive. Therefore, make sure the `META-INF` folder contained within the JAR file has a name with all uppercase or all lowercase characters. Otherwise, the custom autonumber source will not be deployed.

---

2. Place the JAR file(s) in the *agile_home*/integration/sdk/extensions folder on the same computer where the Agile Application Server is installed.

---

**Note:** If you have several application servers in a clustered environment, you must deploy process extension files on each server in the cluster.

---

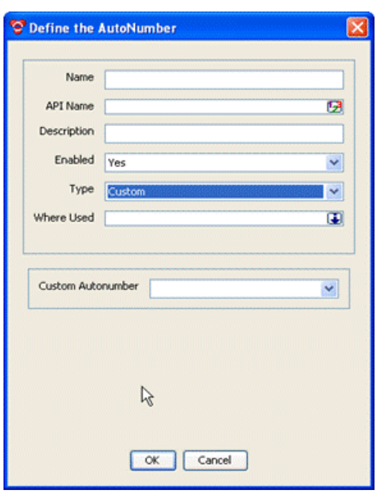
## Configuring Custom Autonumber Sources in Java Client

In Agile Java Client, you can define autonumber sources in the Admin module. To configure Agile PLM system settings, you must have an administrator account.

**To add a custom autonumber source:**

1. Log in to Agile Java Client as an administrator.
2. Click the **Admin** tab.
3. Go to **Settings > Data Settings** > **AutoNumbers**.
4. Click the **AutoNumbers** node.
5. In the AutoNumbers window, click the **New** button. The Define the Autonumber dialog box appears.

***Figure 3–1 The Define Autonumber dialog box***




6. Provide the following information:
   - **Name** - Type the name of the autonumber source.
   - **API Name** - This field is automatically filled in after completing the Name field. Refer to Accessing PLM Metadata with APIName Field in *SDK Developer Guide - Using the APIs*.

- **Description** - Type a brief description of the autonumber source.
- **Enabled** - Select **Yes** or **No.**
- **AutoNumber type** - Select Custom. This activates the Custom AutoNumber field.
- **Where Used** - Select the subclass(es) that can use this autonumber source.
- **Custom AutoNumber** - Select a custom autonumber source from the list (this is where your class will appear).

7. Click **OK** to save the autonumber definition.

## Assigning Autonumber Sources to a Subclass

When you define an autonumber source, you can specify the subclasses where it is used in the Where Used field. You can also assign an autonumber source to a subclass in the Classes node.

**To assign autonumber sources to a subclass:**

1. Log in to Agile Java Client as an administrator.
2. Click the **Admin** tab.
3. Open the **Data Settings** folder.
4. Open the **Classes** node.
5. In the Classes window, double-click a subclass. The subclass window appears.
6. In the Autonumber Source field, click the Down Arrow button. A popup window appears.
7. Select autonumber sources in the Choices list, and then click the Right Angle Button to move them into the Selected list. When you are finished, click **OK**.
8. Click **Save** to save settings.

# Developing Custom Actions

This section describes how to develop custom actions in Java classes. The Agile PLM clients can make direct method calls to these classes to perform the actions.

You can initiate a custom action from the following areas of Agile PLM clients:

- Actions menu
- Tools menu
- External reports
- Workflow Status

## Defining a Custom Action

To define a custom action, create a Java class that implements the ICustomAction interface, a server-side API in the com.agile.px package. The code should define the action to perform. The Agile PLM server initiates the action by calling the doAction() method, which must be provided in your class.

The following example shows the code for a HelloWorld class. When the doAction() method is called, the method returns “Hello World.” If you invoke the HelloWorld custom action from Actions menu, the string “Hello World” will be logged to the object's History table. If you invoke the HelloWorld custom action from a workflow, the string “HelloWorld” will be logged to the change order's History table when it enters the appropriate Workflow status.

***Example 3–2 Defining a HelloWorld class for a custom action***

```
package actions;

import com.agile.px.*;
import com.agile.api.*;

public class HelloWorld implements ICustomAction
{
     public ActionResult doAction(IAgileSession session, INode actionNode,
           IDataObject affectedObject)
     {
        return new ActionResult(ActionResult.STRING, "Hello World");
     }
}
```

The above HelloWorld class does not perform a useful action, it simply demonstrates how to implement the class for a custom action.

### Formatting New Lines (Line Breaks) in PLM Clients

You can configure new lines in `ActionResult` outputs in Web and Java clients using the '\n' character.

---

**Note:** SDK only supports the character '\n' for new lines in both Web and Java clients.

---

**To format new Line Breaks in PLM:**

1. Create a Process Extension that includes with the following ActionResult string.

   ```
   return ActionResult(ActionResult.STRING, "Hello \n This example tests
   formatting new lines \n in the SDK");
   ```

2. Run the Process Extension in the Web and Java clients from Actions menu and Tools menu.

   ```
   The output should be as shown below:
   Hello
   This example tests formatting new lines in the SDK.
   ```

## Custom Actions and User Sessions

When an Agile PLM client invokes a process extension, it does so within the current user's session. Therefore, the process extension should not create any additional `IAgileSession` objects using the Agile API within the process extension code or any code directly invoked from the process extension. Stated simply, process extensions never directly create new Agile PLM sessions.

If you have written a Web service extension (WSX) and want to use that code from within a process extension, you can directly invoke Java methods contained in WSX classes without using the Web services infrastructure, provided those methods do not create a new `IAgileSession` object.

Do not mix Process extension (PX) invocations with Web service extension (WSX) invocations. The PX code must not invoke any WSX code directly, especially when the PX and WSX reside in the same application container. If a process extension uses Web services, WSX could create a new Agile PLM session that is distinct from the session that the process extension uses.

URL-based process extensions can call an external application that communicates with the Agile PLM server and performs some action upon the currently selected business object. To perform such an action, the external application can use the Agile API to create another Agile PLM session. For more information, see "Creating an Agile PLM Session from the Target System" on page 3-16.

## Packaging and Deploying a Custom Action

After you develop classes for a custom action, follow these instructions to properly package and deploy them.

**To package and deploy a custom action:**

1. Use your Java development environment or the Java Archive tool (or JAR tool) to create one or more JAR files for the custom action. Make sure the JAR file(s) includes a META-INF/services directory that contains a file named `com.agile.px.ICustomAction`, which is a text file that lists the fully qualified Java class names, one class per line, for the custom action.

   You can include multiple custom actions in one package. For example, the com.agile.px.ICustomAction file can look like this:

   ```
   actions.HelloWorld
   actions.RFQConsolidation
   actions.RefreshCustomerFromCRM
   actions.StartMfg
   actions.ObsoletePartReplacer
   actions.WorkflowConflictResolver
   ```

   > **Note:** Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file has a name with all uppercase or all lowercase characters. Otherwise, the custom action will not be deployed.

2. Place the JAR file(s) in the *agile_home*/integration/sdk/extensions folder on the same computer where the Agile Application Server is installed.

   > **Note:** If you have several application servers in a clustered environment, you must deploy process extension files on each server in the cluster.

## Roles and Privileges for Custom Actions

When you configure a custom action in Agile Java Client, you can specify the roles it uses. By default, a custom action uses the roles of the current user. However, you can configure a custom action to have expanded privileges. This is an important feature of process extensions. In effect, you can enforce the business logic of a custom action by granting it more privileges than those given to ordinary users. When a custom action is invoked in the Agile PLM client, its roles and privileges contained within the roles override the privileges of the current user. Once the custom action is completed, the client reverts to the user's privileges.

### User Privileges for Configuring Process Extensions

To configure a Process Extension, you must have necessary user privileges to get the user's language setting. If a PX fails, the error message should display in the user's current language. If the user's roles are not set to include the privilege to load current user object info, the server will display all messages in the default system language.

## Roles and Privileges for Checking User Privileges

To determine whether a user has the necessary privileges to perform an action at run time, you must use the `IUser.hasPrivilege()` method.You can adjust your program's logic based on the user's privileges.

The following example shows how to check if the user has the necessary privileges to modify the object before calling the `IDataObject.setValue()` method.

***Example 3–3 Checking a user's privileges before modifying the object***

```
public void checkPriv(IUser user, IDataObject obj) throws APIException{
    Boolean userHasPrivOriginalUser =
              user.hasPrivilege(UserConstants.PRIV_MODIFY, obj, true);
    System.out.println("Check Original User has the modify priv : " +
              userHasPrivOriginalUser + " #\n");
    Boolean userHasPrivInPX =
              user.hasPrivilege(UserConstants.PRIV_MODIFY, obj, false);
    System.out.println
      ("Check the modify priv in assigned PX roles: " + userHasPrivInPX + " #\n");
    if(userHasPrivInPX){
IAgileClass agileClass = obj.getAgileClass();
            IAttribute attr =
                 agileClass.getAttribute(AttributeTypeConstants.TYPE_DESCRIPTION);
            String desc =
                 "HERE IS A NUMBER: " + System.currentTimeMillis();
            obj.setValue(attr, desc);
            System.out.println("Set description to '" + desc + " #\n");
            }
}
```

**Note:** When you call the `hasPrivilege()` method in multiple new threads which are created in Custom Actions, the method only checks the original user's privileges, which does not include the assigned PX roles.

## Configuring Custom Actions in Agile Java Client

In Agile Java Client, you can define custom actions in the Admin module. To configure Agile PLM system settings, you must log in as a user with administrator privileges.

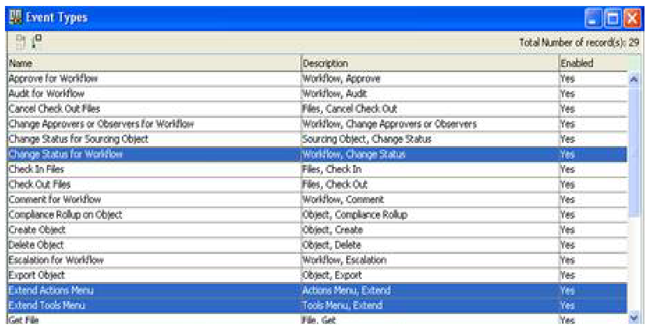
### Using the Process Extension Library

The Process Extension Library is where you define the custom actions that can be used in Agile PLM clients. When you add a custom action to the Process Extension Library, you specify how to initiate that action from the client.

**To add a custom action to the Process Extension Library:**

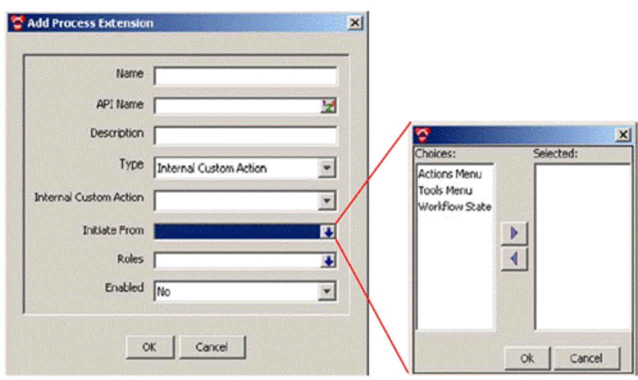
1. Log in to Agile Java Client as an administrator.
2. Click the **Admin** tab.
3. Open the **Data Settings** folder.
4. Open the **Process Extensions** node.
5. The Process Extension Library

***Figure 3–2 The Process Extension Library***




6. In the Process Extension Library window, click the Add Process Extension button to open the Add Process Extension dialog box.

***Figure 3–3 The Add Process Extension dialog***




7. Type the following information:
   - **Name** - Type the name of the process extension.
   - **API Name** - This field is automatically filled in after completing the Name field. Refer to Accessing PLM Metadata with APIName Field in *SDK Developer Guide - Using the APIs*.
   - **Description** - Type a brief description of the process extension.
   - **Type** - Select Internal Custom Action. This activates the Internal Custom Actions field.
   - **Internal Custom Action** - Select a custom action from the list (your classes should appear here).

- **Initiate From** - Select one or more locations from which the process extension can be initiated. Choose from the following options:
  - **Actions menu** - Enables you to select the custom action from the Actions menu of a properly configured class.
  - **External report** - Enables you to generate a report by accessing an external resource or URL. If the process extension is an internal custom action, the External Report option is unavailable.
  - **Tools menu** - Enables you to select the custom action from the Tools menu.
  - **Workflow status** - Invokes the custom action whenever a properly configured workflow enters a particular status.

    If you specify that a process extension is initiated from the Actions menu or a workflow status, you can configure subclasses or workflows to use it. If you specify a process extension to generate an external report, you can use Agile Web Client to create the report. If you specify that a process extension is initiated from the Tools menu, it is always available in the Agile PLM client.
- **Roles** - Select one or more roles to use for the custom action. To use the roles of the current user, leave this field blank. To temporarily override roles of the current user, select one or more roles. Once the custom action is completed, the client reverts to the current user's roles.
- **Enabled** - Select Yes or No.

8. Click **OK** to save the new process extension.

### Assigning Process Extensions to Classes

To add custom actions to the Actions menu of an Agile PLM object (such as a Part or an ECO), you configure the object's class. Each base class, class, and subclass has a Process Extensions tab. The custom actions that you assign to a class must be previously defined in the Process Extension Library.

Process Extensions are inherited from classes and base classes. Consequently, if you assign a process extension to a base class, it is also assigned to classes and subclasses beneath the base class.

> **Note:** Process extensions can be assigned to only one level in a class hierarchy. For example, if a process extension is assigned to the Part subclass, you cannot assign it to the Item base class.

**To assign process extensions to a class:**

1. Log in to Agile Java Client as an administrator.
2. Click the **Admin** tab.
3. Open the **Data Settings** folder.
4. Open the **Classes** node.
5. In the Classes window, double-click a base class, class, or subclass.
6. Click the **Process Extensions** tab.
7. In the toolbar, click the Assign button. The Assign Process Extension dialog box appears.

8. Select custom actions in the Choices list, and then click to move them into the Selected list. When you are finished, click **OK**.
9. Click **OK** to save settings.

### Assigning Process Extensions to Workflow Statuses

For each workflow status except the Pending status, you can assign one or more custom actions that are initiated when the workflow enters that status. The custom actions you assign to a workflow status must be previously defined in the Process Extension Library.

> **Note:** Automated Transfer Orders (ATOs) do not support workflow-triggered process extensions.

**To assign process extensions to a workflow status:**

1. Log in to Agile Java Client as an administrator.
2. Click the **Admin** tab.
3. Open the **Workflow Settings** folder.
4. Open the **Workflows** node.
5. In the Workflows window, double-click a workflow, for example, a Default Workflow.
6. Click the **Status** tab.
7. Select a status other than Pending. The selected status updates the Workflow Criteria properties table that appears below the status table.
8. Double-click the selected status criterion in the Workflow Criteria properties table.
9. In the Process Extensions list, click the Down Arrow button. A popup window appears.
10. Select custom actions in the Choices list, and then click to move them into the Selected list. When you are finished, click **OK**.
11. Click **Save** to save settings.

# Working with AutoView Extensions

This section describes AutoVue, AutoVue's *Augmented Business Visualization* (ABV), and shows using AutoVue's ABV to view PLM Change objects in AutoVue.

## About AutoVue and AutoVue Viewer

*AutoVue* is a solution that enables viewing and marking up documents, drawings, and CAD files using a Web browser. AutoVue's ABV is an integration of Autovue and Agile PLM that supports executing context-based operations in Agile by selecting a part in *AutoVue Viewer* applet and choosing a related operation. For a list of functions that are supported by AutoVue Viewer, refer to `https://blogs.oracle.com/enterprisevisualization/entry/context_is_everything`.

The overall solution that enables this interaction, involves Javascript and PX code to interact with Agile APIs.

- The Javascript and PX code are authored, tested, and maintained by the AutoVue team. The Javascript component is bundled as part of the standard Agile install, and connected to the standard initialization process for the AutoVue Client in Agile.
- The APIs required for this solution are authored, tested, and maintained by the Agile team.

---

**Note:** The focus of this SDK Guide is `IAutoVueExtension` and the other PLM Java interfaces that supports this interaction.

---

## PX Interfaces for AutoVue Extension

The `IAutoVueExtension` interface in the PX framework supports PLM interactions with AutoVue Viewer.

```
public interface IAutoVueExtension {
public String getDefinition(IAgileSession session);
public AutoVueActionResult doAction(IAgileSession session, Map param);
}
```

The `getDefinition` method returns the *hotspot* definitions used by the AutoVue Viewer applet. A hotspot definition is a string of semicolons that are separated by key-value pairs and contain attributes such as "Create ECO" or "Create ECR".

***Example 3–4   A string of semicolon separated key-value pairs in Java***

```
String def =
   "DEFINITION_REGEX=server;" +
   "DEFINITION_MATCHCASE=false;" +
   "DEFINITION_TOOLTIP=AutoVue 2D Professional;" +
   "DEFINITION_COLOR=(0,0,255,128);" +
   "DEFINITION_ACTIONS=Menu1, Menu2;" +
   "DEFINITION_TYPE=DEFINITION_TYPE_TEXT;DEFINITION_KEY=agileDef";
```

With the exception of the `DEFINITION_TYPE` and `DEFINITION_TYPE` keys, all other keys are defined in the *Augmented Business Visualization Developer's Guide*. The DEFINITION_TYPE and `DEFINITION_KEYS` are added by this design to pass definition type and definition key information from the PX to `setHotSpotHandler` method in the AutoVue viewer applet.

The `doAction` method performs the action invoked in AutoVue viewer applet's popup menu item. The param argument contains the following information about the selected hotspot and action:

- definitionKey - definition key returned in the definition string, for example, agileDef
- hotspotKey - hotspot key
- action - the selected caption in the menu item
- properties - properties of the selected hotspot

---

**Note:** It is not necessary to configure the AutoVue extension PX in Java Client, because the PX runs with the user's roles/privilege and is not granted additional roles/privileges.

---

### Actions Performed by the Calling PX

The Agile Web Client acts as a middleman between AutoVue Viewer Applet and the PX.

- When the Viewer applet is initialized, Web Client gets the metadata used by viewer from the PX.
- When an action is triggered, Web Client collects the data from the Viewer and passes it to the PX. Upon completion of PX functions, Web Client gets the result of PX operations and performs the required task in Web Client. For example, displays a newly created object in Web Client.
- When an exception occurs in the PX, Web Client will take the applicable action and displays the result.

> **Note:** There should be only one AutoVue PX in the system. If there are more than one AutoVue PXs, the system will the first one defined in file META-INF\services\com.agile.px.IAutoVueExtension.

### Displaying Results of PX Actions

An ActionResult() method directs the behavior of the Client upon completion of AutoVue PX processes and the corresponding UI behavior.

Three Client actions upon completion of PX processes are:

- Display message - A message is displayed in the browser's window.
- Display exception - When exceptions occur upon completion PX processes, they are displayed in the browser window
- Display object - An object is displayed in Web Client's browser window

***Example 3–5 Defining Hotspots for a PDF file***

```
public class ABVActions implements IAutoVueExtension
{
   public String getDefinition(IAgileSession session) {
     String def =
      "DEFINITION_REGEX=server;" +
      "DEFINITION_MATCHCASE=false;" +
      "DEFINITION_TOOLTIP=AutoVue 2D Professional;" +
      "DEFINITION_COLOR=(0,0,255,128);" +
      "DEFINITION_ACTIONS=Display Message,Display Object,Display Exception;" +
      "DEFINITION_TYPE=DEFINITION_TYPE_TEXT;DEFINITION_KEY=agileDef";
   return def;
}

public AutoVueActionResult doAction(IAgileSession session, Map params) {
   String action = (String) params.get("action");
   if ("Display Message".equalsIgnoreCase(action)) {
      return new AutoVueActionResult(AutoVueActionResult.DISPLAY_MESSAGE,
         "Message from PX - Display Message is clicked.");
   } else if ("Display Object".equalsIgnoreCase(action)) {
    try {
      IItem item = (IItem) session.getObject(IItem.OBJECT_TYPE, "1000-02");
      return new AutoVueActionResult(AutoVueActionResult.DISPLAY_OBJECT,
         item.getId());
   } catch (APIException e) {
      e.printStackTrace();
```

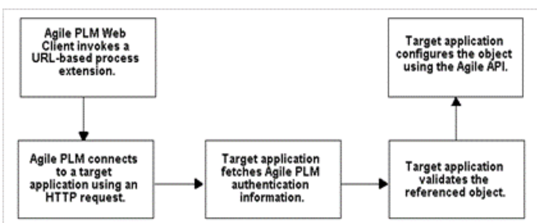
```
        }
            } else if ("Display Exception".equalsIgnoreCase(action)) {
                return new
                  AutoVueActionResult(AutoVueActionResult.SHOW_EXCEPTION,
                      "Message from PX - Display Exception is clicked.");
            }
            return new
            AutoVueActionResult(AutoVueActionResult.DISPLAY_MESSAGE,
                "Message from PX - " + action + " is performed.");
            }
    }
```

# Defining and Deploying URL-Based Process Extensions

The Agile Web Client uses the URL-Based Process Extensions to provide access from the Web Client to external applications.When the Agile PLM Web Client invokes a process extension that references a URL, the client displays the Web page in a new browser window.

What types of Web-based applications could be used for URL-based process extensions? Again, there are few limitations. One example might be a Web-based application that performs business rules validation for an Agile PLM object and updates the object accordingly. The following figure shows the program flow of such a process extension.

***Figure 3–4 Process flow for a URL-based process extension***




You can also use URL-based process extensions to reference a Web-based report engine. To create an external report that uses a URL-based process extension, choose **Create >Report > External** in Agile Web Client. For more information, see "Creating an External Report" on page 3-20.

---

**Note:** Agile Java Client does not support URL PXs.

---

## Before Building a URL-Based Process Extension

Please note the following requirements and constraints when building a URL-based process extension:

- URL-based process extensions cannot be initiated by a change in a workflow, because an Agile PLM client may not be active to trigger the change in status
- URL-based process extensions are not supported for Sourcing projects (`IProject`)

## Defining a URL-Based Process Extension

Following procedure defines a URL-based Process Extension.

**To define a URL-based process extension do as follows:**

1. Log into the Agile Java Client as an administrator.
2. Click the **Admin** tab.
3. Open the **Data Settings** folder.
4. Open the **Process Extensions** node.
5. In the Process Extension Library window, click the New button. The Add Process Extension dialog box appears.

   Type the following for different fields:

   - **Name** - Type the name of the process extension.
   - **Description** - Type a brief description of the process extension.
   - **Type** - Select the URL.
   - **Address** - Specify the address of a Web page. You must specify the complete URL, including the protocol. For example, to specify the Oracle Corporation website, you would type `http://www.oracle.com`, not `www.oracle.com`.
   - **Initiate From** - Select one or more locations from which the Web page can be initiated. Choose from the following options:
     – **Actions menu** - Enables you to select the Web page from the Actions menu of a properly configured class.
     – **Dashboard** - See Developing Dashboard Management Extensions.
     – **External report** - Use this to generate a report by accessing the Web page.
     – **Tools menu** - Use this to select the Web page from the Tools menu.

     If you specify that a process extension is initiated from the Actions menu, you can configure subclasses to use it. If you specify that the process extension to generate an external report, you can use the Agile Web Client to create the report. If you specify that the process extension is initiated from the Tools menu, which is always available in the Agile PLM client.
   - **Enabled** - Select **Yes** or **No**.
6. Click **OK** to save the new process extension.

## Setting Cookie Expiration Properties for URL Process Extensions

Navigating the URL PX generates a Cookie, and once created, the Cookie can be reused to intrude the PLM System. The Cookie Expiration feature is implemented to protect the PLM System from harmful intrusion by unauthorized user. This feature generates a unique token that is passed inside the `j_password` cookie which you can use only once. The following two attributes in the `agile.properties` file control the settings for these properties:

- `agile.sso.checkOneTimePXToken` - This property enables/disables the feature.
- `agile.sso.expirationTime` - This property controls the duration of the token and once the set duration expires, the token is invalidated and the Cookie is no longer reusable.

The default settings for these properties are (true) and 120 seconds respectively, and the PLM Administrator can reset both properties:

- `agile.sso.checkOneTimePXToken` = 'true'

- `agile.sso.expirationTime = 120`

## Passing Encoded Agile PLM Information to Other Applications

---

**Note:** Agile SDK does not support single sign-on through password protected external application servers. However, Agile Web Client can propagate encoded user credentials, and the SDK can reuse these credentials when your PX application uses the Agile SDK. To provide password protected access to an external Application Server, you must hard code the username and password to access the external servlet into your code.

---

If a URL-based process extension is initiated from an object's Actions menu, the object's composite key and class ID, as well as the current username, are encoded in the URL using the **GET** method. The client encodes the data as `ID=value` pairs and appends it to the end of the URL. Each ID is prefixed with "agile," as shown in the following example.

```
http://www.acoolwebsite.com/?agile.username=wangsh&agile.classId=10141&agile.100=1
000-02&agile.1014=A&agile.siteName=Taipe
```

---

**Note:** Unlike the Actions menu, there isn't an Agile PLM object associated with commands on the Tools menu. Consequently, if a URL-based process extension is initiated from the Tools menu, the URL is not augmented with encoded object data.

---

In addition to information encoded in the URL of a URL-based process extension, the encrypted username and its associated password are available from the `j_username` and `j_password` cookies, respectively, which are automatically passed to the target system if the following conditions are met:

- The user initiates a URL-based process extension from Agile Web Client.

---

**Note:** Your Web application must reside in the domain specified in the cookie.domain property of `agile.properties`. Otherwise, security cookies will not propagate.

---

- The target system is permitted to receive cookies.
- The target system is in the same domain as the Agile PLM system.

---

**Note:** If the target system is located outside the company firewall, it should be a secure Web server using SSL.

---

## Creating an Agile PLM Session from the Target System

Using authentication information contained in the HTTP request received from a URL-based process extension initiated from Agile Web Client, the target application can use the Agile API to create an `IAgileSession`. The Agile API client can then retrieve and configure the Agile PLM object referenced by the HTTP request.

When a user logs into Agile Web Client, the authentication process creates a pair of cookies (`j_username` and `j_password`) on the server computer that store the user's encrypted user name and password.

---

**Note:** These cookies will expire after the duration set by the Agile PLM Administrator.

---

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

---

**Note:** The Agile Java Client, unlike the Web client, does not create client-side cookies. Therefore, it does not support the single sign-on feature for process extensions.

---

Cookies are designed to be shared among computers within the same domain. For example, if during installation of Agile PLM, you configure the domain to be "`.agile.agilesoft.com`", then all computers ending with "`.agile.agilesoft.com`" can use the `j_username` and `j_password` cookies.

For more information, see `http://cookiescache.tripod.com/`.

The following example shows how to use the Agile API to extract cookie information from the HTTP servlet request and use that information to generate an IAgileSession. The value of the `AgileSessionFactory.PX_REQUEST` field, which is the key used to create the session, is set to be equal to the servlet request.

***Example 3–6 Creating IAgileSessions from servlet requests with PX_REQUEST***

```
private IAgileSession connect(HttpServletRequest request) throws ServletException
{
  HashMap params = new HashMap();
  params.put(AgileSessionFactory.PX_REQUEST, request);
  session = factory.createSession(params);
  return session;
}
```

If the target application is not servlet-based, there is another way to use the cookie information to create a session. Rather than using `AgileSessionFactory.PX_REQUEST`, you can use `AgileSessionFactory.PX_USERNAME` and AgileSessionFactory.`PX_PASSWORD` fields as keys for the HashMap. The values of these fields are the values of the `j_username` and `j_password` cookies, respectively.

***Example 3–7 Creating IAgileSessions with PX_USERNAME and PX_PASSWORD fields***

```
private IAgileSession connect(Cookie[] cookies) throws Exception {
factory = AgileSessionFactory.getInstance("http://agileserver/Agile");
  HashMap params = new HashMap();
  String username = null;
  String pwd = null;
  for (int i = 0; i < cookies.length; i++) {
    if (cookies[i].getName().equals("j_username"))
      username = cookies[i].getValue();
    else if (cookies[i].getName().equals("j_password"))
      pwd = cookies[i].getValue();
```

```
  }
  params.put(AgileSessionFactory.PX_USERNAME, username);
  params.put(AgileSessionFactory.PX_PASSWORD, pwd);
  session = factory.createSession(params);
  return session;
}
```

## Retrieving an Agile PLM Object from an HTTP Request

If you invoke a URL-based process extension from an object's Actions menu, you may want the target application to retrieve the Agile PLM object and modify it. The object's composite key and class ID are encoded in the URL using the GET method. To help the target application to retrieve the referenced `IAgileObject`, the Agile API provides an overloaded use of the `IAgileSession.IAgileObject`() method, as shown in the following example. The SDK extracts the object ID information from the request and uses it to retrieve the specified object.

***Example 3–8 Retrieving an Agile PLM object from an HTTP request***

```
private IAgileObject getAgileObject(HttpServletRequest request)
throws ServletException {
   IAgileObject obj = session.getObject(null, request);
   return obj;
}
```

If the target application is not servlet-based, you can use the normal `IAgileSession.getObject`() methods to retrieve the referenced object. For the params parameter of `getObject`(), specify a HashMap containing all required attributes for the object's class; the necessary attribute/value pairs are contained in the encoded URL. For a list of identifying attributes for each Agile PLM class, see the following section.

## Identifying Attributes for Agile PLM Classes

Each Agile PLM class has a different set of identifying attributes that could be passed as parameters in an encoded URL. For example, a Change object would pass its class ID and Cover Page.Number attribute. The following table lists the identifying attributes for each Agile PLM class.

| Class | Parameter | Description |
|---|---|---|
| Change | agile.classID | Class ID of selected object |
| Change | agile.1047 | Cover Page.Number |
| Customer | agile.classID | Class ID of selected object |
| Customer | agile.5110 | General Info.Customer Number |
| Commodity | agile.classID | Class ID of selected object |
| Commodity | agile.agile.2000004284 | Title Block.Name |
| Declaration | agile.classID | Class ID of selected object |
| Declaration | agile.agile.2000002615 | Title Block.Name |
| Discussion | agile.classID | Class ID of selected object |
| Discussion | agile.18417 | Cover Page.Number |
| File Folder | agile.classID | Class ID of selected object |
| File Folder | agile.6173 | Title Block.Number |

| Class | Parameter | Description |
|---|---|---|
| File Folder | agile.7951 | Title Block.Version |
| Item | agile.classID | Class ID of selected object |
| Item | agile.1001 | Title Block.Number |
| Item | agile.1014 | Title Block.Rev |
| Item | agile.siteName | Site name - If All is selected, this parameter is omitted |
| Manufacturer Part | agile.classID | Class ID of selected object |
| Manufacturer Part | agile.1647 | General Info.Manufacturer Name |
| Manufacturer Part | agile.1648 | General Info.Manufacturer Part Number |
| Manufacturer | agile.classID | Class ID of selected object |
| Manufacturer | agile.1754 | General Info.Manufacturer Name |
| Package | agile.classID | Class ID of selected object |
| Package | agile.3110 | Cover Page.Package Number |
| Price | agile.classID | Class ID of selected object |
| Price | agile.10355 | General Information.Number |
| Price | agile.10357 | General Information.Rev |
| Program | agile.classID | Class ID of selected object |
| Program | agile.18041 | General Info.Number |
| Sourcing Project | agile.classID | Class ID of selected object |
| Sourcing Project | agile.14824 | General Info.Number |
| PSR | agile.classID | Class ID of selected object |
| PSR | agile.4856 | Cover Page.Number |
| QCR | agile.classID | Class ID of selected object |
| QCR | agile.4029 | Cover Page.QCR Number |
| Report1 | agile.classID | Class ID of selected object |
| Report1 | agile.8071 | General Info.Name |
| RFQ | agile.classID | Class ID of selected object |
| RFQ | agile.13925 | CoverPage.RFQ Number |
| RFQ Response | agile.classID | Class ID of selected object |
| RFQ Response | agile.14472 | CoverPage.RFQ Number |
| RFQ Response | agile.14452 | CoverPage.SupplierName |
| Site | agile.classID | Class ID of selected object |
| Site | agile.11882 | General Info.Name |
| Specification | agile.classID | Class ID of selected object |
| Specification | agile.2000001969 | Title Block.Name |
| Substances | agile.classID | Class ID of selected object |

| Class | Parameter | Description |
|---|---|---|
| Substances | agile.2000001124 | Title Block.Name |
| Supplier | agile.classID | Class ID of selected object |
| Supplier | agile.17761 | General Info.Number |
| Transfer Order | agile.classID | Class ID of selected object |
| Transfer Order | agile.12673 | Cover Page.Transfer Order Number |
| User | agile.classID | Class ID of selected object |
| User | agile.11617 | General Info.Username |
| User Groups | agile.classID | Class ID of selected object |
| User Groups | agile.12077 | General Info.Name |

**Note:** Although the process extensions framework can encode Report information in a URL, Report objects are not supported by the Agile API. Therefore, you cannot use the Agile API to retrieve Report objects referenced in a URL.

## Creating an External Report

In Agile Web Client, you can connect to an external resource or URL to generate an external report. Before you create an external report, you must add the URL associated with the report to the Process Extension Library. For more information, see "Defining a URL-Based Process Extension" on page 3-14.

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

### To create an external report:

1. Log in to Agile Web Client.

**Note:** You cannot create external reports in Agile Java Client.

2. Choose **Create** > **Report** > **External**. The Report Creation Wizard appears.
3. Type the name of the report. Click **Next**.
4. Type the following General Information:
   - **Description** - Type a brief description of the report.
   - **Process Extension** - Select a process extension. The process extension you select is associated with a URL, such as the location of Web-based report engine.
   - **Folder** - Select the report's parent folder.
5. Click **Finish**.

## Deploying Process Extensions in Clustered Environments

If the Agile PLM installer was not run on a server in the application server cluster, the /*agile_home*/integration/sdk/extensions folder will not exist on that server. In this case, you must create the folder manually and copy any process extension JAR files into that folder as shown below.

**To manually create deployment folders for process extensions:**

1. Create the following folder on all application servers in the cluster (if it does not exist): /*agile_home*/`integration/sdk/extensions`
2. Load all process extension JAR files in the /*agile_home*/`integration/sdk/extensions` folder on each server in the cluster.

## Best Practices for Copying third Party JAR Files

If your PX code uses 3rd party JAR files such as axis.jar, you can copy them in the shared library and add them to classpath using the following procedures.

Use the following procedure to configure your shared libraries

**To configure shared libraries in Oracle Application Server:**

1. Place all third party JAR files in a folder, for example, `D:\commonLib`.
2. Stop the Agile server.
3. Navigate to *`OAS_HOME`*`\j2ee\home\application-deployments\Agile`.
4. Open orion-application.xml in a text editor.
5. Append a line to specify your library path:

   library path="*ABSOLUTE_PATH_TO_YOUR_LIBRARY*" /

   Specify the absolute path to the folder having those required JAR files.

   For example, `library path="D:\commonLib`“after these two lines:

   ```
   library path="../APP-INF/lib" /
   library path="../APP-INF/classes"/
   ```

6. Start the Agile Server.

Procedures in this section show how to deploy the following dependent JAR files:

- The dependent JAR file from 3rd party JAR files that do not dependent on Agile JAR files
- Any dependent JAR file from 3rd party JAR files that are dependent or not dependent on Agile JAR files

**To deploy the 3rd party JAR files on WebLogic:**

1. Stop all servers in the domain.
2. Copy the shared JAR file(s) to the lib subdirectory of the domain directory.

   For example, cp c:\3rdpartyjars\utility.jar AGILE_HOME\agileDomain\lib

> **Note:** The WebLogic Server must have read access to the lib directory during startup.

---

**Note:** The Administration Server does not automatically copy files in the lib directory to Managed Servers on remote machines. If you have Managed Servers that do not share the same physical domain directory with the Administration Server, you must manually copy the JAR file(s) to the `domain_name/lib` directory on the Managed Servers.

---

3. Start the Administration Server and all Managed Servers in the domain. WebLogic Server appends JAR files found in the lib directory to the system classpath. Multiple files are added in alphabetical order.

**To deploy the dependent JAR files from a WebLogic Admin console:**

1. In WLS Administration console, select **Deployments > Control > Install**.


2. Select the JAR file and then click


3. In Install Application Assistant accept the default settings and click **Finish**.


4. Make sure the library pointed to the Agile Server/cluster.
5. Create a file called `weblogic-application.xml` with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<wls:weblogic-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wls=http://www.bea.com/ns/weblogic/weblogic-application
xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-application
http://www.bea.com/ns/weblogic/weblogic-application.xsd
http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/j2ee_1_4.xsd">
<wls:library-ref>
<wls:library-name><Shared Library Jar file name></wls:library-name>
</wls:library-ref>
</wls:weblogic-application>
```

6. Place the `weblogic-application.xml` file in Agile_ Home\agiledomain\applications\application.ear\META-INF folder in the Administration server.
7. Restart the Administration and all the Managed servers.

### To deploy the properties files on WebLogic

1. Stop all servers in the domain.
2. Copy the properties file(s) to a directory as in *AGILE_HOME*`/pxConfig` in each server.

> **Note:** WebLogic Server must have read access to the directory during startup. The Administration Server does not automatically copy files in the directory to Managed Servers that are on remote machines. If you have Managed Servers that do not share the same physical domain directory that the Administration Server uses, you must manually copy the properties files to the AGILE_HOME/ `/pxConfig` directory on the Managed Servers.

3. Add the directory containing the properties file(s), for example, AGILE_HOME `/pxConfig` to the WebLogic CLASSPATH as shown below:

   a. Edit *AGILE_HOME*`/agileDomain/bin/setEnv.bat` in all Managed Servers

   b. Add *AGILE_HOME*/**pxConfig** folder to the CLASSPATH

```
set CLASSPATH=%JAVA_HOME%/lib/tools.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/ojdbc14.jar
```

```
setCLASSPATH=%CLASSPATH%;%WLS_HOME%/server/lib/
weblogic_sp.jar;%WLS_HOME%/server/lib/weblogic.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/agbase.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/wlsauth.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/crypto.jar +
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/xercesImpl.jar
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/jdom.jar;
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/log4j.jar;
set CLASSPATH=%CLASSPATH%;%LIB_HOME%/jobaccess.jar;
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/colt.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/jms.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/jndi.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/tibjms.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/oc4jclient.jar
set ClASSPATH=%CLASSPATH%;%LIB_HOME%/oc4j.jar
set CLASSPATH=%CLASSPATH%;../ldaplib/ldaputil.jar
set CLASSPATH=%CLASSPATH%;D:\Agile931b28/agileDomain/config
set CLASSPATH=%CLASSPATH%;AGILE_HOME/pxConfig
```

4. Start the Administration Server and all Managed Servers in the domain

## Process Extensions FAQs

This section answers common questions about process extensions.

**What are process extensions?**

Process extensions extend the functionality of Agile PLM clients through custom actions, external reports, and custom autonumbering, thus tailoring the system to fit a customer's business. You can use Process extensions to connect the Agile PLM server and Agile PLM users to external systems.

**What types of actions can you define with process extensions?**

Process extensions support two types of process extensions actions. They are custom autonumber sources and custom actions. Custom autonumber sources define the numbering sequences used by classes of objects. Custom actions are programs that can be run from Agile PLM clients.

A process extension can also be a reference to a URL. The URL can be a simple website or the location of a Web-based application.

**Can Process Extensions support asynchronous operations?**

Agile Process Extensions only support synchronous operations. If your Process Extension requires asynchronous behavior, you must modify your PX code to implement asynchronous solutions of your choice. For example, you can spawn a thread.

**Can I use Agile's Java API within a process extension program?**

Yes. You can use Agile's Java API and other external Java APIs. The only requirement is that you implement either the `ICustomAutoNumber` or the `ICustomAction` interface, depending on the type of extension.

**How do you initiate a process extension in an Agile PLM client?**

Custom actions can be triggered in the following ways:

- A change to a Workflow status.
- Selecting a custom action from the Tools menu.
- Selecting a custom action from the object's Actions menu.

- Selecting an external report that uses a custom action.
- Creating an object of a class that uses a custom autonumber source.

**Do process extensions have special security requirements?**

No. The process extensions stack sits on the Agile Application Server, so custom actions and custom autonumber sources operate within an environment where the user has already been authenticated and authorized.

**How are roles defined for custom actions?**

By default, a custom action uses the roles of the current user. However, to configure a custom action to have expanded privileges, you can specify the roles required for a custom action in the Process Extension Library of Agile Java Client. When you use a custom action in the Agile PLM client, roles that are specified for the custom action, override the privileges of the current user. Once the custom action is completed, the client reverts to the user's original privileges.

**How do I configure and deploy a process extension?**

Place the JAR file(s) for a process extension in the *agile_ home*/integration/sdk/extensions folder on the application server. Included with the JAR file(s) should be a file named com.agile.px.ICustomAutoNumber or com.agile.px.ICustomAction in the META-INF/services directory. The contents of these files are the fully qualified Java class n*ames, one* class per line, for a custom autonumber source or a custom action, respectively.

**After I deploy a process extension program on the application server, how do I enable it?**

Once process extensions have been deployed, you can configure them for use within Agile PLM clients. In Agile Java Client, you can add custom actions to the Process Extension Library and custom autonumbers to the Autonumbers node table.

**After I've deployed JAR file(s) for a custom action or custom autonumber source, do I need to update the application server classpath?**

No. The classpath is updated automatically by a special-purpose classloader. The classloader extends the application server classpath with any classes located in *agile_ home*/`integration/sdk/extensions` (or the location specified for the `sdk.extensions` property in the `agile.properties` file).

**How do you create a custom autonumber source?**

Create a Java class that implements the `ICustomAutoNumber` interface, a server-side API in the com.agile.px package. The code defines the autonumbering logic, for example, prefix, suffix, number of digits, and so on, and the persistence mechanism. The Agile PLM system gets the next number from the custom autonumber source by calling the `getAutoNumber()` method.

**How do you assign custom autonumber sources in Agile Java Client?**

In the Classes node, you assign autonumber sources to specific subclasses. In the AutoNumbers node, you can also assign subclasses to an autonumber source.

**How do you create a custom action?**

Create a Java class that implements the `ICustomAction` interface, a server-side API in the `com.agile.px` package. The code defines the custom action, whether to modify the current object, create an external report, integrate the Agile PLM client with an external system, or perform some other business logic. When an Agile PLM client initiates a custom action, the Agile PLM system calls the `doAction()` method.

**How do you associate custom actions with the Tools menu, the Actions menu, Status of a Workflow, and external reports?**

In Agile Java Client, open the Process Extensions node to add and configure custom actions. You can associate custom actions with a given Workflow status, the Tools menu and Actions menu for classes. A custom action associated with a Workflow status is initiated automatically when the Workflow assumes that status. A custom action appears on the Tools menu when it is invoked From property and is set to the Tools menu. A custom action appears on the Actions menu for an object when you add it to the Process Extensions tab of the subclass. As indicated earlier in this Guide, external reports can only use URL PXs. As such, a custom action associated with an external report is triggered automatically when that report is executed.

**In what order do process extensions appear on the Tools menu or Actions menu of Agile PLM clients?**

If you add process extensions to either the Tools menu or an object's Actions menu, they are listed after standard menu commands in the order they were created. You cannot reorder or otherwise manage commands on the Tools menu or Actions menu.

**What is the inheritance model of custom actions that are assigned to classes?**

Custom actions can be defined at the base class level, the class level, or the subclass level. A custom action defined at the base class level is available to all classes and subclasses beneath the base class. A custom action defined at the subclass level is available only to that subclass.

**Where do I put PX and WSX configuration property files**?

After deployment changes in Agile PLM Release 9.2.2.2, the `agileDomain\config` directory is no longer in the classpath. You can put PX and WSX property files in this directory: `\oas\j2ee\home\applications\Agile\APP-INF\classes\`.

# 4

# Developing Dashboard Management Extensions

This chapter includes the following:

- About Dashboard Management Extensions
- Developing Custom Chart Dashboard Management Extensions
- Developing Custom Table Dashboard Management Extensions

## About Dashboard Management Extensions

Similar to Process Extensions, Dashboard Management Extensions (DX) extend the functionality of the Agile PLM system. The Extensions support the following formats to access and display PLM data on the Agile PLM Dashboard:

- ChartDataModel for charts
- Collections for tables

When data is defined using these formats, Agile servers can interpret and process this data, and Agile Java Client users with administrator privileges can define Dashboard Tabs and display them in one of the following views or layouts:

- Chart
- Table
- Custom (URL)

The SDK provides the API's that enable connecting the Agile PLM server to internal Agile databases to get the required content, format it as required by the DXs, and display the data in the Agile PLM Dashboard. Similarly, you can use other Java API's such as JDBC to connect to external databases for content.

Briefly, DXs provide the data, Dashboard Tabs and the formats to display the data (tables, charts, and URLs) are configured in Agile Java Client. Finally, Agile PLM users with proper privileges can view the Dashboard Tab and displays in Agile Web Client.

This chapter provides both background information and procedures to develop these methods.

### Roles and Privileges in Dashboard Management Extensions

You must set the Dashboard Tab View privilege in **Admin**>**UserSettings**>**Privileges** so that PLM users can view the Tabs and the related data in Web Client. In addition, Dashboard Tabs are controlled by privileges, and Agile PLM users must have the necessary roles and privileges to view the data on the Tab. Procedures to configure the views and assign privileges are fully documented in *Agile PLM Administrator Guide*.

# Developing Custom Chart Dashboard Management Extensions

The ICustomChart interface enables creating the necessary DXs that will display the required data in chart formats. This interface exposes the method which returns an instance of the public `ChartDataModel getChart(IAgileSession session, Map params)`

> **Note:** Implementations of this interface must have no-arg constructors and they must be reentrant.

## Understanding ChartDataModel and ChartDataSet

The ChartDataModel class organizes the input data in a chart format. It is a concrete class that is exposed to the DXs and It contains one or more `ChartDataSet(s)` that you need to construct the chart.

A ChartDataSet is another concrete class that is exposed to the DXs. It contains the data required to plot a chart. For example, X-axis and Y-axis values and labels. The ChartDataModel is a placeholder for all the data sets.

> **Note:** The ChartDataModel and ChartDataSet classes are exposed in com.agile.px package.

## Defining a Custom Chart DX Data Source

As indicated above, chart DXs display the data in chart formats. The code in Example 4–1, "Defining a DX to display data in a chart format" uses ICustomChart and the exposed classes (ChartDataModel and ChartDataSet) to display the differences between morning and evening temperatures for every day of the week, using predefined input data in a chart format.

***Example 4–1 Defining a DX to display data in a chart format***

```
package dashboard.chart;

public ChartDataModel getChart(IAgileSession session,Map params) throws Exception{
import java.util.Map;
import com.agile.api.IAgileSession;
import com.agile.px.ICustomChart;
import com.agile.px.ChartDataModel;
import com.agile.px.ChartDataSet;

/**
 * A Sample Dashboard DX for Charts with predefined data.
 * This Example displays a comparison chart between
 * Morning and Evening Temperatures for each day of the
 * week with predefined data.
*/
public class TemperatureComparisionChart implements ICustomChart(

/**
 * Returns custom ChartDataModel. ChartDataModel
 * is a placeholder to hold all the
```

```
 * ChartDataSet(s) and any other relevant information related to the charts.
 * @param session current user session.
 * @param params
 * @return com.agile.px.ChartDataModel
*/
public ChartDataModel getChart(IAgileSession session,Map params) throws
Exception{

// Create a ChartDataModel
 ChartDataModel chartDataModel = new ChartDataModel("Temperatures");

// Create a ChartDataSet for Morning and Evening Temperatures
 ChartDataSet chartdataSet[] = new ChartDataSet[2];

// Create a ChartDataSet for Morning Temperatures
 chartdataSet[0] = new ChartDataSet("Morning Temperatures",7);

// Fill in the Morning Temperatures
 double[] morTempValues = {10, 8, 12, 19, 10, 14, 13};
 chartdataSet[0].setValues(morTempValues); // or setYValues that can be used
instead

// Set the Labels
 String[] labels =
 {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"};
  chartdataSet[0].setLabels(labels);

// Create a ChartDataSet for Evening Temperatures
 chartdataSet[1] =
 new ChartDataSet("Evening Temperatures",7);

// Fill in the Evening Temperatures
 double[] eveTempValues = {16, 12, 20, 15, 18, 24, 26};
 chartdataSet[1].setValues(eveTempValues);
 chartdataSet[1].setLabels(labels);

// Set the ChartDataSets in the Chart Model
 chartDataModel.setDataSets(chartdataSet);

return chartDataModel;
}
}
```

## Packaging and Deploying a Custom Chart DX Source

After developing the necessary classes for a new Chart, package and deploy them using the following procedure.

**To package and deploy a Chart DX source:**

1. Use your Java development environment or the Java Archive tool (or JAR tool), to create one or more JAR files for the custom action. Make sure the JAR file(s) includes a META-INF/services directory that contains the file com.agile.px.ICustomChart. This is a text file that lists the fully qualified Java class names, one class per line, for the custom action.

You can include multiple charts in one package. For example, com.agile.px.ICustomChart can resemble this:

```
dashboard.chart.TemperatureComparisionChart
dashboard.chart.AgileObjectsCountChart
dashboard.chart.ActualVsBudgetedLaborCostChart
```

---

**Note:** Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file name are either all uppercase or all lowercase characters. Otherwise, the custom action will fail to deploy.

---

2. Place the JAR file(s) in the `agile_home/integration/sdk/extensions` folder on the same computer where the Agile Application Server is installed.

---

**Note:** If you have several application servers in a clustered environment, you must deploy the Dashboard Extension files on each server in the cluster.

---

## Configuring Chart DXs in Java Client

In Agile Java Client, you can define Chart data sources in the Admin module. To configure the Agile PLM system settings, you must have an administrator account. This is briefly documented in the sequel below. For more information, refer to the *Agile PLM Administrator Guide*.

The data that you provide for a DX, regardless of the layout, is viewed in a Dashboard Management Tab. Because you cannot define a new Table in the Out of Box Tabs such as Executive or Financial, you must define a new Tab and then a Table within the Tab to configure a DX.

**To add an optional Dashboard Management tab:**

1. In Java Client, select **Admin** > **Systems Settings** > **Dashboard Management** and click the **New Dashboard Tab** icon in Dashboard Management.
2. In the Create Dashboard Tab dialog, complete the name (For example, call it **Dashboard Extensions**) and description fields, set the **Visible** field to **Yes**, and then click **OK**. Dashboard Extensions appears as an entry in the Dashboard Management.
3. Click the **Order Tabs for Dashboard** icon to reorder the tabs as required in Java Client.

### To configure a Chart type table in the optional tab:

1. Define a new tab, for example, Dashboard Extensions as shown above.
2. In the new tab (Dashboard Extensions), click the Tables icon. The Dashboard Management - Dashboard Extensions page appears.
3. In this page, click the **New Dashboard Table** icon to open the Create Dashboard Table dialog and define the new table.
4. Select **Chart from the** View List Type drop-down list. The following fields appear:

*Table 4–1 Chart type table data fields*

| Dashboard Table | Description/Purpose | Possible Settings |
|---|---|---|
| Name | Type the name of the table | String |
| API Name | The Name typed above, converted to *CamelCase* naming convention by Agile PLM | String |
| Description | Type the description of the table | String (optional) |
| View List Type | Lists the types of table. Select **Chart** (when you select Chart, additional options are displayed). | Chart, Table, Custom, Advance Search |
| Dashboard Extension | Lists all process extensions created for chart type list. Select the chart process extension you want. | Empty cell |
| Visible | To enable viewing in Web Client | Yes/No |
| Chart Type | Select the type of chart you want displayed | Area, Bar, Line, Pie, Polar, Scatter, Stacked Area, Stacked Bar, Table |
| X axis | Type the X axis label | (optional) |
| Y axis | Type the Y axis label | (optional) |
| Show Legend | To display the chart legend on screen | Yes/ No |
| Legend Position | Select the position where the Legend should be displayed | Bottom, default, left, right, top |
| 3D Style | To view the graph in 3D | Yes/ No |

### Displaying Optional Tabs in Agile Web Client

You can display the new optional Tab in Agile Web Client and users satisfying the privileges requirement can view the tabs and the corresponding data. You can find the necessary procedures in *Agile PLM Administrator Guide*.

## Developing Custom Table Dashboard Management Extensions

The `ICustomTable` interface is defined to create DXs to display the required data in tabular formats. This interface exposes the `getTable(IAgileSession` session, Map params) method which returns an instance of the Collection class.

```
public Collection getTable(IAgileSession session, Map params);
import java.util.*;
```

---

**Note:** Implementations of this interface must have a no-arg constructor and they must be reentrant.

---

## Understanding Collections and CustomTableConstants

The Tabular Data in a DX is a "collection" of Java `HashMaps`. Each Map key represents an attribute in the Table View and the Map represents a row in the table.

The property "Attribute" of a column in View defines the mapping between the data model and the Table View. The value of this property corresponds to the key of a HashMap entry.

- **HashMap keys** - For HashMap entries, an attribute is defined in the Table view. For example, a HashMap entry with "name" as its key value, the property "Attribute" of this attribute will have the value "name." The `get('name')` method will provide the display data for this attribute.
- **Link, Image, Money, Text, Date and Numeric Data** - These data types are supported in Tabular DX formats and return objects with the following properties.
  - **Text** - Date, and Numeric Data types do not require any additional properties.
  - **Link** - A valid URL (as String) serves as the target and label for the display. The properties expected for a link data type are the same for the internal and external links. The DX users resolve the URL for internal links and add them to the URL property. The DX users can specify the target property as "Right Pane" for internal links. By default, the links will be targeted to a new window.
  - **Image** - Images are expected to return an image URL (as String) and label to be displayed as a tool tip on the Image.
  - **Money** - Currency code (String) and Value (Number) are necessary for Money Data types.

**Note:** Keys that support the Link, Money and Image data properties are provided as constants in the class CustomTableConstants. A constant SERVER_URL is provided in this class. You can use it get the Server URL in the DX's from the params.

## Defining a Custom Table DX Data Source

The sample Dashboard DX in the following example creates a collection of rows with predefined data for display in the Dashboard. Each row is a Java Map object with key-value pairs that correspond to each column in the table. This value appears in each cell of the column in the table. The key is the mapping Attribute name in the View. When creating new Attributes (columns) in the View, it is necessary to supply this key in the Attribute field. Attribute Names and the corresponding Data type for this DX are as follows:

| Attribute | Corresponding Data Type |
|---|---|
| `myString` | Text |
| `myExternalLink` | Link |
| `myDate` | Date |
| `myMoney` | Money |
| `myNumber` | Numeric |
| `myImage` | Image |

***Example 4–2 Defining a Dashboard extension to display data in tabular format***

```
package dashboard.table;
import com.agile.api.IAgileSession;
import com.agile.px.ICustomTable;
import com.agile.px.CustomTableConstants;

/** This Sample Dashboard DX creates a collection
* of rows with predefined data
* in the format to be displayed in the Dashboard.
* Each row is a Java Map object which has
* key-value pairs corresponding to each column in the Table.
* The value is displayed in each Cell of the
* column in the table. The key is the
* mapping Attribute name in the View.
* While creating new Attributes (Columns) in the View,
* you must supply this key in the Attribute field.
* The corresponding Attribute Names and
* Data type for this DX are listed below.
*  <table border="1">
*  <tr><td><b>Attribute</b> </td><td><b>Data   Tye<b></td></tr>
*  <tr><td>myString        </td><td>Text      </td></tr>
*  <tr><td>myExternalLink   </td><td>Link      </td></tr>
*  <tr><td>myDate          </td><td>Date      </td></tr>
*  <tr><td>myMoney         </td><td>Money     </td></tr>
*  <tr><td>myNumber        </td><td>Numeric   </td></tr>
*  <tr><td>myImage         </td><td>Image     </td></tr>
*  </table>
*
*/

public class DashboardSampleTable  implements ICustomTable {
 /**
* Returns custom table data in form of collection of rows.
* Row is assumed to be a java Map object.
* @param session the user session
* @param params
* @return : java.util.Collection
 */
public Collection getTable (IAgileSession session,Map params) throws Exception{
String serverUrl =
   (String)params.get(CustomTableConstants.SERVER_URL);

String baseUrl =
  serverUrl.substring(0,serverUrl.lastIndexOf('/'));
  ArrayList result = new ArrayList();

// 1st Row Entry
   HashMap row1 = new HashMap();

// For Text type
   row1.put("myString","Manoj Yeturu");

// For Numeric type
   row1.put("myNumber",new Double(10000));

// For Date Type
   row1.put("myDate",new Date());

// For Image Type. The url for image and label (for tooltip) properties are set
   HashMap hm1Image = new HashMap();
```

```
   hm1Image.put(CustomTableConstants.URL,baseUrl+"/images/action_noshad.gif");

// Tool Tip
   hm1Image.put(CustomTableConstants.LABEL,"Action_Noshad");
   row1.put("myImage",hm1Image);

// For Money Type. The Currency and value properties are set
   HashMap hm1Money = new HashMap();
   hm1Money.put(CustomTableConstants.MONEY_CURRENCY_CODE,"USD");
   hm1Money.put(CustomTableConstants.MONEY_VALUE,new Integer(3000));
   row1.put("myMoney",hm1Money);

// For External Link, url, label (display string) and target
// (Rightpane,_new etc) are set
   HashMap externalLink1 = new HashMap();
   externalLink1.put(CustomTableConstants.URL,"http://www.agile.com");
   externalLink1.put(CustomTableConstants.LABEL,"Agile");
   externalLink1.put(CustomTableConstants.TARGET,"_new");
   row1.put("myExternalLink",externalLink1);
   result.add(row1);

// 2nd Row Entry
   HashMap row2 = new HashMap()

// For Text type
   row2.put("myString","Venkat Tipparam");

// For Numeric type
   row2.put("myNumber",new Double(50000));

// For Date Type
   row2.put("myDate",(new Date()));

// For Image Type
   HashMap hm2Image = new HashMap();
   hm2Image.put(CustomTableConstants.URL,baseUrl + "/images/addressdown.gif");

// Tool Tip
   hm2Image.put(CustomTableConstants.LABEL,"Addressdown");
   row2.put("myImage",hm2Image);

// For Money Type
   HashMap hm2Money = new HashMap();
   hm2Money.put(CustomTableConstants.MONEY_CURRENCY_CODE,"INR");
   hm2Money.put(CustomTableConstants.MONEY_VALUE,new Integer(4000));
   row2.put("myMoney",hm2Money);

// For External Link
   HashMap externalLink2 = new HashMap();
   externalLink2.put(CustomTableConstants.URL,
      "http://www.agile.com/services/support.asp");
   externalLink2.put(CustomTableConstants.LABEL,"Supprt");
   externalLink2.put(CustomTableConstants.TARGET,"_new");
   row2.put("myExternalLink",externalLink2);
   result.add(row2);
   return result;
   }
}
```

### Configuring the Link Data Type for Objects Created in Custom Table DXs

The Link data type is defined as follows in "Understanding Collections and CustomTableConstants" on page 4-6.

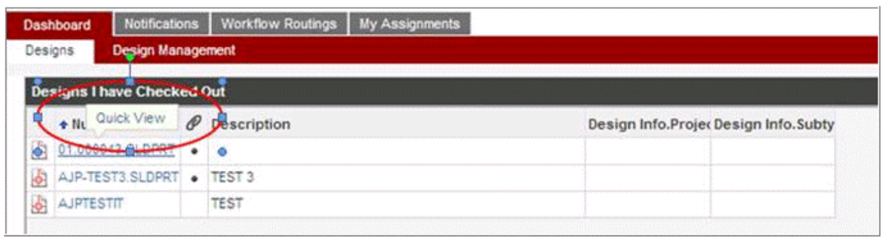
- A valid URL string that serves as the target and label for the display

And

- DX users can specify the target property as "Right Pane" for internal links which are targeted to a new window by default

When Web Client users retrieve an Agile Object in a Dashboard table by invoking Advanced Search, they automatically get the Number with a Link and Quick View as shown in Figure 4–1 below.

***Figure 4–1 The Quick View option in a Dashboard***




The SDK supports configuring the URL in the Link data type to display and open the selected object in the Right Pane for Agile Objects that are generated through user-defined Dashboard table DXs with ICustomeTable. To perform these configurations, you must first invoke the Advanced Search function to retrieve the Object.

#### *Invoking Advanced Search in a Custom Table DX Data Source*

The following code snippet returns the specified Agile Object and enables the Quick View pop-up for that Object.

***Example 4–3 Invoking Advanced Search in a Custom Table DX Data Source***

```
public class DashboardTableContainsQuickView implements ICustomTable{
   public Collection getTable(IAgileSession session, Map params){
      try
      {
         IQuery query = null;
         query = (IQuery)session.createObject
            (-5, ProgramConstants.CLASS_ACTIVITIES_CLASS);
         query.setCaseSensitive(false);
         query.setCriteria("[General Info.Name] contains 'Dashboard'");
         Iterator iter = query.execute().iterator();
         ArrayList result = new ArrayList();
         while (iter.hasNext())
         {
         IRow row = (IRow)iter.next();
         HashMap rowMap = new HashMap();
         IProgram program = (IProgram)row.getReferent();
         program.getId()String number = program.getName();
         rowMap.put("number", number);
         String name = (String)program.getCell
            (ProgramConstants.ATT_GENERAL_INFO_NAME).getValue();
```

```
          HashMap internalLinkMap = new HashMap();
          internalLinkMap.put("label", name);
          internalLinkMap.put("target", "RightPane");
          rowMap.put("name", internalLinkMap);
          result.add(rowMap);
          }
          return result;
       }
       catch (Exception e)
       {
          e.printStackTrace();
       }return null;
    }
}
```

### Enabling Quick View in a Custom Table DX Data Source

The following example enables Quick View for the Object returned by the search function. Code in the shaded area enables Quick View. To display Quick View, you must specify the correct `Object Class ID(getClassId()` and `Object ID (getObjectId)` attributes. In this example, `ActivityHandler` is hard coded, because this is the object type that the user wants to open. For other types of object, for example a Part object, you can use `itemhandler`. The output of this DX is displayed in Figure 4–2, "Sample output generated by the DX".

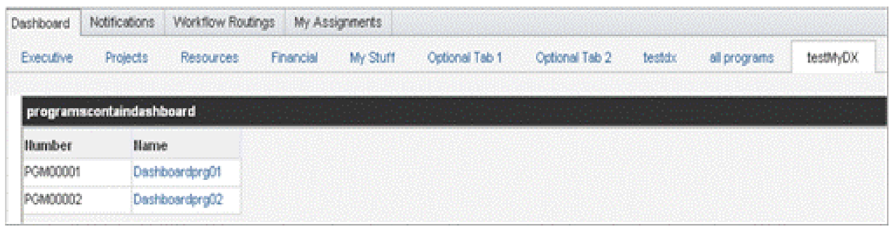
***Example 4–4   Enabling Quick View in a Custom Table DX Data Source***

```
public class DashboardTableContainsQuickView implements ICustomTable{
   public Collection getTable(IAgileSession session, Map params)
      {
        try
         {
           IQuery query = null;
           query = (IQuery)session.createObject
              (-5, ProgramConstants.CLASS_ACTIVITIES_CLASS);
           query.setCaseSensitive(false);
           query.setCriteria("[General Info.Name] contains 'Dashboard'");
           Iterator iter = query.execute().iterator();
           ArrayList result = new ArrayList();
           while (iter.hasNext())
           {
              IRow row = (IRow)iter.next();
              HashMap rowMap = new HashMap();
              IProgram program = (IProgram)row.getReferent();
              program.getId();
              String number = program.getName();
              rowMap.put("number", number);
              String name = (String)program.getCell
                (ProgramConstants.ATT_GENERAL_INFO_NAME).getValue();
              HashMap internalLinkMap = new HashMap();

            // Generate Quick View - Object ID and Class ID are necessary
              int objClsId = ((Integer)row.getClassId()).intValue();
              int objId = ((Integer)program.getObjectId()).intValue();
              int objVersion = ((Integer)program.getObjectVersion()).intValue();
              String urlStr =
                "javascript:displayObject('ActivityHandler',
                '" + objClsId + "', '" + objId + "', '0');
                \"onmouseover=\"showQuickViewLink(event, this)\"
                onmouseout=\"cancelQuickViewTimer(this);
```

```
                \" infourl=\"showObjectInfo('" + objClsId + "',
                '" + objId + "', '', '', '', 'true', '',
                '"+ objVersion +"');\"\");";
                internalLinkMap.put("url", urlStr);
              internalLinkMap.put("label", name);
              internalLinkMap.put("target", "RightPane");
              rowMap.put("name", internalLinkMap);
              result.add(rowMap);
         }
         return result;
}
```

***Figure 4–2 Sample output generated by the DX***




## Displaying Quick View with Mouseover

To display Quick View when moving the mouse over the Object ID, set the URL parameter as shown in the following code snippet. Make sure you are using correct values for `classid` and `objectid` parameters.

***Example 4–5 URL parameters to display Quick View***

```
HashMap internalLinkMap = new HashMap();
   internalLinkMap.put("url","javascript:displayDesignObject
      ('AttachmentHandler', '2000008297','21335435',
         '0', 'RightPane', '0', '', '4');"
   onmouseover="showQuickViewLink(event, this)" onmouseout=
      "cancelQuickViewTimer(this);" infourl=
         "showObjectInfo('2000008297', '21335435', '1', '6173', 'R0', 'true',
            'DASHBOARD_DXTABLE_2488764_GRID', '4');"");
   internalLinkMap.put("label", name);
   internalLinkMap.put("target", "RightPane");
   rowMap.put("name", internalLinkMap);
```

---

**Note:** In this example, 2000008297 is the object's `classId`, 21335435 is the object's ID, and so on. These values were defined in the code snippet in "Enabling Quick View in a Custom Table DX Data Source" on page 4-10.

---

## Opening the Selected Object in the Right Pane

By default, when you use the Link option to open the selected object, the object is displayed in a new Window. The following syntax opens the referenced object in the right pane of the same window for an object retrieved using Advanced Search shown above.

---

**Note:** You must specify the correct Class ID and object ID attributes. In this case,18022 and 4555 respectively. Also, this link does not enable Quick View. It is a quick way to put a link in your code. The Quick View option creates a link that enables the Quick View popup.

---

***Example 4–6 Opening the selected object in the right pane***

```
String name =
(String)program.getCell(ProgramConstants.ATT_GENERAL_INFO_NAME)get.Value();
   HashMap internalLinkMap = newHashMap();
   internalLinkMap.put("url","/web/PCMServlet?module=ActivityHandler&opcode=
      displayObject&classid=18022&objid=4555");
   internalLinkMap.put("label",name);
   internalLinkMap.put("target","RightPane");
   rowMap.put("name",internalLinkMap);
```

## Packaging and Deploying a Custom Table DX Source

After developing the required classes for a new Table, package and deploy them as shown below.

### To package and deploy a Table DX source:

1. Use your Java development environment, or the Java Archive tool (or JAR tool), to create one or more JAR files for the custom action. Make sure the JAR file(s) includes a META-INF/services directory that contains a file named com.agile.px.ICustomTable. This is a text file that lists the fully qualified Java class names, one class per line, for the custom action.

   You can include multiple charts in one package. For example, the com.agile.px.ICustomTable file could look like this:

   ```
   dashboard.chart.ActualVsBudgetedLaborCostTable
   dashboard.chart.DashboardSampleTable
   dashboard.chart.QueryDashboardPrograms
   ```

   ---

   **Note:** Paths within a JAR file are case-sensitive. Therefore, make sure the META-INF folder contained within the JAR file has a name with all uppercase or all lowercase characters. Otherwise, the custom action is not deployed.

   ---

2. Place the JAR file(s) in the `<agile_home>/integration/sdk/extensions` folder on the same computer where the Agile Application Server is installed.

   ---

   **Note:** If you have several application servers in a clustered environment, you must deploy the Dashboard Extension files on each server in the cluster.

   ---

## Configuring Table DXs in Java Client

Similar to Chart type DXs, you can use an existing Dashboard Management tab, or create your own optional tab to add your Table DXs.

### To Add a Table to a Tab:

1. Define a new tab. For example, Dashboard Extensions as shown above.
2. In the new tab (Dashboard Extensions), click the Tables icon. The Dashboard Management - Dashboard Extensions page appears.
3. In this page, click the **New Dashboard Table** icon to open the Create Dashboard Table dialog and define the new table.
4. Select **Table** from the **View List Type** drop-down list. The Create Dashboard Table dialog displaying the necessary fields appears.
5. Complete these fields and then click **OK**. The new table is created.

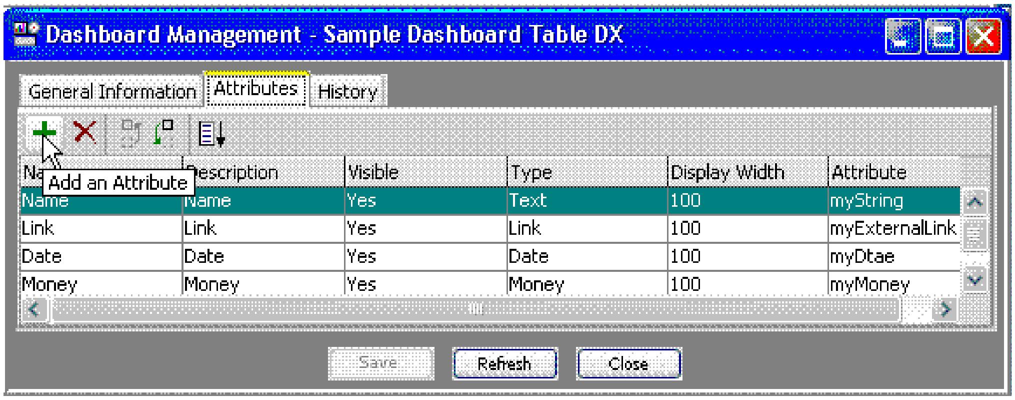
| Dashboard Table | Description/Purpose | Possible Settings |
|---|---|---|
| Name | Type a name for the table | String |
| API Name | The Name field typed above, converted to *CamelCase* naming convention by AgilePLM | String |
| Description | Type a description of the table | String |
| View List Type | Lists the types of table. Select **Table** | Chart, Table, Custom, Advance Search |
| Dashboard Extension | Lists all the process extensions created for the type of Table in view | These are the attributes that were defined in the "Packaging and Deploying a Custom Chart DX Source" on page 4-3. |
| Variable | To enable in Web client | Yes/No |

### To Add Data to Tables:

1. Double-click the new table that you created in "To Add a Table to a Tab:" on page 4-13.
2. Click **Attributes** and then the **Add an Attribute** icon to create the new attribute.

> **Note:** Agile currently supports Text, Numeric, Image, Date, Money, and Link type data as table attributes. They are listed and defined in Packaging and Deploying a Custom Table DX Source

***Figure 4–3 A Dashboard Table DX***




3. In the General Information tab, map the **Attribute** field to the attribute name in the DX.

> **Note:** You are now defining the attributes (columns) that will appear in the table on the Dashboard Tab. The property "Attribute" defines the mapping between the data model and the view. For example, if the attribute name in the DX is `myStringy` and the selected attribute type is Text, map the attribute field whose attribute name is `myStringy`.

4. For more information, refer to the *Agile PLM Administrator Guide*.

## Defining Custom (URL) Extensions

A Dashboard PX of the type URL is configured to initiate from Dashboard Management. When defining Custom extensions, simply select Custom as the table type. See "To Add a URL to a Tab:" below. No other mapping is required for Dashboard PX's of type URL.

> **Note:** URL Process Extensions are defined in the Process Extensions Library to initiate from Dashboard Management.

### To Add a URL to a Tab:

1. Define a new tab, for example, Dashboard Extensions as shown above.
2. In the new tab (Dashboard Extensions), click the Tables icon. The Dashboard Management - Dashboard Extensions page appears.
3. In this page, click the **New Dashboard Table** icon to open the Create Dashboard Table dialog and define the new table.
4. Select **Custom from View List Type** drop-down list. The **Create Dashboard Table** dialog displaying fields listed in the following appears.

5. Complete the **Create Dashboard Table** dialog fields and then click **OK**.

| Dashboard Table | Description | Possible Settings |
|---|---|---|
| Name | Type a name for the URL | String |
| Description | Type a description | String |
| View List Type | Lists types of table. Select **Custom** | Chart, Table, Custom, Advance Search |
| Dashboard Extension | Lists all process extensions created for Custom type list. | Employee Portal, Yahoo, Google, Process Extension_ URLs |
| Visible | To enable in Web Client | Yes/No |

# 5

# Working with Agile PLM Events and Event Context Objects

This chapter includes the following:

- Understanding Agile PLM Events and Event Framework
- Key Components of an Agile PLM Event
- Working with Event Context Objects
- Working with Event Information and Event Script Objects

## Understanding Agile PLM Events and Event Framework

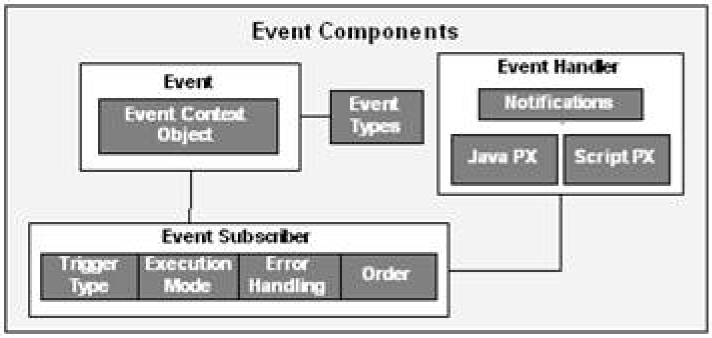
In Agile PLM, Events act as trigger points for generating an automation action within the PLM application. Every Event is generated from a source within Agile PLM applications. The source can be a business action triggered by a user, a UI action, or system initiated source such as a timer. These sources can signal other PLM modules that something (an event) has occurred within the application and it may require an action. The required action can be taken by a user or the PLM module. Event-driven applications greatly facilitate the management of event-based integrations and complex event analysis in real time modes.

In the Agile SDK environment, Event framework extends the PX framework which facilitated automating and extending the Agile PLM Applications. Event framework provides a flexible environment for rapid development and deployment of event-driven applications. To support this environment, Event framework provides a comprehensive set of data parameters to create, configure, and execute different types of Agile PLM Events.

## Key Components of an Agile PLM Event

Event framework empowers users to configure Events and Event subscribers. The basic components of Event framework are shown in the following illustration and described in the following paragraphs.

***Figure 5–1 Event Components***




## Event Types

In Agile PLM, *Event type* refers to a particular action, for example, *Create Object*, *Delete Object*, *Audit for Workflow*. Agile PLM provides a list of pre-defined Event types for which an event can occur.

For each Event type, depending on its Handler type, there is a corresponding Java and Script interface for its Event Context object. For example, for *Create Object*, there is the ICreateEventInfo Java interface and `ICreateScriptObj` Script interface respectively.

Event type is selected from the drop-down list in Java Client's *Create Event* dialog when the Event is created. To access this dialog, refer to *Agile PLM Administrator Guide*. Default Event types and their descriptions is shown in the figure below.

***Figure 5–2 Default Event types and descriptions***

| Name | Description |
|---|---|
| Approve for Workflow | Workflow, Approve |
| Audit for Workflow | Workflow, Audit |
| Cancel Check Out Files | Files, Cancel Check Out |
| Change Approvers or Observers for Workflow | Workflow, Change Approvers or Observers |
| Change Status for Sourcing Object | Sourcing Object, Change Status |
| Change Status for Workflow | Workflow, Change Status |
| Check In Files | Files, Check In |
| Check Out Files | Files, Check Out |
| Comment for Workflow | Workflow, Comment |
| Compliance Rollup on Object | Object, Compliance Rollup |
| Create Object | Object, Create |
| Delete Object | Object, Delete |
| Escalation for Workflow | Workflow, Escalation |
| Export Object | Object, Export |
| Extend Actions Menu | Actions Menu, Extend |
| Extend Tools Menu | Tools Menu, Extend |
| Get File | File, Get |
| Incorporate Item | Item, Incorporate |
| Promotion Failure for Workflow | Workflow, Promotion Failure |
| Purge Version Files | Files, Purge Version |
| Reject for Workflow | Workflow, Reject |
| Reminder for Workflow | Workflow, Reminder |
| Save As Object | Object, Save As |
| Scheduled Event | Event, Scheduled |
| Transfer Authority | Authority, Transfer |
| Unincorporate Item | Item, Unincorporate |
| Update Relationship | Relationship, Update |
| Update Table | Table, Update |
| Update Title Block | Title Block, Update |

**Note:** You can neither create, or delete these Event types.

## Event Handler and Handler Types

An Event Handler represents a custom action that is called when the Event is raised. They extend the function of an action taken by a user, interface, or the system when the Event is triggered. Information about the Event is passed from the Event to the Handler by the *Event Context object*. Handlers are invoked by Event Triggers.

The SDK and Event framework support the following Event Handler types:

- **Java PX -** Java PXs are Java process extensions that implement the `IEventAction` interface in `com.agile.px` package and trigger the compiled Java code. See "Event Information Objects" on page 5-9.
- **Script PX - Script PXs are Script process extensions based on Groovy script language**.

The Groovy script code is directly stored in Agile PLM databases. Event Script objects (Event Script PX handlers) that you develop are text files that are deployed on the Agile PLM server using the Event Management Node in Java Client. For information on submitting the text files, see "Working with Agile PLM Administrator" on page 5-36. For information about Groovy implementation in Event framework and Event Script PX development process, see Appendix C, "Groovy Implementation in Event Framework."

- **Notification** - Agile PLM can send notifications to users, either when the user is required to act, or to notify the user of actions that have occurred. Notifications can be triggered from the SDK and Script. Event Notifications are addressed in the Agile PLM Administrator Guide. To use the SDK to programmatically send Notifications, refer to “Sending Notifications with Agile SDK” in *SDK Developer Guide - Using Agile APIs*.

**Note:** SDK developers use the Java PX and Script PX to extend the capabilities of Agile PLM. The Agile PLM Administrator uses the Notification templates to define and configure Notifications for action or information.

## Event Subscribers

An Event Subscriber links an Event Handler to a specific Event. Thus, when a particular event occurs, Event Handlers associated with the Event through Event Subscribers are initiated in the order requested by the configuration. The Handler action includes invoking a Java Process Extension, a Script Process Extension, or sending Notifications.

### Event Trigger and Trigger Types

Event Trigger determines when an Event Extension is raised. The Event framework provides the hooks to automate actions when an incident happens. For example, when a change of status occurs, a CEO approval is required, the trigger signals the occurrence of an action which will subsequently notify all Event subscribers for that Event. Most Agile PLM Events are associated with business actions. Examples are:

- **Create Object Event** - associated with the object creation action
- **Change Status for Workflow Event** - associated with the Workflow status change action

### *Event Trigger Types*

- **Pre** - This trigger type signals a point before the occurrence of an action. The Pre trigger is commonly used for events that require data validation or other preparations for the upcoming action. Event Handlers are invoked synchronously from the **Pre** trigger point.
- **Post** - This trigger type signals a point immediately after the action's changes are committed to the database. The post trigger is commonly used for Events that perform auditing tasks, notifications, and integration tasks with external systems related to the completed action. Event Handlers are invoked synchronously or asynchronously from the **Post** trigger point.

## Synchronous and Asynchronous Execution Modes

Event Handlers are invoked synchronously or asynchronously. In general, the term synchronous means it is run as part of the application within the current action. A synchronous operation blocks a process until the operation completes, while an asynchronous operation is nonblocking and only initiates the operation.

In Agile PLM, the difference between the two modes is:

- **Synchronous** - In this mode, the Event Handler is executed in the same thread (as part of the execution of the action taken) as the Agile PLM thread that triggers the event (for example, a change in a Workflow status). The original Agile PLM action will resume after the handler action is completed.
- **Asynchronous** - In this mode, the Event Handler has its own thread (runs independent of the execution of the action taken) and it cannot be stopped once it is started. This transaction is either committed or rolled back based on its own status. The Agile PLM thread that triggers the event will continue to run independently regardless of the Handler action has finished or not (Non-block). Notifications are always handled asynchronously.

## Synchronous and Asynchronous Operations in OAS Clusters

PLMs that are configured for operation in clusters, use the Java Message Service (JMS) for messaging purposes within the cluster. JMS is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients.

In OAS clusters, JMS resources are hosted only on the primary node, or JMS-host. If the primary node is down, asynchronous Events are still triggered on the secondary node and are set to queued status, but they are never executed. Once the primary node is recovered, depending on the implementation of OAS, asynchronous Events can execute and cause the final result, provided they can pick up the queued JMS messages at the time the system went down. Otherwise, these Events remain in the queue and are never executed.

In these situations, you must stop and restart the cluster, starting with the primary node. If you cannot recover the primary, you must configure the primary (the JMS-host) on another node before starting the cluster. Queued Events do not execute if you drop the primary and continue to use the application on the secondary nodes.

---

**Note:** This constraint is only applicable to the operation of asynchronous Events in OAS clusters. It does not apply to WLS clusters, because WLS supports clustered JMS and has no single "primary" or "JMS-host" node.

---

### Event Error Handling Rule

Event Error Handling Rule is used with Event Handlers that are executed synchronously. Options are Continue and Stop. The selected option determines the behavior of Agile PLM when an error is encountered while executing the Event Handler. For more information on error handling rules, refer to Agile PLM Administrator Guide.

- **Continue** - In case of error, Handlers with synchronous execution mode will ignore the error during execution.
- **Stop** - If there is an error during the execution of a Handler with synchronous execution mode, the original action and the Event Subscription is ceased.

### Event Order

**Event Order** is a positive integer that determines the sequential "Order" in which the Event handler is invoked. Thus, you can control the execution order of Event Handlers when there are multiple Event Subscribers for the same Event type on the same Agile object.

---

**Note:** If you have both Custom PXs and Java synchronous PXs configured for a Workflow Change Status action, Java PXs always execute before Custom PXs.

---

### Event FAQs

This section answers common questions about the Event framework and Java and Script process extensions.

**What are the differences between Custom process extensions and Java process extensions (Java PX)?**

Similar to Custom process extensions, **Java PXs** also extend the functionality of Agile PLM clients through custom actions. This is done by implementing the IEventAction interface in com.agile.px in Event framework. You can use Process extensions to connect the Agile PLM server and Agile PLM users to external systems. In addition, Java PXs contain Event Context objects which provide more information than Custom PXs.

**Can I use Agile's Java API within a Java process extension program?**

Yes. You can use Agile's Java API and other external Java APIs as you did with Custom PXs.

**Do Java PXs have special security requirements?**

No, similar to Custom PXs.

**How are roles and privileges defined for Java PXs/Script PXs?**

By default, a custom action (a Handler) uses the roles of the current user. However, to configure a custom action to have expanded privileges, you can specify the roles required for the Handler in the Java Client. When the Handler is executed, the roles specified for the Handler override those of the current user. Once the Handler is completed, the client reverts to the user's original roles and privileges.

**Do user assigned roles override roles assigned to a Java or Script PX during configuration?**

No. Roles assigned to a Java or Script PX override the user's original roles. Thus all actions that occurred inside the PX Handler are subject to privileges based on these roles. However, the access of the event context object does not need the privilege check, including the getXXX and setXXX method calls.

**How do I configure and deploy a Java process extension?**

Similar to Custom PXs, place the JAR file(s) for a process extension in the agile_home/integration/sdk/extensions folder on the application server. Included with the JAR file(s) should be a file named com.agile.px.IEventAction in the META-INF/services directory. The contents of these files are the fully qualified Java class names, one class per line, for an Event action.

**After deploying a Java PX on the application server, how do I enable it?**

Once Java PX codes are deployed, you can configure them for use within Agile PLM in Java Client by selecting **Admin > Settings > System Settings > Event Management > Event Handlers > New > Create Event Handler > Java PX > Event Action**.

**After deploying JAR file(s) for a Java PX Handler, is it necessary to update the application server classpath?**

No. The classpath is updated automatically by a special-purpose classloader. The classloader extends the application server classpath with any classes located in agile_home/integration/sdk/extensions (or the location specified for the sdk.extensions property in the agile.properties file).

**Which Custom PXs can I migrate to the Event framework?**

The Event framework supports migrating only Custom Action PXs. These are PXs that are initiated from the Actions Menu, Tools Menu, and Change Status for Workflow.

The corresponding Event types are: Extend Actions Menu, Extend Tools Menu, and Change Status for Workflow.

**What error handling rules must the developer/user specify?**

For Synchronous Handler, the user must specify the error handling rule to determine how the system reacts if it encounters an error while processing this subscription. The error handling rule only applies to synchronous Handlers. It supports the following two choices, Stop and Continue.

Agile PLM stops any further event processing, and then returns to the originator who raises the event. All remaining synchronous subscribers are not called.

In the case of pre-event, upon receiving the error from the subscriber, the originator simply throws the error to the client that initiates the action and the original action is not performed. The system may also rollback changes made by Handlers. However, whether the transaction can be rolled back or not depends on the Handler type. If it is a Java PX Handler, no transaction rollback is performed because the Handler is the SDK program which has its own transaction.

**Do I need to deploy script PXs?**

No, Scripts are pasted in the editor in the Script Handler and Agile PLM will store the code in the database. Consequently, programs are delivered in plain text files and not in object code.

**Can I send Notifications using Event Handler?**

Yes, you can send a Notification from a Java PX and a Script PX.

**When should I use scripting?**

Use Scripts for prototyping, simple operations, and test driven developments.

**Do I need to compile my Script code?**

No, Script code is validated for syntax errors when you save it in the Handler and will be compiled when the Event is triggered.

**What are Dirty files and related methods and interfaces?**

They are documented in the Javadoc generated SDK documentation folder under IEventDirtyRowFileUpdate. To access the SDK samples folder, see the **Note** in "Client-Side Components" on page 1-2.

**Can a single action trigger multiple Events?**

Yes, for some actions such as Update Multiple Attachment Rows, a single action will trigger multiple Events. In addition, if there are Subscribers for Pre and Post trigger types, then the order in which the Subscribers are invoked can vary depending on the PLM client, the action, and the object type. For example, if you are deleting three rows from an Attachment Table with one Subscriber for the Pre trigger and one Subscriber for the Post trigger, the behavior in PLM clients are as follows**:**

- **Web Client** - A single Update Table Event is triggered
- **Java Client** - Three Update Table Events are triggered and one Event for each deleted row as shown below:
  - For Changes, Items, Transfer Orders, MFR Parts, Suppliers, Sites, Customers, and Package objects, the sequence of the Event Subscribers that are invoked is:
    - Pre (handler from the pre trigger subscriber - for the first row)
    - Pre (handler from the pre trigger subscriber - for the second row)
    - Pre (handler from the pre trigger subscriber - for the third row)
    - Post (handler from the post trigger subscriber - for the first row)
    - Post (handler from the post trigger subscriber - for the second row)
    - Post (handler from the post trigger subscriber - for the third row)
  - For Product Service Requests, Quality Change Requests, MFRs, Users and User Groups, the sequence is: Pre - Post - Pre - Post - Pre - Post

**Best Practices?**

- What should we avoid in Handlers for Pre-Event?

  Use Pre-Event mainly for validation. Although you can modify the Context object, you should avoid direct object updates using SDK calls. (Sometimes, using SDK calls in Pre-Event may cause object version mismatch)?

- Subscriber ordering?

  Sometimes, for the same event type, you can have the Event Subscribers at Base Class level, Class level, and Subclass level. For example, you may have several Event Subscribers for Create Object Event Type. One at Item Base Class level, one at the Part Class level, and one at the Part Subclass level. If you prefer to execute the Subscribers based on Class hierarchy, it is recommended to allocate an order range for the Base Class, each Class, and each Subclass. For example, you can assign the following range for different Hierarchy levels:

  Base Class 0 – 99; Class 100 – 199; Subclass 200 – 299

- Can I mix Agile SDK calls and Script PX within Script PX?

  You can mix the two as long as you don't use both SDK calls and script PX calls to update Agile objects in the same Handler code. To update the Agile object, you can either use SDK calls or update APIs that are supported by Script Context object, but not both.

  **What are the differences between Variant Management events and other system events?**

  - Variant Management does not have "Pre" or "Post" trigger types. Once an event is enabled, it replaces the system behavior instead of extending it.
  - Variant Management event types are only applicable to the Model part subtype.
  - The Variant Management event types are not linked to a specific action only.
  - Variant Management event types support only one specific execution mode (Synchronous) and error handling rule (Stop).

# Working with Event Context Objects

The flowing paragraphs describe the role of Event Context objects, their creation, different Event types, and the information they maintain.

## Understanding Event Context Objects

The Context object passes information from the Event to the Handler and between Handlers. When an Event is raised, an Event context object is created. Information maintained by the context object includes Event type, pre- and post-Event triggers, plus business-related data for the given Event, such as the Agile object for which the Event occurred. The business-related data varies based on Event type.

Different Event types have different Context objects. Interfaces for Context objects are documented in Event Information Objects and Event Script Objects. These interfaces are documented in the Javadoc HTML files as *Event Information objects* **(**Java interfaces) and *Event Script objects* (Script interfaces). To find the Javadoc HTML files in the SDK_ samples.zip folder, see the Note in "Client-Side Components" on page 1-2.

### Persistent and Transient Data

In Event framework, objects passed to Event Handlers (Java PX or Script PX) by Context objects contain "**P**ersistent" and "Transient" data.

---

**Note:** For Variant Management all Context objects contain "Transient" data only.

---

- **Persistent data** - This is data that is already in PLM databases. All Agile SDK APIs and Process Extensions including Web or URL extensions deal with this type of data. When you use `getDataObject()` to get the values of an `IDataObject` object, the data you get is already in the database, hence "Persistent."

> **Note:** Only the `getDataObject()`method returns Persistent data. Other context object get data methods will return Transient data unless stated otherwise in the Javadoc API definition.

- **Transient data** - This is data that is not in the PLM database as yet and is in a state of change. Transient data contains information about user requests for the action that triggered the Event.

When the Event is triggered, Agile PLM creates Transient data in the Event Context object. The same Transient data is passed on by the Context object to all Handlers for "Pre" and "Post" unless specified otherwise in the Javadoc API definition.

> **Note:** You must not modify Transient data in the "Post" Event triggering instance. If modified, it will throw an exception. Variant Management events do not have "Pre" or "Post" trigger types. Once a Variant Management event is enabled, it replaces the system behavior instead of extending it.

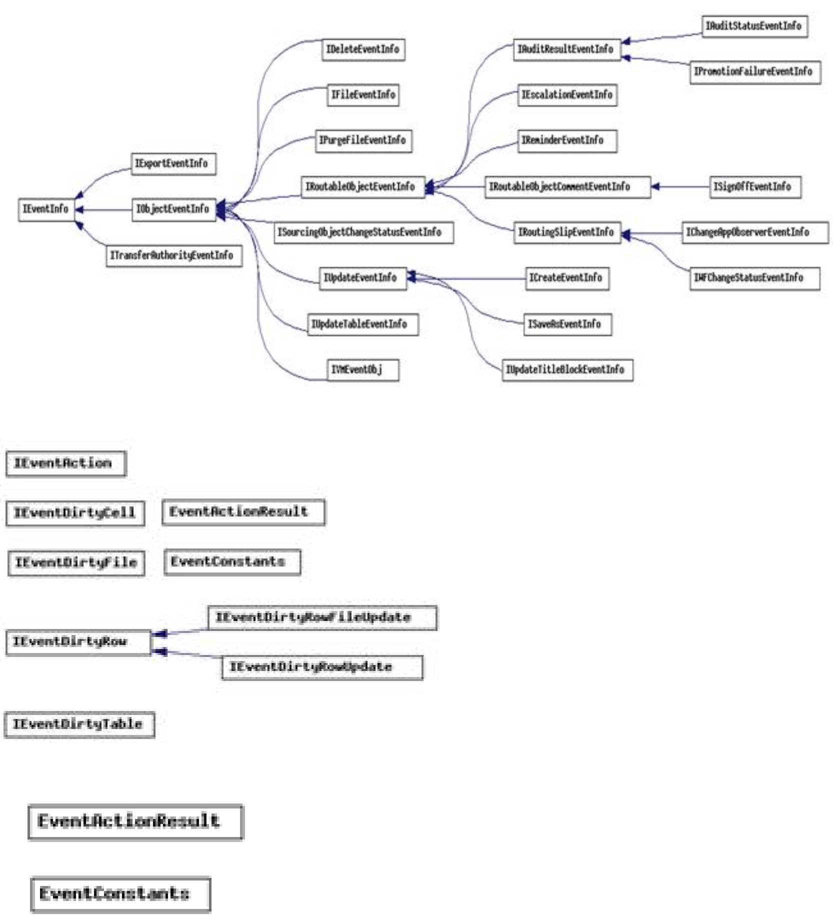
### Event Information Objects

These are the interfaces for the Java PXs.

| Event Type | Event Information Object |
|---|---|
| "Approve for Workflow" and "Reject for Workflow" | `ISignOffEventInfo` |
| "Audit for Workflow" | `IAuditStatusEventInfo` |
| "Change Approvers or Observers for Workflow" | `IChangeAppObserverEventInfo` |
| "Change Status for Sourcing Object" | `ISourcingObjectChangeStatusEventInfo` |
| "Change Status for Workflow" | `IWFChangeStatusEventInfo` |
| "Comment for Workflow" | `IRoutableObjectCommentEventInfo` |
| "Create Object" | `ICreateEventInfo` |
| "Create Variant Instance", "Derive Variant Model Option BOM", "Update Variant Configuration", "Validate Variant Configuration", "Validate Variant Instance Selections", and "Validate Variant Model Option BOM", | `IVMEventObj` |
| "Delete Object" | `IDeleteEventInfo` |
| "Escalation for Workflow" | `IEscalationEventInfo` |
| "Export Object" | `IExportEventInfo` |
| "Extend Tools Menu" and "Scheduled Event" and the base interface for all Event information objects | `IEventInfo` |
| "Get File", "Check In Files", "Check Out Files", and "Cancel Check Out Files" | `IFileEventInfo` |
| "Incorporate File", "Unincorporate File" | `IIncorporateFileEventInfo` |

| Event Type | Event Information Object |
|---|---|
| “Incorporate Item”, “Unincorporate Item”, “Extend Actions Menu”, and “Compliance Rollup on Object” | IObjectEventInfo |
| “Promotion Failure for Workflow” | IPromotionFailureEventInfo |
| “Purge Version Files” | IPurgeFileEventInfo |
| “Reminder for Workflow” | IReminderEventInfo |
| “Save As Object” | ISaveAsEventInfo |
| “Transfer Authority” | ITransferAuthorityEventInfo |
| “Update Table” and “Update Relationship” | IUpdateTableEventInfo |
| “Update Title Block” | IUpdateTitleBlockEventInfo |

***Figure 5–3 Event information objects class hierarchy***



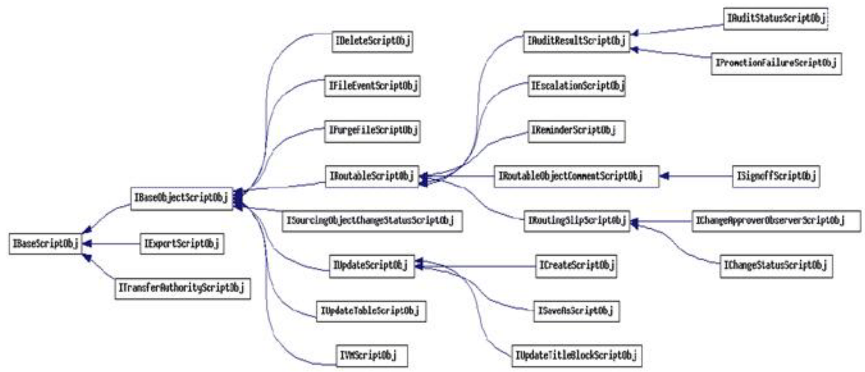
### Event Script Objects

These are the interfaces for the Script PXs.

| Event Type | Script Object Interface |
|---|---|
| “Approve for Workflow” and “Reject for Workflow” | ISignoffScriptObj |
| “Audit for Workflow” | IAuditStatusScriptObj |
| “Change Approvers or Observers for Workflow” | IChangeApproverObserverScriptObj |
| “Change Status for Sourcing Object” | ISourcingObjectChangeStatusScriptObj |
| “Change Status for Workflow” | IChangeStatusScriptObj |
| “Comment for Workflow” | IRoutableObjectCommentScriptObj |
| “Create Object” | ICreateScriptObj |
| “Create Variant Instance”, “Derive Variant Model Option BOM”, “Update Variant Configuration”, “Validate Variant Configuration”, “Validate Variant Instance Selections”, and “Validate Variant Model Option BOM”, | IVMScriptObj |
| “Delete Object” | IDeleteScriptObj |
| “Escalation for Workflow” | IEscalationScriptObj |
| “Export Object” | IExportScriptObj |
| “Extend Tools Menu” and “Scheduled Event “Event Script objects, also the base interface for all Event Script objects | IBaseScriptObj |
| “Get File', “Check In Files”, 'Check Out Files' and “Cancel Check Out Files” | IFileEventScriptObj |
| “Incorporate Item”, “Unincorporate Item”, “Extend Actions Menu”, and “Compliance Rollup on Object” | IBaseObjectScriptObj |
| “Promotion Failure for Workflow” | IPromotionFailureScriptObj |
| “Purge Version File Folder” | IPurgeFileScriptObj |
| “Reminder for Workflow” | IReminderScriptObj |
| “Save As Object” | ISaveAsScriptObj |
| “Transfer Authority” | ITransferAuthorityScriptObj |
| “Update Table” and “Update Relationship” | IUpdateTableScriptObj |
| “Update Title Block” | IUpdateTitleBlockScriptObj |

**Note:** In a Script PX, you can invoke any SDK or Java call. You can also access any third party Java or Groovy Libraries if they are deployed in the SDK extensions directory.

***Figure 5–4 Event script objects class hierarchy***




# Working with Event Information and Event Script Objects

Information provided in this section uses the Event Information objects and Event Script Objects to develop PX Handlers for Event-related actions, for example, general object actions such as Create Object, Delete Object; Workflow actions such as Change Status for Workflow.

Descriptions and samples for Event Information objects and Event Script objects appear in the following paragraphs. Other information includes code samples using these objects and guidelines to ensure proper handling of special instances of Events in Agile PLM.

## Working with Base Event Actions

All Event Information objects use the Base Event Information objects that. are Java interfaces. See the illustration in "Event Information Objects" on page 5-9.

### Base Event Information Object - Java PX

The Base Event Information object is IEventInfo.

- **Purpose and function** - This is the interface for the *Scheduled Event* and Extended Tools Menu Event Information objects. In addition, IEvent Info is the inherited interface for all Event Information objects. It maintains information on Event type, Event trigger type, Event name, Event Subscriber name, Event Handler name, and user defined Maps.

User defined Maps serve as a place holder for any user defined data and provide a communication channel between subscribers. They are set in the Synchronous Java PXs and read by subsequent Synchronous Java PXs and Asynchronous Java PXs. Maps set inside Asynchronous PXs cannot be used by other Java PXs. If the Java PX fails, the Map is still accepted by Agile PLM and is passed to the next Java PX but all other changes in the Event Context objects are discarded.

IObjectEventInfo is the base event information object for object-related events.

- **Purpose and function** - IEventInfo contains the Agile object for which the Event is triggered. Also, it is the interface for Incorporate Item, Unincorporate Item, Extend Actions Menu, and Compliance Rollup on object Event Information objects, and inherits from IEventInfo.

- **Inherited interface:** IEventInfo

These examples show using IEventInfo and IObjectEventInfo.

***Example 5–1 Using IEventInfo***

```
private void testIEventInfo(IEventInfo req)throws APIException{

// getEventTriggerType()
   int evtTriggerType = req.getEventTriggerType();

// getEventName()
   String eventName = req.getEventName();

// getEventSubscriberName()
   String subscriberName = req.getEventSubscriberName();

// getEventHandlerHame()
   String handlerName = req.getEventHandlerName();

// setUserDefinedMap()
   Map map = new HashMap();
   map.put("METHOD", "setUserDefinedMap()");
   req.setUserDefinedMap(map);

// getUserDefinedMap()
   Map map2 = new HashMap();
   map2 = req.getUserDefinedMap();
   String mapValue = map2.get("METHOD").toString();
}
```

***Example 5–2 Call made by IObjectEventInfo***

```
private void testIObjectEventInfo(IEventInfo req)throws APIException{
   String objNumber = "";
   IObjectEventInfo info = (IObjectEventInfo)req;

// getDataObject()
   IDataObject obj = info.getDataObject();
   if (obj==null)
      objNumber = "NULL";
   else
      objNumber = obj.getName();
}
```

### Base Event Script Objects - Script PX

Base Event Script object is `IBaseScriptObj`.

- Purpose and function - This is the interface for the *Scheduled Event* and *Extend Tools Menu* Event Script objects. `IBaseScriptObj` is the inherited interface for all Event Script objects. `IBaseScriptObj` maintains information on Event type, Event trigger type, Event name, Event Subscriber name, Event Handler name, and User Defined Maps.

  `IBaseObjectScriptObj` is the base event information object for object-related events.

  > **Note:** This interface provides a Script method for sending Agile PLM Notifications.

- **Purpose** and function - `IBaseObjectScriptObj` contains the Agile object for which the Event is triggered. Also, it is the interface for Incorporate Item, Unincorporate Item, Extend Actions Menu, and Compliance Rollup on object Event Information objects and inherits from `IBaseScriptObj`.
- **Inherited interface:** `IBaseScriptObj`

The following examples use `IBaseScriptObj` and `IBaseObjectScriptObj`.

***Example 5–3 Using IBaseScriptObj***

This example opens the SDK session and retrieves information pertaining to the Event.

```
void invokeScript(IBaseScriptObj obj){

// getEventType()
   int evttype = obj.getEventType();

// getEventTriggerType()
   int evtTriggerType = obj.getEventTriggerType();

// getEventName()
   String eventName = obj.getEventName();

// getEventSubscriberName()
   String subscriberName = obj.getEventSubscriberName();

// getEventHandlerHame()
   String handlerName = obj.getEventHandlerName();

//logMonitor ()
   obj.logMonitor("Status is Passed");

//getAgileSDKSession()
   IAgileSession session = obj.getAgileSDKSession();

//getPXEventInfo ()
   IEventInfo req = obj.getPXEventInfo();

//sendNotification()
   obj.sendNotification("Test", true, [ "admin"],
      " passed from BaseScriptObj" + eventName);

// setUserDefinedMap()
   obj.setUserDefinedMap
      ( [ 'Agile 93' : 'PLM Product', 'Scripting':'Is a Fun Tool']);

// getUserDefinedMap()
   Map myMap = obj.getUserDefinedMap ();
}
```

Example 5–4, "Using IBaseObjectScriptObject" retrieves Class ID and object number and sets the P1, P2, and P3 attribute values of the object.

***Example 5–4 Using IBaseObjectScriptObject***

```
void testIBaseObjectScriptObj
        (IBaseObjectScriptObj obj){
   String objNumber = "";

//Disable all the warning exceptions raised
   obj.disableAllWarnings();

//get class ID
   int classID = obj.getClassId();

//get object number
   objNumber = obj.getObjectNumber();

//set attributes
   obj.setValueByAttId(CommonConstants.ATT_PAGE_TWO_TEXT01,"Text Value");

// get attribute value
   obj.getValueByAttId(CommonConstants.ATT_PAGE_TWO_TEXT01);

//log information to handler monitor
   obj.logMonitor
      ("Object Number:" + objNumber); obj.logMonitor("Class Id :" + classID);
}
```

## Working with General Object Actions

General Object actions are actions such as Create, Delete, Save As, and Update Title block. Information objects and Script Event objects for these Events are grouped and described according to their inherited interfaces.

### General Object Actions - Java PX

The following paragraphs document the applicable methods and procedures.

#### Create Object

The Information object for this Event is ICreateEventInfo.

- **Purpose and function** - ICreateEventInfo retrieves the number and Subclass identifier of the requested new object. It can also overwrite the number and Subclass set by the clients.
- **Inherited interfaces** - IUpdateEventInfo, IObjectEventInfo, IEventInfo. The IUpdateEventInfo retrieves the Array of Dirty attributes that users can overwrite, or set with new attributes and values.

The following examples show using ICreateEventInfo and IUpdateEventInfo

***Example 5–5 Using ICreateEventInfo***

```
private void testICreateEventInfo(IAgileSession session, IEventInfo req)
throws APIException {
   ICreateEventInfo info = (ICreateEventInfo)req;
   String number = "";
   Integer subclass = null;
   String newNumber = getNewNumber(info);

   Integer newSubclassId = getNewSubclassId(session, req);
   Integer newSubclass = null;
```

```
// getNewNumber()
   number = info.getNewNumber();

// getNewSubclassId()
   subclass = info.getNewSubclassId()

// setNewNumber()
   info.setNewNumber(newNumber);

// setNewSubclassId()
   info.setNewSubclassId(newSubclassId);
   newSubclass = info.getNewSubclassId();
}
```

***Example 5–6 Using IUpdateEventInfo***

```
private void testIUpdateEventInfo(IAgileSession session, IEventInfo
req) throws Exception {

// Interface methods
   IUpdateEventInfo info = (IUpdateEventInfo)req;

//getCells()
   IEventDirtyCell[] cells = info.getCells();

// getAttributeIds()
   Integer[] attrs = info.getAttributeIds();

// setCell() and Get class specific P1 attribute
   Integer p1attrId = getP1Attribute(session, info);

// Override client value
   info.setCell(p1attrId, "set desc from CO");

// Add new Dirty value
   info.setCeldirtyl(CommonConstants.ATT_PAGE_TWO_TEXT02, "setCell()");
   String value2 = info.getValue(CommonConstants.ATT_PAGE_TWO_TEXT02).toString();

// setCell() one from each attribute type
   * Date: Page Two.Date01
   * Text: already cover
   * MultiText: Page Three.MultiText10
   * List: Page Two.List01
   * MultiList: Page Three.MultiList01
   * Numeric: Page Two.Numeric 01
   * Money: Page Three.Money01
*/

   Integer subClassId = getSubclassId(info);
   IAttribute attr1 =
      session.getAdminInstance().getAgileClass
         (subClassId).getAttribute(CommonConstants.ATT_PAGE_TWO_LIST01);

   IAttribute attr2 =
      session.getAdminInstance().getAgileClass(subClassId).getAttribute
         (CommonConstants.ATT_PAGE_THREE_MULTILIST01);

   IAgileList list1 =
      (IAgileList)attr1.getAvailableValues();
   list1.setSelection(new Object[]{"b"});
```

```
    IAgileList list2 = null;
    if (attr2!=null){
       list2 = (IAgileList)attr2.getAvailableValues();
       list2.setSelection(new Object[]{"a", "b", "e"});
    }
    SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy");
    String d = "1/31/2009";
    Date date = df.parse(d);
    info.setCell(CommonConstants.ATT_PAGE_TWO_DATE01, date);
    String multitext = "set multitext field in CO";
    info.setCell(CommonConstants.ATT_PAGE_TWO_LIST01, list1);
    info.setCell(CommonConstants.ATT_PAGE_TWO_LIST02, list1);

// To test IEventDirtyCell
    info.setCell(CommonConstants.ATT_PAGE_TWO_NUMERIC01, 888.66);
    if (attr2!=null){
       info.setCell(CommonConstants.ATT_PAGE_THREE_MULTITEXT10, multitext);
       info.setCell(CommonConstants.ATT_PAGE_THREE_MULTILIST01, list2);
    }
    Money money = new Money (new Integer(100), "USD");

// removeCell()
    info.removeCell(CommonConstants.ATT_PAGE_THREE_TEXT01);

}
```

### Update Title Block

The Information object for this Event is `IUpdateTitleBlockEventInfo`.

- **Purpose and function** - `IUpdateTitleBlockEventInfo` is the interface for Update Title Block Event information object. Checks whether this is a redline update or undo-redline update on the Title Block of IItem object which has originated from the Affected Items table of the Change object
- **Inherited interfaces** - `IUpdateEventInfo, IObjectEventInfo, IEventInfo`

#### *Save As Object*

The Information object for this Event is `ISaveAsEventInfo`.

- **Purpose and function** - `ISaveAsEventInfo` performs the following tasks:
  - Retrieves the number and Subclass of the newly saved object
  - Overwrites the number and Subclass that are set by PLM clients
- **Inherited interfaces** - `IUpdateEventInfo, IObjectEventInfo, IEventInfo`

#### *Delete Object*

The Information object for this Event is `IDeleteEventInfo`.

- **Purpose and function** - `IDeleteEventInfo` is the interface for Delete Event information object. It performs the following tasks:
  - Retrieves the number and Subclass of the newly saved object
  - Overwrites the number and Subclass that are set by PLM clients
- **Inherited interfaces** -`IObjectEventInfo, IEventInfo`

### Export Object

The Information object for this Event is `IExportEventInfo`.

- **Purpose and function** - `IExportEventInfo` performs the following tasks:
  - Retrieves the format of the export file
  - Returns the array of objects that are exported
  - Returns the tables for exporting the object
- **Inherited interfaces** - IEventInfo

## General Object Actions - Script PX

This section lists and describes Script PXs that support General Action Objects

### Create Object

The Script Event object for this Event is `ICreateScriptObj`.

- **Purpose and function** - The same as `ICreateScriptObj`
- **Inherited interfaces** - `IUpdateScriptObj, IBaseObjectScriptObj, IBaseScriptObj`
- **Purpose and function of the inherited interfaces** - The same as `IUpdateEventInfo, IObjectEventInfo` and `IEventInfo`

The following examples use `ICreateScriptObj` and `IUpdateScriptObj`.

**Example 5–7 Using ICreateScriptObj**

```
// In this example, ICreateScriptObj modifies the number and class ID of the new
object.

void testICreateScriptObj(IBaseScriptObj obj){

   String origNumber = "";
   String newNumber = "";
   int newSubclassId =
      ItemConstants.CLASS_DOCUMENT ;

// new subclass ID of choice
   int newSubclass ;
   int subclass;

// getNewNumber()
   origNumber = obj.getNewNumber();
   newNumber = origNumber +"new";

// setNewNumber()
   obj.setNewNumber(newNumber);
   newNumber = obj.getNewNumber();

// getNewSubclassId()
   subclass = obj.getNewSubclassId();

// setNewSubclassId()
   obj.setNewSubclassId(newSubclassId);
   newSubclass = obj.getNewSubclassId();

// log new object number and new subclass ID in to handler monitor
```

```
   obj.logMonitor("new object number is:" + newNumber);
   obj.logMonitor("new subclass ID is:" + newSubclass);
}
```

In Example 5–8, "Using IUpdateScriptObj"the `IUpdateScriptObj` method Gets the ID and value of Dirty attributes. It then sets the Dirty attribute values obtained from the context object, and removes the value of Dirty attributes.

***Example 5–8 Using IUpdateScriptObj***

```
void testIUpdateScriptObj(IBaseScriptObj obj){
   String dirtyAttr = "";
   String dirtyValue = "";

// get Attribute Ids
   int[] attrs = obj.getAttrIds();
   attrs.each {attr->

// get Attribute value
   dirtyAttr = obj.getDirtyAttr(attr);

// log attribute Id and value in to handler monitor
   obj.logMonitor("Dirty Attr Id :" + attr);
   obj.logMonitor(" dirty Attr value:" + dirtyAttr);
   }

// Overwrite client value
   obj.setDirtyAttrValue(CommonConstants.ATT_PAGE_TWO_TEXT02,
      "set text value from CO");

//Remove Dirty Attribute value
   obj.removeDirtyAttr(CommonConstants.ATT_PAGE_TWO_TEXT02);

// get dirty attribute value after removing value from CO
   dirtyValue = obj.getDirtyAttrValue(CommonConstants.ATT_PAGE_TWO_TEXT02);
}
```

### Update Title Block

The Script Event object for this Event is `IUpdateTitleBlockScriptObj`.

- **Purpose and function** - `IUpdateTitleBlockScriptObj` checks whether this is a redline update or undo-redline update on the Title Block of the IItem object which has originated from the Affected Items table of Change object
- **Inherited interfaces** - `IUpdateScriptObj, IBaseObjectScriptObj, IBaseScriptObj`

### Save As Object

Script Event object for this Event is `ISaveAsScriptObj`.

- **Purpose and function** - `ISaveAsScriptObj` retrieves the name or number for the new object.
- **Inherited interfaces** - `ISaveAsScriptObj, IBaseObjectScriptObj, IBaseScriptObj`

#### *Delete Object*

The Information object for this Event is `IDeleteScriptObj`.

- **Purpose and function** - `IDeleteScriptObj` deletes the Event Script object.
- **Inherited interfaces** - `IBaseObjectScriptObj, IBaseScriptObj`

#### *Export Object*

Script Event object for this Event is `IExportScriptObj`.

- **Purpose and function** - `IExportScriptObj` retrieves the format for the export object.
- **Inherited interfaces** - `IExportScriptObj`

Example 5–9, "Using IExportScriptObj" retrieves information about the object that is exported.

***Example 5–9 Using IExportScriptObj***

```
void testIExportScriptObj(IBaseScriptObj obj){
   int format;
   int[] selectedTables;
   String objects ="";
   String user = "";
String objectNumber =
"P00002" ; // the object number being exported

// get current user
   user = obj.getCurrentUser();

// get file format for the export.
   format = obj.getExtractedFormat();

// get list of Object Names being exported
   objects = obj.getExtractedObjects();

// get tables selected for export
   selectedTables = obj.getSelectedTables(objectNumber);

// log current user, extracted format, object names and tables which
exported into handler monitor
   obj.logMonitor("current user is:" + user);
   obj.logMonitor("file format is:" + format);
   obj.logMonitor("Object Names are:" + objects);
   obj.logMonitor("Selected Tables are:" + selectedTables);
}
```

## Working with Table and Relationship Actions

These actions include updating the supported object tables for specific business objects.

### Table and Relationship Actions - Java PX

This section lists and describes the applicable Java PXs.

### Update Table

The Information object for this Event is `IUpdateTableEventInfo`.

- **Purpose and function** - `IUpdateTableEventInfo` is the interface for *Update Table* and *Update Relationship*. It retrieves the Dirty table for the affected object.
- **Inherited interfaces** - `IObjectEventInfo` and `IEventInfo`.
- **Related interfaces:**
  - `IEventDirtyFile` is the interface for a Dirty file associated with `IEventDirtyRowFileUpdate` or `IFileEventInfo`. It represents a single row in a file table and retrieves the checkout date of the Dirty file.
  - IEventDirtyTable is the interface for a Dirty table associated with IUpdateTableEventInfo or IEventDirtyRow. The Dirty table contains a collection of modified rows. It provides access to transient table information for modified tables.
  - IEventDirtyCell is the interface for a Dirty cell associated with IEventDirtyRowUpdate or IUpdateEventInfo. Represents a single cell in a row. It returns the attribute identifier corresponding to this Dirty cell.
  - IEventDirtyRowFileUpdate retrieves the Dirty file. It is the interface for a Dirty row used to perform Dirty Row Actions (Add file, Replace file).
  - IEventDirtyRowUpdate retrieves the Dirty row for which the update occurs. This interface is used by the update on all tables except the attachment table.

> **Note:** You can find information on "Dirty" objects in the SDK samples Documentation folder. To access this folder, see the Note in "Client-Side Components" on page 1-2.

These examples use IEventDirtyTable and IEventDirtyRowUpdate on an Item BOM table.

#### *Example 5–10 Using IEventDirtyTable*

```
private void testIEventDirtyTable(IAgileSession session,
   IEventDirtyTable table, IDataObject obj, int evtTriggerType) throws Exception {

//Get TableId()
   String tableName = getTableName(obj, table);

// size()
   int size = table.size();

//iterator()
   Iterator it = table.iterator();
   while (it.hasNext()){
      IEventDirtyRow row = (IEventDirtyRow)it.next();
      if(row.getAction()!=
         EventConstants.DIRTY_ROW_ACTION_ADD_FILE&&row.getAction()!=
            EventConstants.DIRTY_ROW_ACTION_REPLACE_FILE)

//user defined method
       testIEventDirtyRowUpdateCommon(session, row, obj, evtTriggerType);
   else
       testIEventDirtyRowFileUpdate(row);// user defined method
   }
}
```

***Example 5–11 IEventDirtyRowUpdate on Item BOM table***

```
private void testIEventDirtyRowUpdate_ItemBOM_Update
     (IEventDirtyRowUpdate row, IDataObject obj) throws Exception {

// user defined method
/* setCell() - Override
   * List01 ==> c
   * MultiText30 ==> setCell() on update
   * Text01 ==> setCell()on update
   * Numeric01 ==> 888.66
   * BOM Notes ==> setCell() from CO
*/

//List01
   IAttribute attrList =
   obj.getAgileClass().getAttribute(ItemConstants.ATT_BOM_BOM_LIST01);
   IAgileList list =
      (IAgileList)attrList.getAvailableValues();
   list.setSelection(new Object[]{"c"});
   row.setCell(ItemConstants.ATT_BOM_BOM_LIST01, list);

//MultiText30
   row.setCell(ItemConstants.ATT_BOM_BOM_MULTITEXT30, "setCell() MT30 update");

//Text01
   row.setCell(ItemConstants.ATT_BOM_BOM_TEXT01, "setCell() T01 update");

//Numeric01
   row.setCell(ItemConstants.ATT_BOM_BOM_NUMERIC01, 888.66);

//BOM Notes
   row.setCell(ItemConstants.ATT_BOM_BOM_NOTES, "setCell() Notes update")

/*
   * setCell() - New
   * List02 ==> a
   * Text02 ==> setCell() on update
*/

// List02
   row.setCell(ItemConstants.ATT_BOM_BOM_LIST02, list);

//Text02
   row.setCell(ItemConstants.ATT_BOM_BOM_TEXT02, "setCell() T02 update");

// Qty
   row.setCell(ItemConstants.ATT_BOM_QTY, new Integer(2));

//removeCell() ==> Date01 & Find Num
   row.removeCell(ItemConstants.ATT_BOM_BOM_DATE01);
   row.removeCell(ItemConstants.ATT_BOM_FIND_NUM);

// getCell()
   EventDirtyCell cell = row.getCell(ItemConstants.ATT_BOM_BOM_LIST02)

}
```

## Update Relationship

See Update Table in "Table and Relationship Actions - Java PX" on page 5-20.

### Table and Relationship Actions - Script PX

This section lists and describes the Script PXs that support Table and Relationships action.

#### *Update Table*

The Script Event object for this Event is IUpdateTableScriptObj.

- **Purpose and function** - IUpdateTableScriptObj is the interface for Update Table and Update Relationship. It retrieves the dirty table for the affected object.
- **Inherited interfaces** - IUpdateTableScriptObj. See "Base Event Script Objects - Script PX" on page 5-13.
- **Related interfaces**
  - IEventDirtyRow is the base interface for a dirty row associated with IUpdateTableScriptObj. It represents a single row in a table. There is an action associated with the row.
  - IEventDirtyRowFileUpdate retrieves the dirty file row. It is the interface for a dirty file row used to perform Dirty File Row Actions (Add file, Replace file).
  - IEventDirtyRowUpdate retrieves the Dirty row. It is the interface for a Dirty row used to perform Dirty Row Actions (Add, Delete, Update, Redline Add, Redline Delete, Redline Update, Undo Redline.)
  - IEventDirtyFile is the interface for a Dirty file associated with IEventDirtyRowFileUpdate or IFileEventInfo. It represents a single row in a file table and retrieves the checkout date of the Dirty file.

The following examples show using IEventDirtyTable and IEventDirtyRowUpdate on an Item BOM table.

***Example 5–12 Using IEventDirtyTable***

```
// IUpdateTableScriptObj: get table id and Dirty row ids
   void testIUpdateTableScriptObj(IBaseScriptObj obj)
{
//getTableId()
  int table_id= obj.getTableId();
  obj.logMonitor("Table Id" + table_id);

// getDirtyRowIds()
  rowIDs =obj.getDirtyRowIds();
  rowIDs.each{rID-> //loop through Dirty rows
  obj.logMonitor("rowID is " + rID);

//getDirtyRow()
  row = obj.getDirtyRow(rID);

// getAction
  action = row.getAction();
  obj.logMonitor ("Action" + action);
  if ((action == EventConstants.DIRTY_ROW_ACTION_ADD_FILE) || (action
      ==  EventConstants.DIRTY_ROW_ACTION_REPLACE_FILE))
  testIEventDirtyRowFileUpdate(obj,row,rID); // user method
else
  testIEventDirtyRowUpdate(obj,row,rID); // user method }
}
```

The following Item BOM table example gets the object number and action for a single row associated with IUpdateTableScriptObj.

***Example 5–13 Using IEventDirtyRowUpdate***

```
void testIEventDirtyRowUpdate
      (IBaseScriptObj obj,IEventDirtyRow row, int rID) {

//getRowId()
   int rid = row.getRowId();

// getObjectNumber()
   int rnumber = row.getObjectNumber();
   obj.logMonitor("row object" + rnumber);

// getDirtyRow
   dirtyRow = obj.getDirtyRow(rID);

//from client, update following attributes in Bom table
// find number,Bom notes, Multi Text01,List01,date01, Text01, numeric0
//getDirtyAttrIds()
   dirtyRow = obj.getDirtyRow(rID);
   dirtyAttrs = dirtyRow.getDirtyAttrIds();
   def sort_attrs = dirtyAttrs as List;

//sort attribute Ids
   sort_attrs.sort();

// getDirtyAttrValue()
   sort_attrs.each {dirtyAttr->
   attrValue = dirtyRow.getDirtyAttrValue(dirtyAttr);
   obj.logMonitor('***'+"attribute value" +'=' + attrValue);
   }
// array of vlues to overwrite attributes from Context object
//[find number,Bom notes, Multi Text01,List01,date01, Text01, numeric01]
set_dirty_value = ["5","Bom Note","Multi text 03","e","2009-01-30 08:00:00","Text
01","224466"];

//setDirtyAttrValue , overwrite attribute values
   int indexy =0;
   sort_attrs.each{att->
   dirtyRow.setDirtyAttrValue(att , set_dirty_value[indexy++]);
   }

// getDirtyAttrValue() after overwrite in CO
   int indexB=0;
   sort_attrs.each {dirtyAttr->
   attrValue2 = dirtyRow.getDirtyAttrValue(dirtyAttr);
   all_attrValue2[indexB++]= attrValue2;
   }
// removetDirtyAttr() remove dirty attributes and rollback changes
      sort_attrs.each {dirtyAttr->
      dirtyRow.removetDirtyAttr(dirtyAttr);
   }
}
```

## Working with Variant Management Events

Variant Management events include:

- Create Variant Instance

  The event handler creates the derived Instance BOM.
- Derive Variant Model Option BOM

  The event handler creates the logical structure of the Instance BOM without actually creating new items or changing the BOM tab of an item.
- Update Variant Configuration

  The event handler adds or removes configuration options and runs propagations and pre-selections.
- Validate Variant Configuration

  The event handler checks the consistency of the Configuration Graph and the Model Option BOM.
- Validate Variant Instance Selections

  The event handler checks validation rules for the configuration, for example, minimum/maximum violations, or if an Option Class has enough valid child options.
- Validate Variant Model Option BOM

  The event handler checks validation rules on the Model Option BOM. For example,  if the minimum quantity value is smaller/equal to the maximum quantity value, or if an Option Class has valid child options.

These events can only be triggered on Model part subtype. Once the event is enabled, it replaces the system behavior instead of extending it.

---

**Note:** You can download the Variant Management SPX and JPX default scripts from Oracle Agile PLM's Event and Web Services Samples website. For information to access this site, see the Note in "Client-Side Components" on page 1-2.

---

### Variant Management Events - Java PX

The information object for these Events is IVMEventObj:

- Create Variant Instance
- Derive Variant Model Option BOM
- Update Variant Configuration
- Validate Variant Configuration
- Validate Variant Instance Selections
- Validate Variant Model Option BOM

IVMEventObj is the interface for the Variant Management events information object. It provides access to the Configuration Graph, the Model Option BOM and the Instance BOM (if available).

---

**Note:** You can find more information on the purpose and function of these Events and when they are triggered, in the *Agile PLM Administrator Guide* and the chapter entitled "Configuring Variant Management" in *Agile PLM Product Collaboration User Guide.*

---

Inherited interfaces - IObjectEventInfo, IEventInfo

The following example shows how to use IVMEventObj

```
private void testIVMEventObj(IVMEventObj req) {
          IConfigurationGraph graph = req.getConfigurationGraph();
         // get Model Option BOM
         IModelOptionBOM mob = req.getModelOptionBOM();
         // getting unique id of item that has been selected/deselected/modified
         // by the user
         IUniqueId currentId = req.getCurrentUniqueId();
         IModelOptionBOMItem item = mob.findItem( currentId);
         graph.selectOption( item );
     }
```

### Variant Management Events - Script PX

The information object for these Events is IVMScriptObj:

- Create Variant Instance
- Derive Variant Model Option BOM
- Update Variant Configuration
- Validate Variant Configuration
- Validate Variant Instance Selections
- Validate Variant Model Option BOM

IVMScriptObj is the interface for the Variant Management events information object. It provides access to the Configuration Graph, the Model Option BOM and the Instance BOM (if available).

> **Note:** You can find more information on the purpose and function of these Events and when they are triggered, in the *Agile PLM Administrator Guide* and the chapter entitled "Configuring Variant Management" in *Agile PLM Product Collaboration User Guide.*

Inherited interfaces - IBaseObjectScriptObj, IBaseScriptObj

The following is a usage example of IVMScriptObj method.

***Example 5–14 Using the IVMScriptObj method***

```
void testIVMEventObj(IBaseScriptObj obj)
{
// get configuration graph
          IConfigurationGraph graph = obj.getConfigurationGraph();
         // get Model Option BOM
         IModelOptionBOM mob =  obj.getModelOptionBOM();
         for ( IConfigurationOption option in graph ) {
           IModelOptionBOMItem mobItem = mob.findItem(
option.getUniqueId() );
           if (option.getQuantity() > mobItem.getMaxQuantity()) {
               obj.addErrorMessage( option, "Quantity must not be greater than
                  max. quanity())
           }
}
```

## Working with Workflow Object Actions

These are Workflow-related actions such as, Change Status for Workflow, Change Approvers or Observers for Workflow, and change the status of Product Cost Management's Sourcing Project.

This section lists and describes the Java PXs that support Workflow object actions.

## Change Status for Workflow

The Information object for this Event is IWFChangeStatusEventInfo.

- **Purpose and function** - IWFChangeStatusEventInfo retrieves changes in Workflow status of the object and assigned notifiers and checks whether the "auto-promote" flag is set or not set.
- **Inherited interfaces** - IRoutingSlipEventInfo , IRoutableObjectEventInfo, IObjectEventInfo, IEventInfo
- **Inherited interfaces purpose and function**:
  - IRoutingSlipEventInfo - **Is the inherited** interface for all Event Information objects that contain the Routing slip Object and provides methods to set/get approvers, observers, comments, and urgent flags
  - IRoutableObjectEventInfo - **Is the inherited** interface for all Event Information objects related to workflow actions and provides methods such as get/set approvers or observers

The following examples use IWFChangeStatusEventInfo, IRoutingSlipEventInfo and IRoutableObjectEventInfo.

***Example 5–15 Using IWFChangeStatusEventInfo***

```
private void testIWFChangeStatusEventInfo(IAgileSession session, IEventInfo req)
throws APIException {
    IWFChangeStatusEventInfo info = (IWFChangeStatusEventInfo)req;

//getNotifiers()
   IDataObject[] notifiers = info.getNotifiers();

// isAutoPromote()
   boolean isAutoPromote = info.isAutoPromote();

// setNotifiers()
   IUser user = getUser(session, "yvonnec");
   IUserGroup ug = getUserGroup(session, "SOA");
   Collection col = new ArrayList();
   col.add(user);
   col.add(ug);
   info.setNotifiers(col);

// getFromStatus()
   IStatus fromStatus = info.getFromStatus();

// getToStatus()
   IStatus toStatus = info.getToStatus();
}
```

***Example 5–16 Using IRoutingSlipEventInfo***

```
private void testIRoutingSlipEventInfo(IAgileSession session,
```

```
  IEventInfo req) throws APIException {
   IRoutingSlipEventInfo info = (IRoutingSlipEventInfo)req;
   Collection colAppvrs = new ArrayList();
   Collection colObsvrs = new ArrayList();
   Collection colNewAppvrs = new ArrayList();
   Collection colNewObsvrs = new ArrayList();
   String newComments = "Override in context object.";

//getApprovers()
   IDataObject[] approvers = info.getApprovers();
   String approverList = arrayToString(approvers);

//getObservers()
  IDataObject[] observers = info.getObservers();

//getComments()
  String comments = info.getComments();

//isUrgent()
  boolean isUrgent = info.isUrgent();

//setApprovers()
   IUser user1 = getUser(session, "badriv");
   IUser user2 = getUser(session, "albertl");
   IUser user3 = getUser(session, "brucec");
   IUserGroup ug1 = getUserGroup(session, "SOA");
   info.setApprovers(colAppvrs);
   IDataObject[] newApprovers = info.getApprovers();

//setObservers()
   info.setObservers(colObsvrs);
   IDataObject[] newObservers = info.getObservers();

//setComments()
info.setComments(newComments);
String latestComments = info.getComments();

//setUrgent()
   boolean newUrgent = getOppositeBoolean(isUrgent);
   info.setUrgent(newUrgent);
   boolean latestUrgent = info.isUrgent();
}
```

***Example 5–17 Using IRoutableObjectEventInfo***

```
private void testIRoutableObjectEventInfo(IEventInfo req) throws APIException {
   IRoutableObjectEventInfo info = (IRoutableObjectEventInfo)req;

   //getWorkFlow()
   IWorkflow wf = info.getWorkFlow();
}
```

## *Approve for Workflow*

The Information object for this Event is ISignOffEventInfo.

- **Purpose and function** - ISignOffEventInfo is the interface for Approve for Workflow and Reject for Workflow. It returns and overwrites information about status, approvers (users), approver groups, and checks the status of the signoff flag.

- **Inherited interfaces** - IRoutableObjectCommentEventInfo, IRoutableObjectEventInfo, IObjectEventInfo, IEventInfo
- **Inherited interfaces purpose and function** - IRoutableObjectCommentEventInfo is the interface for Comment for Workflow. It retrieves and overwrites Comment type data provided for Workflow, Approve for Workflow, and Reject for Workflow Events.

### *Reject for Workflow*

See "Approve for Workflow" on page 5-28.

### *Escalation for Workflow*

The Information object for this Event is `IEscalationEventInfo`.

- **Purpose and function** - `IEscalationEventInfo` retrieves the following information about the Escalation for Workflow Event:
  – Sign-off user, escalated to users, escalation period, and status of the workflow
- **Inherited interfaces** - `IRoutableObjectEventInfo`, `IObjectEventInfo`, `IEventInfo`
- **Inherited interfaces purpose and function** - See "Approve for Workflow" on page 5-28.

## *Reminder for Workflow*

The Information object for this Event is `IReminderEventInfo`.

- **Purpose and function** - `IReminderEventInfo` returns notifiers assigned to the Workflow reminder.
- **Inherited interfaces** - `IRoutableObjectEventInfo, IObjectEventInfo, IEventInfo`

### Audit for Workflow

The Information object for this Event is `IAuditStatusEventInfo`.

- **Purpose and function** - `IAuditStatusEventInfo` retrieves the type of Workflow Audit Action that is performed.
- **Inherited interfaces** - `IAuditResultEventInfo, IRoutableObjectEventInfo, IObjectEventInfo, IEventInfo`
- **Inherited interfaces purpose and function** - `IAuditResultEventInfo` is the interface for Audit for Workflow and Promotion Failure for Workflow. It retrieves error messages for the audit or promotion failure.

### Promotion Failure for Workflow

See "Audit for Workflow" on page 5-31.

### *Comment for Workflow*

The Information object for this Event is `IRoutableObjectCommentEventInfo`. See "Approve for Workflow" on page 5-28.

### Change Approvers or Observers for Workflow

The Information object for this Event is `IChangeAppObserverEventInfo`.

- Purpose and function - `IChangeAppObserverEventInfo` retrieves the action type, status, and the applied to status of the Change Approvers or Observers for the Workflow Event plus the change action by the approver or observer on the Workflow state.
- Inherited interfaces - `IRoutingSlipEventInfo`

## Workflow Object Actions - Script PX

This section lists and describes the Script PXs that support Workflow object actions.

### Change Status for Workflow

The Information object for this Event is `IChangeStatusScriptObj`.

- **Purpose and function** - `IChangeStatusScriptObj` retrieves changes in the Workflow status of the object, the assigned notifiers, and checks whether the "auto-promote" flag is set or not set.
- **Inherited interfaces** - `IRoutingSlipScriptObj , IRoutableScriptObj, IBaseObjectScriptObj, IBaseScriptObj`
- **Inherited interfaces purpose and function**:
  - `IRoutingSlipScriptObj` - **Inherited** interface for all Event Script objects that contain the Routing slip Object
  - `IRoutableScriptObj` - **Inherited** interface for all Event Script objects related to workflow actions

The following examples use the `IChangeStatusScriptObj, IRoutingSlipScriptObj, and IRoutableObjectScriptObj` scripts.

***Example 5–18 Using IChangeStatusScriptObj***

```
IChangeStatusScriptObj: Change Status for Workflow Event script object.
   void testIChangeStatusScriptObj (IChangeStatusScriptObj obj)
{

// getFromStatus()
   String current_status = obj.getFromStatus();

// getToStatus()
   String next_status = obj.getToStatus();

// getNotifiers()
   String notifiers =obj.getNotifiers();

// setNotifiers()
   obj.setNotifiers(["admin", "demo1","weiz", "demo5"]);

// isAutoPromote()
   boolean auto = obj.isAutoPromote();

}
```

***Example 5–19 Using IRoutableScriptObj***

```
IRoutableScriptObj : get workflow
void testIRoutableScriptObj(IBaseScriptObj obj)
{

// getWorkflow()
   String wf = obj.getWorkflow();
}
```

## Approve for Workflow

The Information object for this Event is `ISignOffScriptObj`.

- **Purpose and function** - `ISignOffScriptObj` is the interface for Approve for Workflow and Reject for Workflow. It returns information about status, approvers (users), approver groups, and checks the status of the signoff flag.
- **Inherited interfaces** - `IRoutableObjectCommentScriptObj, IRoutableScriptObj, IBaseObjectScriptObj, IBaseScriptObj`
- **Purpose and function of Inherited interfaces** - `IRoutableObjectCommentScriptObj` is the Interface for Comment for Workflow. It retrieves comments provided for the Routable object.

### *Reject for Workflow*

See "Approve for Workflow" on page 5-28.

### *Escalation for Workflow*

The Information object for this Event is `IEscalationScriptObj`.

- **Purpose and function** - `IEscalationScriptObj` retrieves names of users to whom the Workflow is escalated to.
- **Inherited interfaces** - `IRoutableScriptObj, IBaseObjectScriptObj, IBaseScriptObj`
- **purpose and function of Inherited interfaces** - See "Approve for Workflow" on page 5-28.

### *Reminder for Workflow*

The Information object for this Event is `IReminderScriptObj`.

- **Purpose and function** - `IReminderScriptObj` returns the notifiers assigned to the reminder period and Workflow reminder.
- **Inherited interfaces** - `IRoutableScriptObj, IBaseObjectScriptObj, IBaseScriptObj`

### *Audit for Workflow*

The Information object for this Event is `IAuditStatusScriptObj`.

- **Purpose and function** - `IReminderScriptObj` retrieves the type of Workflow Audit Action that is performed.
- **Inherited interfaces** - `IAuditResultScriptObj, IRoutableScriptObj, IObjectScriptObj, IBaseScriptObj`

- **Inherited interfaces purpose and function** - `IAuditResultScriptObj` is the interface for Audit for Workflow and Promotion Failure for Workflow. It retrieves error or warning messages and Workflow status information for the audit or promotion failure.

#### *Promotion Failure for Workflow*

See "Audit for Workflow" on page 5-29.

#### *Comment for Workflow*

The Information object for this Event is `IRoutableObjectCommentScriptObj`. See "Approve for Workflow" on page 5-28.

### Change Approvers or Observers for Workflow

The Information object for this Event is `IChangeAppObserverScriptObj`.

- **Purpose and function** - `IChangeAppObserverScriptObj` retrieves the action type, status, and the applied to status of the Change Approvers or Observers for the Workflow Event and the change action by the approver or observer on the Workflow state.
- **Inherited interfaces** - `IRoutingSlipScriptObj`

## Working with Specific Object-Based Actions

These actions support incorporating and unincorprating the Item object in PLM's Production Collaboration solution and change status for Sourcing projects.

### Specific Object-Based Actions - Java PX

The related actions are listed and described below.

#### *Incorporate Item and Unincorporate Item*

The Information object for this Event is `IObjectEventInfo`.

#### *Change Status for Sourcing Project*

The Information object for this Event is `ISourcingObjectChangeStatusEventInfo`.

- **Purpose and function** - `ISourcingObjectChangeStatusEventInfo` retrieves the type of action that is performed on the Sourcing object.
- **Inherited interfaces** - `IObjectEventInfo` and `IEventInfo`

### Specific Object-Based Actions - Script PX

The following paragraphs describe the Script PX that supports Specific Object-Based Actions.

#### *Incorporate Item and Unincorporate Item*

The Script Event object for this Event is `IBaseObjectScriptObj`.

#### *Change Status for Sourcing Project*

The Information object for this Event is `ISourcingObjectChangeStatusScriptObj`.

- **Purpose and function** - `ISourcingObjectChangeStatusScriptObj` retrieves the type of action that is performed on the Sourcing object.
- **Inherited interfaces** - `IBaseObjectScriptObj` and `IBaseScriptObj`

## Working with Files and Attachments Objects Actions

These are the file-related actions such as check out file, check in file, and purging version files.

### Files and Attachments Objects Actions - Java PX

The following paragraphs describe the Java PX that supports Files and Attachments objects actions.

#### *Purge File Version*

The Information object for this Event is `IPurgeFileEventInfo`.

- **Purpose and function** - `IPurgeFileEventInfo` retrieves the version for the purged file.
- **Inherited interfaces** - `IObjectEventInfo` and `IEventInfo`

### Incorporate File, Unincorporate File

The Information object for this Event is `IIncorporateFileEventInfo`.

- **Purpose and function** - `IIncorporateFileEventInfo` retrieves the affected File Folders
- **Inherited interfaces** - `IObjectEventInfo` and `IEventInfo`

### Files and Attachments Objects Actions - Script PX

The following paragraphs describe the Script PX that supports Files and Attachments objects actions.

#### *Get File, Check Out Files, Check In Files, Cancel Check Out Files*

The Event Script object for these Events is `IFileEventScriptObj`.

- **Purpose and function** - IObjectEventInfo retrieves the Dirty files for the selected file attachments.
- **Inherited interfaces** - `IBaseObjectScriptObj` and `IBaseScriptObj`

#### *Purge File Version*

The Event Script object for this Event is `IPurgeFileScriptObj`.

- **Purpose and function** - `IPurgeFileScriptObj` retrieves the version for the purged file
- **Inherited interfaces** - `IBaseObjectScriptObj` and `IBaseScriptObj`

The following example shows using `IFileEventScriptObj`.

***Example 5–20 Using IFileEventScriptObj***

```
// FileEventScriptObj: get file for file actions related events
```

```
void testIFileEventScriptObj(IBaseScriptObj obj,IEventDirtyRow row)
{

// getFile()
   file = row.getFile();
   testIEventDirtyFile(obj,file);
}
```

***Example 5–21 Using IEventDirtyFile***

```
// IEventDirtyFile: get file information for Dirty file associated with
// IEventDirtyRowFileUpdate
void testIEventDirtyFile(IBaseScriptObj obj,IEventDirtyfile file)
{

// getFilename()
   String file_name = file.getFilename();

// getSize()
   int file_size = file.getSize();

// getType()
   file_type = file.getType();

// getFileFolderNumber()
   file_folder_num = file.getFileFolderNumber();

// getFileFolderVersionNumber()
   file_folder_ver = file.getFileFolderVersionNumber();

//getCheckoutUser()
   user = file.getCheckoutUser();

// getCheckoutDate()
date = file.getCheckoutDate();
   obj.logMonitor("file name is:" + file_name);
   obj.logMonitor("file size is:" + file_size);
   obj.logMonitor("file type is:" + file_type);
   obj.logMonitor("file folder number is:" + file_folder_num);
   obj.logMonitor("file folder version number is:" + file_folder_ver);
   obj.logMonitor("checkout user is:" + user);
   obj.logMonitor("checkout date is:" + date);
}
```

## Working with Product Governance and Compliance Actions

This action supports checking compliance of PG&C objects.

### Product Governance and Compliance Actions - Java PX

The following paragraphs describe the Java PXs that support Product Governance and Compliance actions.

#### *Compliance Rollup On Object*

The Information object for this Event is IObjectEventInfo. This interface and the applicable inherited interfaces are documented in "General Object Actions - Java PX" on page 5-15.

### Product Governance and Compliance Actions - Script PX

The following paragraphs describe the Script PX that supports Product Governance and Compliance actions

#### *Compliance Rollup On Object*

The Information object for this Event is `IBaseScriptObj`. This interface and the applicable inherited interfaces are documented in "General Object Actions - Script PX" on page 5-18.

## Working with Miscellaneous Object Actions

These are the file-related actions such as check out file, check in file, and purging version files.

### Miscellaneous Object Actions - Java PX

The following paragraphs describe the Java PXs that support the miscellaneous object actions

#### *Transfer Authority*

The Information object for this Event is `ITransferAuthorityEventInfo`.

- **Purpose and function** - `ITransferAuthorityEventInfo` contains the data belonging to Transfer Authority action and retrieves and overwrites data set by the PLM client for the Transfer Authority action.
- **Inherited interfaces** - `IExportEventInfo`.

### Miscellaneous Object Actions - Script PX

The following paragraphs describe the Script PX that supports miscellaneous Object actions.

### Transfer Authority

The Information object for this Event is `ITransferAuthorityScriptObj`.

- **Purpose and function** - `ITransferAuthorityScriptObj` retrieves and overwrites data set by the PLM client for the Transfer Authority action.
- **Inherited interfaces** - `IBaseScriptObj`

## Working with Event Integration Points in PLM Clients

Events can be invoked from Agile PLM clients' Extend Actions Menu, Extend Tools Menu, and Scheduled Event described below.

### Event Integration Points - Java PX

The following paragraphs list and describe the Java PX that support Event integration points in PLM Clients.

#### *Extend Actions Menu*

The Information object for this Event is IObjectEventInfo. This interface and the applicable inherited interfaces are documented in "General Object Actions - Java PX" on page 5-15.

#### *Extend Tools Menu*

The Information object for this Event is IEventInfo. This interface and the applicable inherited interfaces are documented in "General Object Actions - Java PX" on page 5-15.

#### *Scheduled Event*

The Information object for this Event is IEventInfo. This interface and the applicable inherited interfaces are documented in "General Object Actions - Java PX" on page 5-15.

### Event Integration Points - Script PX

The following paragraphs list and describe the Script PXs that support Event integration points in PLM Clients.

#### *Extend Actions Menu*

The Information object for this Event is `IBaseScriptObj`. This interface and the applicable inherited interfaces are documented in "General Object Actions - Script PX" on page 5-18.

#### *Extend Tools Menu*

The Information object for this Event is `IBaseObjectScriptObj`. This interface and the applicable inherited interfaces are documented in "General Object Actions - Script PX" on page 5-18.

#### *Scheduled Event*

The Information object for this Event is `IBaseScriptObj`. This interface and the applicable inherited interfaces are documented in "General Object Actions - Script PX" on page 5-18.

## Guidelines for Java PX and Script PX Handlers

These comments and recommendations are intended to ensure the proper development and implementation of your Java PXs and Script PXs. They include:

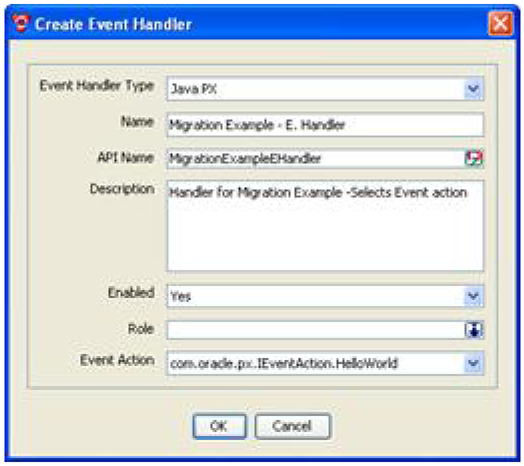
- Information that you need from the Agile PLM Administrator to code the handler and information that you must convey to the Administrator to configure and implement the Event
- Information to ensure proper handling of Events
- Information to test your PXs

### Working with Agile PLM Administrator

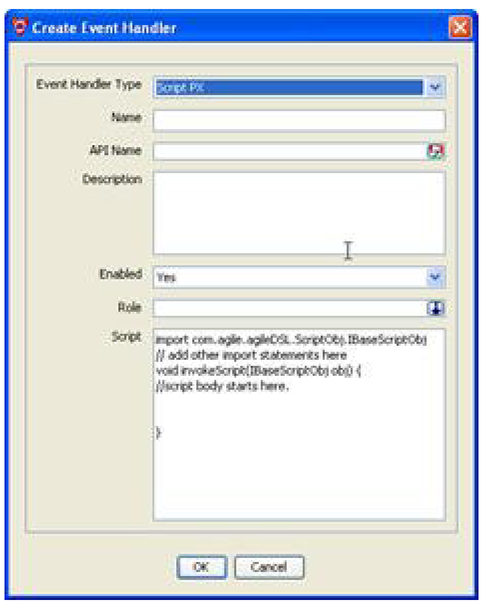
The Agile PLM Administrator will convey the necessary information to determine the Event type, execution mode and trigger type. Once you have developed the Java PX or Script PX handlers, you must inform the Administrator about the Event subscriber. This information includes Event type, execution mode, trigger type, order, and applicable error handling rule.

In addition to information about Event subscriber, the Administrator needs the following specific information to configure the Event Handler.

- **Java PX Handlers -** Java PXs are deployed on the server. For procedures, see "Packaging and Deploying a Custom Autonumber Source" on page 3-3. Once deployed, the Administrator locates it in Java Client by selecting **Admin > Settings > System Settings > Event Management > Event Handlers > New**. The Create Event Handler dialog box appears. In **Event Handler Type** field **select > Java PX > Event Action**. The Administrator needs to know the name that appears in the **Event Action** field, to configure the Handler.




- **Script PX Handlers -** The script PX code is a text file. You must send the script text file to the PLM Administrator. The Administrator will then paste it in the Script field of the Create Event Handler dialog and proceed to configure the Subscriber. This field is accessed from Java Client by selecting **Administrator > Event Management > Event Handlers > Create Event Handler > Script PX > Script**.




### Testing Event Java PX and Event Script PX

If the Agile PLM Administrator uses your Handlers to configure the Event Subscriber, you should coordinate testing the PXs with the Administrator. On the other hand, if you do the configurations, then use the following information for this purpose.

Invoke the new Java PX or Script PX to ensure the action specified in the Handler (code) executes properly.

If the PX is configured to be invoked by a user action from the Tools menu, you can test it in Web Client or Java Client as follows:

- In Web Client tool bar select the Tools and Settings button and then **<Event_name>**.
- In Java Client tool bar select the Process Extensions button or **Tools > Process Extensions >***<Event_name>.*

## Triggering Guidelines for Java PX, Script PX, and Notification Handlers

Use the following guidelines when developing Handlers to ensure proper triggering of the special instances of these Event types.

### General Object Actions

The following paragraphs describe the Java and Script PXs that ensure proper triggering of general object Events, such as Create and SaveAs Events.

#### *Create Object Event and SaveAs Event*

- **Web Client and Java Client behavior** - This action will trigger one Create Object Event (or SaveAs Event) and its Context object will contain all Dirty attributes for Page 1, Page 2 and Page 3.
- **SDK behavior -** This action will trigger one Create object Event (or SaveAs Event). In addition, depending on table attributes that are updated (Page 1, Page 2 and Page 3), one to three Update Title Block Events are triggered. The context object will contain all Dirty attributes belonging to that table.

#### *Update Title Block Event*

**Web Client, Java Client, and SDK behavior** - Depending on table attributes that are updated (Page 1, Page 2 and Page 3), one to three Update Title Block Events are triggered, one for each table. The context object will contain all Dirty attributes belonging to that table.

#### *Update Table Event*

When you add, modify, or delete more than one row, this action has the following Event triggering behavior:

- One Event is triggered and the Context object will contain all applicable rows. The only exceptions are in Java Client:
  - **Relationships Table-** One Event is triggered for each applicable row and the Context Object will contain the applicable row
  - **Attachment Table -** For Update and Delete Actions, one Event is triggered for each applicable row and the Context Object will contain the applicable row

### Workflow Actions

The following paragraphs list and describe the Java and Script PXs that support Workflow-related actions.

#### *Promotion Failure for Workflow Event*

The following actions trigger this Event:

- Failure of Change Status triggered through Relationship

- Autopromotion Failure - Autopromotion Failure is invoked by the following object actions when autopromotion conditions are not met after the completion of these actions. In addition, each triggered Event is tracked in the object History.
  - Change Status
  - Sign off the change
  - Remove Approvers
  - Update Cover Page, Page 2, and Page 3 attributes
  - Edit Affected Item Table, Relationship, File Attachment Tab
  - Cancel checkout attachment
  - Checkin attachment

#### *Create Automatic Transfer Object Action (ATO)*

ATO creation is enabled when ECO status is changed from Pending to Submitted. The sequence of Events that are triggered when the ATO is created is:

1. Pre and Post Event Subscribers are created for:
   - ECO Change status
   - ATO Create
   - ATO Change Status
2. An ECO is created
3. Workflow is assigned and status is changed to submitted

### Files and Attachments Actions

The following paragraphs list and describe when File and Attachment Event are triggered.

#### *Check In File Event*

This action will trigger the following Events:

- Check In File Event on File Folder objects **-** This Event is triggered when the Check In file action is performed on File Folder objects.
- Check In File Event and Update Table Event on Business objects **-** These Events are triggered when the Check In file action is performed on Business objects.

> **Note:** Update Table Event on Business objects is triggered after Check In File Event is triggered. Also, The Event is only triggered if the folder version of the attachment is not set to **[LATEST]**. In Web Client one Event is triggered for all selected rows. In Java Client, one Event is triggered for each selected row.

#### *Check Out File Action*

This action will trigger the following Events:

- Check Out Files Event and Get File Event on File Folder objects **-** These Events are triggered when the Check Out file action is performed on File Folder objects

- Check Out Files Event and Get File Event on Business objects **-** These Events are triggered when the Check Out file action is performed on Business objects in Java Client,
  - If **Download files in one ZIP file** option is selected, one Get File Event is triggered and each row triggers a Checkout Event
  - If **Download files in one ZIP file** option is not selected, each row triggers one checkout Event and each file triggers a Get File event

> **Note:** In Web Client one Event is triggered for all selected rows. In Java Client, one Event is triggered for each selected row.

### Cancel Check Out File Event

This action will trigger the following Events:

- Cancel Check Out Files Event on File Folder objects **-** This Event is triggered when the Cancel Check Out file action is performed on File Folder objects
- Cancel Check Out Files Event on Business objects **-** This Event is triggered when the Cancel Check Out file action is performed on Business objects

> **Note:** In Web Client one Event is triggered for all selected rows. In Java Client, one Event is triggered for each selected row.

### Get File Event

The following actions trigger this Event is:

- Clicking the **Get** button on the Attachments table of Business objects or Files table of File Folder objects
- Clicking **Filename** on the Attachments table of the object

> **Note:** If the file is viewable, Agile Viewer is launched and no event will be triggered.

- Check Out file action

> **Note:** In Java Client, if users choose to download files individually instead of downloading the .ZIP file, one Event is triggered for each selected File.

### Incorporate File

The following actions trigger this Event:

- Clicking the **Incorporate** button on the Affected Files table of DFCO.
- Checking the **In File** action on the Affected Files table of DFCO.

### Unincorporate File

The following actions trigger this Event:

- Clicking the **Unincorporate** button on the Affected Files table of DFCO

# A

# Altering Existing WSX Code to Interact with JAX-WS-Based Framework

This appendix provides background and procedural information to modify existing WSX code to interact with the JAX-WS-based framework of PLM Release 9.3.4 and future PLM releases.

## Introduction

Agile WebService Extensions (WSX) was implemented to enable PLM installations extend the functionality of their PLM systems and expose site-specific solutions as a web service. This framework was based on the Apache eXtensible Interaction System (AXIS) and was available in PLM 9.3.3 and earlier releases.

Starting with PLM Release 9.3.4 and future PLM releases, Agile Web services were updated to use JAX-WS, a technology in Java for building web services to communicate using XML This document provides detailed procedures to migrate existing pre-release 9.3.4 web services to PLM Release 9.3.4.

### Objectives

This document seeks to enable users to:

- Modify existing web services code to interact with JAV-WS technology
- Understand basic concepts of web services using the JAV-WS technology and workflow in this technology
- End to end testing of modified JAX-WS code

### References

You can find useful information about JAX-WS technology at the following sites:

- Apache Axis - `http://ws.apache.org/axis/`
- JAX-WS Reference Implementation - `https://jax-ws.java.net/`

### Pre-requisites

Make sure you have downloaded and installed the following products/components to modify existing WSX code to operate in an Agile PLM Release 9.3.4 environment:

- Agile PLM 934
- JDK 7 or higher
- Apache ANT

## Preparing Project Libraries

To enable web services interact with JAX-WS technology, the JAX-WS-based client and Apache Commons CLI libraries must run the `MyFirstWebService` example. Also, you must copy the new jar file to support `javaws` web service client.

To do this, open the WSX lib folder, and with the exception of the `commons-cli.jar` file, delete all other `.jar` files in the folder. The new.JAR file that you must copy to the lib folder, is `com.oracle.webservices.wls.jaxws-wlswss-client.jar`. You can access this file from `ORACLE_HOME/wlserver/modules/clients/`.

---

**Note:** Alternatively, if you delete the `commons-cli.jar` file, you can download the `commons-cli-1.2-bin.zip` file from Apache's Download Commons CLI sight.

---

## Updating the build.XML File

Use the following procedure to update the `build.xml` file.

**To update the build.xml file:**

1. Open the build.xml file located in the root project folder, and with the exception of `${built.classes.dir}, ${api.jar}, commons-cli.jar`, remove all `.jar` files.
2. Add `com.oracle.webservices.wls.jaxws-wlswss-client.jar` located in the lib folder to `build.classpath`. The ant build path reference is as follows:

```
<path id="build.classpath">
   <pathelement location="${built.classes.dir}"/>
   <pathelement path="${api.jar}"/>
   <pathelement path="${lib.dir}/commons-cli.jar"/>
   <pathelement path=
      "${lib.dir}/com.oracle.webservices.wls.jaxws-wlswss-client.jar"/>
</path>
```

3. Remove the following Axis `taskdef` entries because it is obsolete:

```
<taskdef resource="axis-tasks.properties">
<classpath>
<path refid="build.classpath"/>
<pathelement location="${built.classes.dir}"/>
</classpath>
</taskdef>
```

4. Add the following ant `taskdef` statements to the `build xml`, as shown below.

```
<taskdef name="clientgen"
classname="weblogic.wsee.tools.anttasks.ClientGenTask"
classpathref="build.classpath" />
```

---

**Note:** From an existing WSDL file, the `clientgen` Ant task generates the client component files that client applications use to invoke both WebLogic and non-WebLogic Web Services.
http://docs.oracle.com/cd/E13222_01/wls/docs92/webserv/anttasks.html#wp1075710

---

5. Delete ant target generate-stubs, generate-stubs-file, `generate-stubs-url` from the `build xml` file.
6. Add ant target `generate-stubs` and `generate-stubs-file` as shown below.

```
<target name="generate-stubs-file" depends="init>
   <clientgen wsdl="${wsx.url}/${serviceName}?wsdl"
              destDir="${built.dir}/src"
              type="JAXWS">
   </clientgen>
  <jar jarfile="${built.dir}/MyFirstWebServiceStub.jar">
     <fileset dir="${built.dir}/src" includes="**/*.class"/>
  </jar>
</target>
<target name="generate-stubs" depends="init,generate-stubs-file"/>
```

7. From `${basedir}/custom.properties`, add the property file in `build.xml` to load other necessary data such as the `wsx.url, servicename, username` and so on.

```
<property file="${basedir}/custom.properties" />
```

# Importing the Project to Eclipse Development Environment

This will enable you to use the Eclipse development environment to develop your `MyFirstWebService` example.

### To import the project to Eclipse development environment:

1. In Eclipse, click **File > New > Java Project**.
2. In the New Java Project dialog, type **MyFirstWebService** and in the Location box, type the path to `wsx` in SDK_Samples folder (See the **Note** in"Client-Side Components" on page 2.) If necessary, unzip SDK_samples.zip.
3. In Eclipse, select the **MyFirstWebService** project, and right click the pointer to open the Properties for MyFirstService dialog.
4. In MyFirstWebService dialog, select the **Libraries** tab, followed by the **Add External JARs** button to add the `AgileAPI.jar` and add `com.oracle.webservices.wls.jaxws-wlswss-client.jar, commons-cli.jar` in the lib folder, located in the SDK_Samples folder.

# Migrating the WS Server Code from AXIS to JAX-WS

JAX-WS (a part of the Java Web Services Development Pack) uses annotations that were introduced in Java SE 5 to simplify the development and deployment of web service clients and endpoints. It uses annotations with Web services to configure bindings and handler chains to set names of `portType`, `service`, and other WSDL parameters.

---

**Note:** Annotations are used at build time to map Java to WSDL files and XML schema, and at run time to control how the JAX-WS runtime environment processes and responds to Web service invocations.

---

For MyFirstWebService example, you can use the following statement to declare the `Jaxws` web service.

```
@WebService(portName = "MyFirstWebService", serviceName = "MyFirstWebService",
targetNamespace = "http://xmlns.oracle.com/MyFirstWebService")
```

# Deploying the Web Service in Agile Extensions

Run the ANT target deploy-service to compile and deploy MyFirstWebService to Agile `$AGILE_HOME/integration/sdk/extensions` folder.

Subject to your environment, you can access WSDL from the following sites:

- `http://hostname:port/virtualPath/externsion/MyFirstWebService?wsdl`

# Generating Web Service Client Artifacts

To generate the client-side stubs from existing WSDL, do as follows:

**To generate client-side stubs from existing WSDL:**

1. Create `custom.properties` in project's WSX folder with the following content:

   ```
   wsx.url=http://slc01bem.us.oracle.com:7001/web/extension
   serviceName=MyFirstWebService
   ```

2. Run the ant target `generate-stubs` to generate the client stubs `jar`, `MyFirstWebServiceStub.jar` in the `lib` folder.

# Migrating the Client Code

To use the new generated client-side stubs, import the `MyFirstWebServiceStub.jar` file to Eclipse and open `MyFirstClient.java`. The `getstub()` method shown below initializes the client-side stubs to the desired web service engine.

```
static MyFirstWebService getStub(CommandLine cl) throws Exception {
URL u = generateURL(cl);
MyFirstWebService_Service locator = new MyFirstWebService_Service();
MyFirstWebService stub = (MyFirstWebService) locator.getMyFirstWebServicePort();

// Configure the stub with the necessary authentication information
Map<String, Object> reqContext = ((BindingProvider)stub).getRequestContext();
reqContext.put(BindingProvider.USERNAME_PROPERTY, cl.getOptionValue(USER_SHRT));
reqContext.put(BindingProvider.PASSWORD_PROPERTY,
cl.getOptionValue(PASSWORD_SHRT));
reqContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, u);
```

```
return stub;
}
```

## Modifying the Runner CLI Scripts

The Apache Commons CLI library, provides an API for parsing the command line options parsed to programs. Thus, running this API enables you to modify the two `tmpl` files shown below.

**To modify the tml file for Windows:**

1. Open `runner.bat.tmpl` for windows.
2. Modify the classpath as shown below:

```
set "CLASSPATH=@BUILT.CLASSES.DIR@"
set "CLASSPATH=%CLASSPATH%;@LIB.DIR@/commons-cli.jar"
set "CLASSPATH=%CLASSPATH%;@BUILTLIB.DIR@/MyFirstWebServiceStub.jar"
set "CLASSPATH=%CLASSPATH%;
     @LIB.DIR@/com.oracle.webservices.wls.jaxws-wlswss-client.jar"
```

**To modify the .tml file for Linux/Unix:**

1. Open the runner.sh.tmpl for Liunx.
2. Modifythe `classpath`asfollows:

```
cp=@BUILT.CLASSES.DIR@
cp=$cp:@LIB.DIR@/commons-cli.jar
cp=$cp:@LIB.DIR@/com.oracle.webservices.wls.jaxws-wlswss-client.jar
```

## Running MyFirstWebService Sample on the WebService Client

To run the MyFirstWebService sample, invoke the CLASSPATH initializations information in the `runner.sh` (UNIX) or `runner.bat` (Windows.

**To print out a usage statement for MyFirstClient, type the following command:**

```
>runner client.MyFirstClient
```

**To return the TitleBlock.Description field for part 1000-02, type the following:**

```
>runner client.MyFirstClient -T 15000 -a "attribute_name"
-e virtual_path -h host -l port_no. -n item_number -p password -u username

>runner client.MyFirstClient -T 15000 -a "TitleBlock.Description"
-e Agile -h  -l 80 -n 1000-02 -p agile -u jeffp
```

# B

# Variant Management Configuration Graph Schema

The schema in this appendix describes the structure of the Configuration Graph of an XML document.

## The XML Schema

```
<?xml version="1.0" encoding="windows-1252"?>
<xsd:schema targetNamespace="http://xmlns.oracle.com/Agile/ABO/Configurator"
xmlns="http://xmlns.oracle.com/Agile/ABO/Configurator"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" attributeFormDefault="unqualified">
<xsd:element name="ConfiguratorInitReq"
   type="ConfiguratorInitReqType"/>
<xsd:complexType name="ConfiguratorInitReqType">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The request ABO for getURL service call in Agile getURL Requester ABCS.</b>
<b>ModelID: The model ID for which the init message has to be created.</b>
<b>Organisation: The organisation that is to be used for this BOM.</b>
<b>ReturnURL: The return URL for the termination message if any from client
side.</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="ModelD" type="xsd:string"/>
<xsd:element name="Organisation" type="xsd:string" minOccurs="0"/>
<xsd:element name="ReturnURL" type="xsd:string" minOccurs="0"/>
<xsd:element name="ConfigHeaderId" type="xsd:string" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ConfiguratorInitResponseType">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The Response ABO for getURL service call in Agile getURL Requester ABCS.</b>
SDK Developer Guide - Developing PLM Extensions
132 Agile Product Lifecycle Management
<b>URL: url of the target server.</b>
<b>initializePL: Init message to be posted on the URL as a long String.</b>
<b>any: If the init message is XML.</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="URL" type="xsd:string"/>
<xsd:element name="initializePL" minOccurs="0">
```

```
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="paramName" type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:any minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ConfiguratorBOMType">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The configuratorBOM response type.</b>
<b>AppHeader:The App header for agile specific parameters.</b>
<b>ConfigHeader: Configurator Header</b>
<b>BOM: BOM representing the selected option BOM.</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="AppHeader" type="AppHeaderType" minOccurs="0"/>
<xsd:element name="ConfigHeader" type="ConfiguratorHeaderType" minOccurs="0"/>
<xsd:element name="BOM" type="BomType" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="AppHeaderType">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The header for adding agile specific data.</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="UserID" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ConfiguratorHeaderType">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The header info from Configurator EBO.</b>
<b>ConfigHeaderId: Header ID</b>
<b>ConfigRevision: Confi Revision</b>
<b>ValidConfiguration: Represent ValidConfiguration</b>
<b>CompleteConfiguration: Complete configuration</b>
<b>ExitType: Exit type</b>
<b>PricesCalculatedFlag: Flag to see if the price calculated.</b>
<b>BomQuantity: Quantity of the BOM itself.</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="ConfigHeaderId" type="xsd:string"/>
<xsd:element name="ConfigRevision" type="xsd:string"/>
<xsd:element name="ValidConfiguration" type="xsd:string"/>
<xsd:element name="CompleteConfiguration" type="xsd:string"/>
<xsd:element name="ExitType" type="xsd:string"/>
<xsd:element name="PricesCalculatedFlag" type="xsd:string"/>
<xsd:element name="BomQuantity" type="xsd:int"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BomType">
```

```
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The BOM consisting of different options.</b>
<b>ModelID: Model ID</b>
<b>OrganizationCode: Organisation code for the BOM</b>
<b>ConfigParameters: BomParameters for the BOM</b>
<b>Option: The child Options</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="ModelID" type="xsd:string"/>
<xsd:element name="OrganizationCode" type="xsd:string"/>
<xsd:element name="ConfigParameters" type="BomParameters"/>
<xsd:element name="Option" type="OptionType" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="OptionType">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The option line items.</b>
<b>ItemID: The ItemID</b>
<b>ConfigParameters: The Bom parameteres for this Option</b>
<b>Option: The child Options</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="ItemID" type="xsd:string"/>
<xsd:element name="PositionID" type="xsd:string" minOccurs="0"/>
<xsd:element name="ConfigParameters" type="BomParameters"/>
<xsd:element name="Option" type="OptionType" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="BomParameters">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>The type representing the Bom parameters.</b>
<b>BomTypeCode: Represents BOM type code</b>
<b>Quantity: Quantity used</b>
<b>Uom: Unit of measurement</b>
</xsd:documentation>
</xsd:annotation>
<xsd:sequence>
<xsd:element name="BomTypeCode" type="xsd:string"/>
<xsd:element name="Quantity" type="xsd:float"/>
<xsd:element name="Uom" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="ABCSStatus">
<xsd:annotation>
<xsd:documentation xml:lang="en">
<b>Status texts for responses</b>
</xsd:documentation>
</xsd:annotation>
<xsd:restriction base="xsd:string">
<xsd:enumeration value="SUCCESS"/>
<xsd:enumeration value="FAILURE"/>
<xsd:enumeration value="WARNING"/>
<xsd:enumeration value="ERROR"/>
<xsd:enumeration value="FATAL"/>
</xsd:restriction>
```

```
</xsd:simpleType>
</xsd:schema>
```

# C

# Groovy Implementation in Event Framework

This appendix describes the Groovy scripting language and sources of information about this tool. Other topics addressed, include procedures to start a script, access the SDK using scripts, and sample use cases.

## What Is Groovy?

Groovy is an object-oriented programming language that can be used as a scripting language for the Java Platform.

### Sources of Information

World Wide Web provides links to many sites that offer information about Groovy. Publishers and vendors of the print media also offer information on this tool. A few are listed below.

- From World Wide Web:
  - **Groovy Home** - Provides links to documentation, downloads, tutorial, user guide, Eclipse plugin examples, advanced usage guide, and other sites maintained by Groovy Home **(**`http://groovy.codehaus.org/`**)**
- From publishers and vendors:
  - **Publisher: Manning Publications** - Groovy in Action by Dierk Koenig, Andrew Glover, Paul King, and Guillaume Laforge
  - **Publisher: Morgan Kaufmann** - Groovy Programming: An Introduction for Java Developers by Kenneth Barclay and John Savage

### Script PX or Java PX?

Scripts are suitable for rapid development and deployment of applications with simple business logic. They empower Agile PLM Administrators and power users to develop extensions unique to their requirements and make rapid modifications when necessary. Scripts are not suitable for developing complex applications with performance critical data structures.

Use scripts to:

- Automate functions with simple business logic such as data validation, notification, or defaulting field values
- Implement unique customization for existing applications
- Build extensions to existing systems

- Rapid prototyping
- Write test use cases

# Event Framework Implementation

Event framework implementation requires running the scripting engine. Key implementation considerations are summarized below.

## Key implementation considerations

- The scripting engine runs inside the Java 2 Platform Enterprise Edition (J2EE) on the Agile PLM server and is based on the Groovy language.
- Groovy is fully embedded in Event framework.
- Groovy is the only supported scripting language.
- Script codes are currently stored in CLOB fields in the Agile PLM database.
- Event Script objects (Event Script PX handlers) that you develop are text files that are deployed from the Event Management Node by selecting **Event Management > Event Handlers** in Java Client.
- Scripts can call the Script API and Agile SDK.
- The *Agile PLM Administrator Guide* provides both background information and sample procedures to understand and manage Events.

## Starting a Script

1. Use void invokeScript(IBaseScriptObj obj) to start your script. InvokeScript is script's starting point of the execution and IBaseScriptObj is the base interface for all Event Script Objects.
2. The run time type of the Obj is dynamically resolved based on the type of Event that invoked this script. For example, if the script is invoked on a Create Event, then ICreateScriptObj is dynamically resolved as the type of Obj at run time. You can invoke any method defined on ICreateScriptObj and its super-interfaces on the instance of this Obj.
3. Use IAgileSession getAgileSDKSession() to access the SDK session and invoke SDK functions.
4. Use AgileDSLException to return “exception” information from scripts.

## Accessing SDK with Scripts

You can access an SDK session and Java PX Event object when writing SDK programs in a script.

```
/**
* Returns agile SDK session.
* @return agile SDK session
* @throws AgileDSLException
* if the method fails
* @since \@Agile93@
*/
public IAgileSession getAgileSDKSession() throws AgileDSLException;

/**
* Returns PX eventInfo.
```

```
* @return PX eventInfo
* @throws AgileDSLException
* if the method fails
* @since \@Agile93@
*/
public IEventInfo getPXEventInfo() throws AgileDSLException;
```

## Use Cases

You can find Script PX Handler examples for most Event types in "Client-Side Components" on page 1-2. These handlers use the Pre/Post triggers and Synchronous/Asynchronous executions and cause different actions when they are invoked.

# Glossary

### ACP

See **Agile Configuration Propagation (ACP)**

### Activity

A project activity in Agile Product Portfolio Management, such as a program, task, or phase.

### Affected Files

Similar to Affected Items, these objects are EC files that are Design Release Candidates.

### Agile Configuration Propagation (ACP)

Propagating existing configuration the PLM to the newly installed version of PLM.

### ACS

See **Agile Content Service (ACS)**

### Agile Content Service (ACS)

ACS is an event-driven XML-based publishing service that makes the product record available to a wide variety of business applications and users, internally and across the global manufacturing network

### Agile Destination

A package created by an Agile PLM system in the target PLM using Web Services to import from the Attachments tab of the package in the target system.

### Agile Integration Services (AIS)

A collection of predefined Web Services in the Agile Integration Framework that enable communication between the Agile Application Server and disparate systems

### Agile Product Portfolio Management

The Agile PLM project management solution that is integrated with the product information in PLM.

### AI

Affected Items tab on Change objects in Agile.

### AIS

See **Agile Integration Services (AIS)**.

### Approved Manufacturer Parts List (AML)

List of approved manufacturer parts associated with an item.

### AML

See **Approved Manufacturer Parts List (AML)**.

### API

See **Application programming interface (API)**.

### Application programming interface (API)

A set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types.

### Assembly

A product assembly lists the parts in a product and shows the substances and materials that comprise those parts. It is linked to specifications that can restrict how much of a particular substance that product assembly may contain

### Automated transfer orders (ATO)

Content published by Agile PLM users in real time with a content transfer order (CTO) or set up subscribers to automatically create automated transfer orders (ATO) based on a schedule or triggered by a workflow status change.

### Baseline

A snapshot of a project, usually in its initial stage, used as a reference for future comparison in Agile Product Portfolio Management.

### Bill of Material (BOM)

A hierarchical representation of a product that is made up of other products.

### Bill of Substances (BOS)

A hierarchical list of substances that are contained in the parts and assemblies that make up a BOM.

### BOM

See **Bill of Material (BOM)**.

### BOS

See **Bill of Substances (BOS)**.

### CAD

See **Computer-aided design (CAD)**

### Commodity

A class of goods that is in demand, that is supplied without qualitative differentiation regardless of supplier.

### Computer-aided design (CAD)

The use of computer systems to assist in the creation, modification, analysis, or optimization of a design. CAD software is used to increase the productivity of the designer, improve the quality of design, improve communications through documentation, and to create a database for manufacturing. CAD output is often in the form of electronic files for print, machining, or other manufacturing operations.

### Contract (Price)

This is a subclass of the Published Prices class. Contract prices are prices provided by the supplier for a specific item or manufacturer part. This price information applies only for the specified duration and can apply to any project.

### Co-Sourcing

The process of leveraging product cost across suppliers.

### DCO

See **Design Change Order (DCO)**

### Deliverable

A unit of work required for a project's success, usually fulfilled by generating a digital file. (Word processing documents, spreadsheet documents, PDFs, presentation documents, and so on.) Deliverables can also be Agile PLM objects and processes. Also called 'content' in Agile Product Portfolio Management.

### Design Change Order (DCO)

A Change Order subclass that is available when the effected File Tab is enabled and provides access to all Agile PLM Workflow functions.

### Design File Folder

An EC file folder that is integrated with CAD and PLM files, providing full access to PLM Workflow function.

### EC

See **Engineering Collaboration (EC)**

### EC Client

A Java-based UI to access, administer and operated the EC solution.

### ECO

See **Engineering Change Order (ECO)**

### Engineering Change Order (ECO)

An object that carries with it all the proposed changes to a product and/or its BOM. When approved and implemented, the proposed changes become effective.

### Engineering Collaboration (EC)

An application that provides data and process integration between CAD applications and Agile PLM. It allows CAD designers and engineers to capture and control the data representing a primary source of the product record.

### Extensible Markup Language (XML)

A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

### File Manager

The File Manager manages files in a repository or vault in the file system and provides a place to store and retrieve files locally or remotely. You can install it on the same server as the Agile Application Server or on a separate one. You can also install the File in a redundant configuration and/or distributed across geographic regions.

### File Transfer Protocol (FTP)

A standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet.

### FQPN

See **Fully qualified path name (FQPN)**.

### FTP

See **File Transfer Protocol (FTP)**

### Fully qualified file name

The exact name of a file on a computer that is completely specified such that it is unambiguous and cannot be mistaken for any other file on that system.

### Fully qualified path name (FQPN)

The full path of a resource, directory or file, stored in a computer. It is composed by the full path to the resource and its syntax depends on the operating system.

### Gantt Chart

A project management tool that shows project activities and schedule as a bar chart. The chart lists project activities in sequence, and presents critical information such as the start and end dates of each activity, as well as interdependencies between activities.

### Item Master

The product record. It is the entire collection of Items - Parts, Documents, and any other user-defined subclasses of the Items class maintained under change control in the Agile system.

### Java Message Service (JMS)

The Java Message Service (JMS) API is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients.

### JMS

See **Java Message Service (JMS)**.

### Lifecycle Phase

Current state in an object's workflow.

### LRR

Latest Released Rev - concerning a Part or Document.

### NCNR

Non-Cancelable Non Returnable. Applies to an item. NCNR can be a Yes or No, depending on the supplier. You can ask for the NCNR information in the supplier response. This is one of the critical factors in finding the best deal among the supplier responses.

### PDX

See **Product Definition eXchange (PDX)**.

### PLM

See **Product Lifecycle Management (PLM)**.

### Percent allocation or % allocation

The percentage of a resource's time allocated to a specific task or tasks in Agile Product Portfolio Management..

### Percent complete or % complete

Amount of time and effort expended on a project measured as a percentage of the time and effort required to complete the whole project. Used in Agile Product Portfolio Management.

### Product Definition eXchange (PDX)

A standard designed for the e-supply chain. This standard is based on the XML format because it provides a simple yet powerful and flexible way to encode structured data into a format that is both human- and computer-readable. In PLM, PDX packages contain product content, such as items.

### Product Lifecycle Management (PLM)

The process of taking parts/documents from inception to production to phase-out, and all the stages in between.

### Protocol

A system of digital rules or agreed-upon format for data exchange within or between devices. It determines the type of error checking and data compression used.

### Published Price

This is a subclass of the Published Prices class. Published prices are prices provided by the suppliers in response to an RFQ and published from the project. The published price information can also be used in other projects.

### PCO

See **Price Change Order**

### Price

An object that carries with it all the proposed changes to a product and/or its BOM. It can be approved and implemented to make the proposed changes effective.

### Price Change Order

It is an object that carries with it all the proposed changes to a price. It can be approved and implemented to make the proposed changes effective.

### Quote history

A subclass of the Quote Histories class. Quote history prices are the stored prices from supplier responses that you can use. Any change in the response line of an RFQ is stored in the historical response and is usable at any time.

### Request for Information (RFI)

A material declaration that lists the parts in a product assembly and shows the substances and materials contained in the part.

### Request for Quote (RFQ)

A standard business process whose purpose is to invite suppliers into a bidding process to bid on specific products or services.

### Request for Proposal (RFP)

A solicitation, often made through a bidding process, by an agency or company interested in procuring a commodity, service or valuable asset, to potential suppliers.

### Response Line

A response line has information about only one item. The negotiation of price and terms for items is dealt with in a response line.

### Resource Pool

A group of users who can be bulk assigned as resources for a particular project or task in Agile Product Portfolio Management.

### RFI

See**Request for Information (RFI)**.

### RFP

See **Request for Proposal (RFP)**

### RFQ

See **Request for Quote (RFQ)**

### RFQ Response

A medium of communication between the user and the supplier. One response from a supplier can contain multiple response lines for different items. Price data is added to the project automatically when the supplier submits the response.

### Schedule Editor

The scheduling engine that handles updates to the project schedule in Agile Product Portfolio Management.

### Schema

In computer programming, a schema is the organization or structure for a database. The activity of data modeling leads to a schema.

### SDK

See **Software Development Kit (SDK or "devkit")**

### Software Development Kit (SDK or "devkit")

A set of software development tools that allows the creation of applications for a certain software package, software framework, hardware platform, computer system, video game console, operating system, or similar development platform.

### Sourcing Project

The entry point of sourcing and product pricing. A sourcing project tracks data required for sourcing and pricing, to perform data analysis for effective pricing.

### Standard Cost

Applies to an item. This is the market cost of the item. It is site-specific. The standard cost is for a unit.

### Supplier

A supplier of one or several commodities.

### Target Cost

Applies to item. This is the expected cost of the item by you or the supplier. This can be a percentage of the standard cost. Target cost is for a unit.

### Timesheet

The time entry system in Agile Product Portfolio Management, used to track actual hours spent by resources on project activities and to calculate corresponding labor cost.

### TLA

See**Top Level Assembly (TLA)**

### Top Level Assembly (TLA)

The level in a BOM that indicates the ultimate product being manufactured.

### Transfer order

Every time Agile Content Service (ACS) publishes product content, it produces a transfer order that keeps track of what, where, and when product content is transferred.

### UPK

See **User Productivity Kit (UPK)**

### User Productivity Kit (UPK)

The Oracle online help system used in some Oracle products.

### Web Service Extensions (WSX)

A Web service engine that enables communication between Agile Product Lifecycle Management system and disparate internal and external systems.

### WSX

See **Web Service Extensions (WSX)**.

### XML

See **Extensible Markup Language (XML)**.

### XML Schema

Description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by XML rules.