Agile Product Lifecycle Management

AIS Developer Guide

Release 9.3.3

E39309-02

December 2013

ORACLE

Agile Product Lifecycle Management AIS Developer Guide, Release 9.3.3

E39309-02

Primary Author: F. Tabibzade

Contributing Author:

Contributor:

Contents

3 Exporting Data

4 Importing Data

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

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:

- Understanding AIS
- Web Services Operations
- Web Services Extensions
- Security Considerations

Understanding AIS

The Agile Integration Services (AIS) are a collection of predefined Web Services in the Agile Integration Framework that enable communication between the Agile Application Server and disparate systems including:

- Enterprise Resource Planning (ERP) systems
- Customer Resource Management (CRM) systems
- Business-to-Business Integration systems (B2B)
- Other Agile PLM systems and supply chain partners

Using AIS to exchange content with other systems simplifies the process for aggregating unprocessed product content, and makes critical product content available in real time to other systems. AIS Web Services also provide import and export capabilities that you can use to:

- Make product content available to Enterprise Application Integration (EAI) systems
- Share product content with product design, manufacturing planning, shop floor, and ERP and CRM applications
- Make product content available to B2B systems that can transfer Agile Application Server data across corporate boundaries to external applications
- Provide content to custom applications
- Import product content data from ERP and other supply-chain applications

Key Features

Key features supported by AIS include:

- Programmatic access to data -- AIS supports programmatic access to data stored in Agile PLM systems.

- Programmatic validation of imported data-- Using the Agile SDK methods, AIS enables checking the imported data for compliance with server rules before they are actually imported.
- Product content accessibility -- AIS provides accessibility to Agile product content outside of corporate firewalls using standard HTTP(S) technology.
- Apache Axis support -- Agile Web Service Extensions (WSX) are based on the *Apache eXtensible Interaction System* (Axis) packages and Agile certifies client applications that use the Axis package client.

> **Note:** Export and Import attachment types are not compatible with the .Net attachment types.

- Multiple output format support -- AIS supports the aXML and Product Data eXchange (PDX) 1.0 formats (PDX is defined in Web Services Operations).
- Internet Security safeguards -- AIS communicates with XML-compliant applications using Internet-standard communication and security protocols (HTTP and SSL), for a secure and firewall-friendly interface.
- Multilevel BOM support -- AIS supports converting a multilevel BOM into individual parts data in the XML format.

AIS Architecture

Connection to AIS is established using standard Web service invocation methodologies shown in the following illustration.

AIS Folders

AIS documentation and source files for sample clients are in the AIS_samples.zip folder. You can find this folder in Oracle Oracle Technology Network at: http://www.oracle.com/technetwork/indexes/samplecode/agileplm-sample-520945.html. For more information, contact your System Administrator, or refer to your Agile PLM installation Guide.

The AIS_samples.zip file includes the following folders:

- documentation - Documents the Web Services that are supported by the Agile Import/Export APIs.
- lib - Contains the common JAR files used by AIS samples.
- sample - Contains the source code of a sample Import/Export Web service client.

Understanding AIS Web Services

AIS Web Services is a technology for building distributed applications. These services, which are 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, you 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 (RPC) across the Internet. It uses:

- HTTP(S) or other transport protocols such as HTTP to transport requests and responses.
- Simple Object Access Protocol (SOAP) to communicate the request and response information.

The key benefits of Web Services are:

- Service-oriented Architecture (SOA) -- Unlike packaged products, Web Services can be delivered as streams of services that allow access from any platform. SOAs are a new approach to enterprise application integration by building applications from software services.
- Interoperability -- Web Services ensure complete interoperability between systems.
- Integration -- Web Services facilitate flexible integration solutions, particularly if you are connecting applications running 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.

Web Services have certain limitations of the technology as it exists today, for example, supported software and specifications, problems using certain versions of applications or tools. In view of these constraints, consider the following when developing Web service applications:

- When developing Web service applications, avoid advanced operations such as distributed garbage collection, object activation, or call by reference. SOAP as a simple mechanism for handling data and requests over a transport medium is not designed to handle these operations.
- Web Services are network-based and are therefore affected by network traffic. The latency for any Web service invocation can often be measured in hundreds of milliseconds. Thus, the extent of functionality provided by the service must be significant enough to warrant making a high-latency call.
- Web Services do not work well with *conversational programming* languages. Thus, when designing services that you want to expose, make the service as independent as possible from these development tools.

Web Services Architecture

Web services architecture is best defined in terms of its roles and protocol stack:

- Web service roles:
  - Service provider - The provider of the service by Implementing and making it available on the Internet.
  - Service requester - 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 service 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 to exchange 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 allows description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.
- A WSDL document defines services as collections of network endpoints (called ports). A port is defined by associating a network address with a reusable binding. A collection of ports defines a service.
  - Service discovery layer - Centralizes services into a common registry by using the Universal Description, Discovery, and Integration (UDDI) protocol.

Note: AIS Web services do not currently support UDDI or other service discovery layers.

Web Services Operations

AIS enable you to export data programmatically into structured XML documents and import data into the Agile PLM system with the following prebuilt Web service operations:

- exportData -- A Web service operation that extracts data from an Agile PLM system. The exportDataRequest element encapsulates all the information needed to extract data from an Agile PLM system. The ExportData Web service operation supports the following formats:
  - Product Data eXchange (PDX) -- A standardized XML format for representing structured product content. It provides a means for partners in the e-supply chain (OEMs, EMS providers, and component suppliers) to exchange product content and changes (BOMs, AMLs, ECRs, ECOs).

For more information about PDX, including a link to DTD, visit http://webstds.ipc.org/2571/2571.htmhttp://webstds.ipc.org/2571/2571.htm.

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

The list and location of Agile aXML schema files for different releases of the application are available at the following sites:

- Release 8.5 - http://www.oracle.com/technology/products/applications/xml/plm/2003/12/aXML.xsd
- Release 9.0 - http://www.oracle.com/technology/products/applications/xml/plm/2004/02/aXML.xsd
- Release 9.0SP4 - http://www.oracle.com/technology/products/applications/xml/plm/2004/12/aXML.xsd
- Release 9.1 - http://www.oracle.com/technology/products/applications/xml/plm/2004/10/aXML.xsd
- Release 9.2, 9.2.0.x - http://www.oracle.com/technology/products/applications/xml/plm/2005/11/aXML.xsd
- Release 9.2.1, 9.2.1.x - http://www.oracle.com/technology/products/applications/xml/plm/2006/03/aXML.xsd
- Release 9.2.2, 9.2.2.1, 9.2.2.2, 9.2.2.3 - http://www.oracle.com/technology/products/applications/xml/plm/2007/03/aXML.xsd
- Release 9.2.2.4, 9.2.2.5, 9.2.2.6, and 9.2.2.7 - http://www.oracle.com/technology/products/applications/xml/plm/2008/05/aXML.xsd
- Release 9.3 and 9.3.0.1 - http://www.oracle.com/technology/products/applications/xml/plm/2009/06/aXML.xsd
- Release 9.3.1 and 9.3.1.1 - http://www.oracle.com/technology/products/applications/xml/plm/2010/09/aXML.xsd
- Release 9.3.1 and 9.3.1.1 - http://www.oracle.com/webfolder/technetwork/xml/plm/2010/09/aXML.xsd
- Release 9.3.2 - http://www.oracle.com/webfolder/technetwork/xml/plm/2011/09/aXML.xsd
- Release 9.3.3 -http://www.oracle.com/webfolder/technetwork/xml/plm/2013/09/aXML.xsd
- exportPartList -- A Web service operation that takes a multilevel BOM and "flattens" it into a list of the items and associated manufacturer parts, and their quantities in the BOM; it then returns the data in aXML format. That is, it enables you to extract a rolled up set of parts, and the related Quantities Per Top Level Assembly (QPTLA). The exportPartlistRequest element encapsulates all the information needed to extract a flattened partlist from an Agile PLM system.

> **Note:** The value of the QPTLA is computed as the sum over recursive products starting from the top of the BOM tree. exportPartlist calculates the QPTLA for each unique item-revision pair, and returns the results in the Part Quantities element of the resulting aXML output.

- `importData` -- A Web service operation that imports data into the Agile PLM system. The `importDataRequest` element encapsulates all the information needed to request an import operation. The source for the import data can be an Agile database, a third party Product Data Management (PDM) system, or an Enterprise Resource Planning (ERP) system. The Agile server stores information about customer-specific items. It also maintains the relationships that assembly parts have with BOM components and that parent items have with approved manufacturers.
- `importSupplierResponse` -- A Web service operation that imports an RFQ response from a supplier. The response is associated with an existing RFQ in the Agile PLM system.

> **Note:** The `importSupplierResponse` Web service operation is deprecated and may not be supported in future releases. Use `importData` instead. For more information, see "Importing Supplier Responses" on page 4-8.

These Web service operations are invoked by submitting a properly formatted XML document to AIS. The contents of the XML document define the parameters that determine how the operation should be performed. For more information about using the prebuilt AIS Web Services, see Chapter 2, "Using AIS Web Services."

Web Services Extensions

You can use the Agile SDK to develop Web service extensions (WSX) that leverage the functionality of AIS while extending the functionality of the Agile PLM system. For example, if you need to extract data from the Agile server and then transform it before sending it to another ERP system, you could create a custom Web service that leverages the Export web service. For more information about web service extensions, refer to the *Agile SDK Developer Guide*.

Security Considerations

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

User name and password security is enforced whenever a client attempts to invoke an Agile Integration Service operation.

For more information about Java security and cryptography support, refer to the following Web page:
`http://docs.oracle.com/javase/1.5.0/docs/guide/security/index.html`.

2

Using AIS Web Services

This chapter includes:

- Tools that you can use to develop client applications
- Languages that can generate and process XML and process HTTP request/response messages
- General steps to access the prebuilt AIS Web Services (The examples in the `AIS_sample` folder illustrate these steps.)

Tools

There is no single tool set to access Web Services. Tools that you choose are a function of the environment that you use to develop your clients. Essentially, you need tools to enable generating and processing XML files and HTTP request/response messages.

Although AIS is based on Axis, which is a SOAP processor, you can use other implementations of SOAP tools regardless of the source language to build Web service clients.

Client Programming Languages

Oracle recommends, tests, and certifies Java to develop AIS clients.

WSDL is only supported with J2EE.

This is a list of some of the Web sites where you can find more information about these development tools:

- Apache Axis -- Open source SOAP implementation for Java. See the following Web site: `http://ws.apache.org/axis/`
- GlassFish Application Server -- GlassFish is a complete Java EE 5 Application Server. See Java Web Services Developer Pack at `http://glassfish.java.net/public/downloadsindex.html`.
- Microsoft .Net -- An XML Web Services platform for Microsoft Windows that can be used to create Web service clients. See the following Web site for more information: `http://msdn.microsoft.com/en-us/netframework/`

AIS was only tested with Java. Other tools and environments such as VB .Net should work, but AIS was not tested and is not supported for these tools and environments (operability with .Net (Visual C#) was tested with an earlier version of AIS).

- SOAP::Lite for Perl -- A Perl implementation of the SOAP protocol. See the following Web site: `http://www.soaplite.com/`

Accessing AIS Web Services

In general, to access AIS Web Services, you need to:

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 AIS via HTTP or HTTPS -- Once an appropriate set of client-side Java *stubs* are 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 AIS samples provide comprehensive examples of how SOAP and Web service-related libraries can make this process simple. The following sections illustrate using the `ExportData` sample in "Running the Java Samples" on page 2-5 and the above steps in greater detail.

Checking Your AIS System

Before compiling and running the AIS Web service client samples, make sure the clients are functioning properly on the Agile PLM Application Server. For more information, refer to the *Agile PLM Installation Guide*.

About AIS Java Samples

AIS provide several Java Web service client samples for you to download. These samples use Axis to connect with the AIS Web service engine to generate client-side stubs. You can use these sample clients to export and import data. They provide command-line interfaces to the `ExportData, ExportPartlist`, and `ImportData` Web service operations.

These AIS Java samples do not expose all AIS functionalities. They are only sample clients. For example, they enable running only one query at a time, while AIS supports runing multiple queries and then aggregating the results. You may choose to develop AIS clients with additional functionality. The samples provide source code that you can use to practice developing your own AIS clients. For more information about functionalities supported by the Export and Import Web Services, refer to the Export and Import XML Schema documentation in Agile PLM's *Import/Export User Guide*.

Before building and running the AIS samples, download the following required tools:

- Java 2 SDK SE Version 1.5. You can download this software at: `http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-javase5-419410.html`
- The Apache Project's Ant build tool, version 1.6, available at: `http://ant.apache.org/`

Installing the Java SDK

This section provides 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.cshrc` or (depending on your shell) file to set up your environment.

Installing Ant

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

To install 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-1.6.0-bin.zip`).
2. Open a command prompt window and verify that Ant can be invoked by entering the following command: `%ANT_HOME%\bin\ant -version`

   The following output should be displayed: `Apache Ant version 1.6.0` compiled on <date>

To install the Ant on Solaris:

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

   The ANT distribution for UNIX is a tar file (for example, `apache-ant-1.6.0-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 entering the following command: `$ANT_HOME/bin/ant -version`

   Upon successful installation, the following message appears:

   `Apache Ant version 1.6.0 compiled on <date>.`

Building the Java Samples

Building the Java samples is straightforward. You need the Ant build tool, which is available for download at: `http://ant.apache.org/`. For procedures, see "Installing Ant" on page 3. Run Ant within the samples directory, pointing the URL to your AIS installation.

If you generated client stubs for the AIS samples from the WSDL, they will run on any other computer. Alternately, if you have the WSDL, you can use it to generate the client stubs on another computer.

To build the AIS Java samples on Windows:

1. Download wsdl4j-1.5.1.jar from http://archive.apache.org/dist/ws/axis/1_2/.
2. Copy the contents to the ais/lib folder and rename it to `wsdl4j.jar`.
3. Open a command prompt window and navigate to the AIS samples folder, which contains the file build.xml.
4. Type the following command:

   ```
   %ANT_HOME%\bin\ant -Dais.url=https://<hostname:port/virtualPath>/ws -Dusername=
   <username> -Dpassword= <password target>
   ```

   Where:

   - `hostname` - This is name of the Agile server.
   - port - This is the application server port. If you are using an Oracle Application Server to host the Agile PLM system, type 7777. If you are using WebLogic Server, type 7001
   - target - This identifies the AIS sample to build. Available build targets are export, import, and all. The default target is all. If you do not specify a target, all AIS samples will be built.
   - username - This is the Agile PLM user name
   - password - This is the Agile PLM password
5. After you build the samples, use the *runner* file in the AIS samples directory to run the samples. It contains all the necessary CLASSPATH initializations for the samples

   Note: Agile PLM requires both `username` and `password` to build the Java samples. The *makefile* execution will fail if the three parameters are not set.

To build the AIS Java samples on Solaris:

1. Navigate to the AIS samples directory and locate the file build.xml and then type the following command:

   ```
   $ANT_HOME/bin/ant -Dais.url=http://<hostname:port/virtualPath/ws target>
   ```

2. After you build a sample, use the runner file in the AIS samples directory to run the sample. This file contains all the necessary CLASSPATH initializations for the samples. For more information, refer to the comments (in Javadoc) for each sample.
3. If you are connecting to a secure URL that uses SSL, type the following command instead. For descriptions of hostname, port, virtualPath, `username`, `password`, and `target`, see the previous section

   ```
   $$ANT_HOME/bin/ant -Dais.url=http://<hostname:port/virtualPath>/ws
   -Dusername=<username> -Dpassword=<password target>
   ```

To build Java AIS samples on servers with Secure Sockets Layer (SSL) enabled:

1. Get the self-signed certificate from the server.
2. Install the self-signed certificate into your Java development environment.
3. Build the sample programs as described above by connecting to the server using HTTPS.

4. Run the sample programs as usual but include the command line parameter -P. For example:

```
runner importer.ImportData -P HTTPS <insert other parameters here>
```

   The `Readme.txt` file that is installed with the AIS samples includes more information about obtaining a certificate, installing it in your Java environment, and building and running the AIS samples on an SSL-enabled system.

5. After you build the samples, use the `runner` file in the AIS samples directory to run the samples. This file contains the necessary CLASSPATH initializations for the samples.

Running the Java Samples

Depending on your operating environment (Windows or Solaris), once you perform the build, one of the following files will appear in the AIS samples directory:

- On Windows, the file is runner.bat
- On Solaris, the file is runner.sh

These files contain the necessary CLASSPATH initializations and you can use them to simplify the process of invoking a sample application.

The invocations below will print out usage statements for each of the examples. You can use these usage statements along with the additional documentation provided on the samples to determine how to run the samples in a meaningful fashion.

To print out usage statements for the clients, type the following commands:

```
> runner export.ExportData
> runner export.ExportPartlist
> runner importer.ImportData
> runner importer.ImportSupplierResponse
> runner importer.ValidateData
```

export.ExportData Usage

Usage: export.ExportData <options>

Option	Description
-a axml	This selects the aXML output format instead of the default PDX output format.
-c criteria	This is the search criteria for locating objects to export. The criteria must be formatted using the Agile SDK query language. For more information, refer to *Agile PLM SDK Developer Guide*.
-e virtual-path	This is the Agile PLM virtual path. For example, if you access Agile Web Client via `http://www.sample.com/Agile/PLMServlet`, the virtual path is "Agile". When you install the Agile PLM system, the default virtual path is "Agile".
-f filter	This is the predefined filter name or ID. If you have administrator privileges, you can define Agile PLM filters using the Agile Java Client.

Option	Description

-F filter-flag	This is the ad hoc filter flag. The legal values for this argument derive from the <filters> element shown in the Export XML Schema documentation. The filter flags correspond to child elements with names ending in "Filter," like `ChangeFilter` and `ItemFilter`. The basic pattern for this option is *filter-name.attribute.value* where*:*
	
	filter-name corresponds to the name of the XML element, such as `ItemFilter` (the "`Filter`" suffix may be omitted).
	
	attribute corresponds to the name of the attribute that is defined (for example, "PageTwo").
	
	value corresponds to the value for the attribute. If the attribute is a boolean, the value is optional and defaults to "true." For the Attachments attribute, the value "Tables and Files" causes the attachment table and all the referenced files to be exported. Following is an example of a filter flag:
	
	`-F "Item.TitleBlock" "Item.Attachments.TableAndFiles" "Item.BOM.Recurse"`
	
	If you are extracting data to PDX files, the filter flag should be a superclass filter such as `ItemFilter or ChangeFilter`. In the following example, `ChangeFilter` is used.
	
	`runner export.ExportData -h agilesvr -l 7001 -u aisuser -p agile -t ECO -c "[Number] is not null" -F "ChangeFilter.CoverPage" -o eco.pdx`
	
	In aXML files, the filter flag must be a class filter such as `PartFilter` or `ECOFilter`. In the following example, ECOFilter is used.
	
	`runner export.ExportData -h agilesvr -l 7001 -u aisuser -p agile -t ECO -c "[Number] is not null" -F "ECOFilter.CoverPage" -o eco.axml -a axml`
	
	For a complete list of filter types, refer to the Export XML Schema Documentation in Agile PLM's *Import/Export User Guide*.

-h host	This is the Agile PLM server machine. The default is `localhost`.
-l port	This is the port on which the Agile PLM server is listening. The default is 80.
-o output-file	This is the output file name. It defaults to: either out.pdx or out.axml, depending on the output format.
-p password	This is the user's password.
-P protocol	This is the URL protocol. Valid values are either HTTP (the default) or HTTPS.
-s site	This is the manufacturing site for which data is extracted. If you do not specify a manufacturing site, data is extracted for all sites.
-t type	This is the type of the required object. Type either the class name or the predefined object type `Default:Items`. For a list of predefined object types, refer to Export XML schema documentation in Agile PLM's *Import/Export User Guide*. This is the Export XML schema, export.xsd. You can find this file in the samples folder described in "AIS Folders" on page 1-2.

Option	Description

?	The pre-defined types listed in export.xsd maps to Agile PLM classes, not subclasses. For example, the predefined ECO object type actually maps to the Change Orders class, not the ECO subclass. If you specify -t ECO when you run `ExportData`, objects of the Change Orders class will be exported, not objects of the ECO subclass.
	
	If you want to use only your Agile PLM system's class names and subclass names for object types instead of the predefined Export object types, you can modify the ExportData.java source code and disable pre-defined object types by replacing the following lines of code:
	
	`try {`
	`// Let's try to use a predefined type.`
	`objType.setPredefined(ObjectTypes.fromString(type));`
	`} catch (Exception ex) {`
	`// Fall back to specifying a type by name (i.e., user-defined type)`
	`objType.setTypeName(type);`
	`}`
	
	with this line:
	
	`objType.setTypeName(type);`

-T timeout	This is the time in minutes to wait for a response. The defaults to 15 minutes.
-u user	This is the Agile PLM username.

The `export.ExportData` client does not have an option to specify an item's revision. When you use the client to export items, the latest released revision is exported. However, you can develop an AIS client that lets you specify a revision to export. For more information, refer to the Export XML Schema documentation in Agile PLM's *Import/Export User Guide*.

These examples show how to run the export.ExportData client.

- `runner export.ExportData -h agilesvr -u aisuser -p e-agile -l 7001 -c "[Title Block.Number] equal to 'P00014'" -t Part -F "Item.TitleBlock" "Item.PageTwo" "Item.Attachments.TableAndFiles" "Item.BOM.Recurse" -o P00014.pdx`
- `runner export.ExportData -h agilesvr -u aisuser -p agile -l 7001 -c "[Title Block.Number] equal to '1000-02'" -f "Default Item Filter" -t Item -s "San Jose" -o D:\data\out.pdx`
- `runner export.ExportData -h agilesvr -u aisuser -p e-agile -l 7001 -c "[Title Block.Number] equal to '1000-02'" -f "Default Item Filter" -t Item -a axml`
- `runner export.ExportData -h agilesvr -u aisuser -p e-agile -l 7001 -c "[General Info.Name] equal to 'ACT'" -f "Default Manufacturer Filter" -t Manufacturer`

Substitute appropriate port numbers. For example, for Weblogic use port 7001, and for OAS use port 7777. For readability, these examples use attribute name, such as [Title Block.Number], instead of IDs. Agile strongly recommends using attribute IDs. If you use attribute names, make sure they are fully qualified to avoid ambiguity.

export.ExportPartlist Usage

Usage: `export.ExportPartlist <options>`

Option	Description
-c criteria	This is the search criteria to locate objects you want to export. The ExportPartlist command exports data only for items with AMLs (approved manufacturer parts and their associated manufacturers). The criteria you specify must be formatted using the Agile SDK query language. For more information, refer to *Agile SDK Developer Guide*.
-e virtual-path	This is the Agile PLM virtual path. For example, if you access Agile Web Client via `http://www.sample.com/Agile/PLMServlet`, the virtual path is "Agile". When you install the Agile PLM system, the default virtual path is "Agile".
-f filter	This is the predefined filter name or ID. If you have administrator privileges, you can define Agile PLM filters using the Agile Java Client.
-F filter-flag	This is the ad hoc filter flag. The valid values for this argument derive from the <filters> element shown in the Export XML Schema documentation in Agile PLM's *Import/Export User Guide*. The filter flags correspond to child elements with names ending in "Filter," such as `ChangeFilter` and `ItemFilter`. The basic pattern for this option is filter-name.attribute.value. filter-name corresponds to the name of the XML element, such as `ItemFilter` (the "`Filter`" suffix may be omitted). attribute corresponds to name of the attribute being defined (for example, "PageTwo"). value corresponds to the value for the attribute. If the attribute is a boolean, the value is optional and defaults to "True." For the Attachments attribute, the value "Tables and Files" causes the attachment table and all the referenced files to be exported. The filter flag should be a class filter such as `PartFilter` (or Part). For a complete list of filter types, see the Export XML Schema Documentation in "AIS Folders" on page 1-2. This is an example of a filter flag: `-F "Part.TitleBlock" "Part.Attachments.TableAndFiles" "Part.BOM.Recurse"`
-h host	This is the Agile PLM server machine. The default is localhost.
-l port	This is the port on which the Agile PLM server is listening. The default is 80.
-o output-file	This is the output file name. The default is out.axml, or out.pdx.
-p password	This is the user's password.
-P protocol	This is the URL protocol. Valid values are either HTTP (the default) or HTTPS.
-r revision	This is the item revision to export.
-s site	This is the manufacturing site for which data is extracted. If you do not specify a manufacturing site, data is extracted for all sites.
-T timeout	This is the time to wait for a response (in minutes). It defaults to15 minutes.
-u user	This is the Agile PLM `username`.

These examples show how to run the `export.ExportPartlist` client.

- `runner export.ExportPartlist -h agilesvr -u aisuser -p agile -l 7778 -c "[Title Block.Number] equal to 'P00408'" -f "Default Item Filter" -o D:\out.axml`
- `runner export.ExportPartlist -h agilesvr -u aisuser -p agile -l 7001 -c "[Title Block.Number] equal to 'P00502'" -r "A" -f "Default Item Filter" -o D:\data\out.axml`
- `runner export.ExportPartlist -h agilesvr -u aisuser -p agile -l 7778 -c "[Title Block.Number] equal to 'P00025'" -f "Default Item Filter" -o D:\data\partlist_rev.axml -r "A"`
- `runner export.ExportPartlist -h agilesvr -u aisuser -p agile -l 7778 -c "[Title Block.Number] equal to 'P00163'" -f "Default Item Filter" "Default Manufacturer Filter" "Default Manufacturer Part Filter" -o D:\data\partlist_bom.axml -r "B"`

importer.ImportData Usage

Usage: `importer.ImportData <options>`

Option	Description
-a mapfile	This is a previously saved mapping definition file.
-e virtual-path	This is the Agile PLM virtual path. For example, if you access Agile Web Client via `http://www.sample.com/Agile/PLMServlet`, the virtual path is "Agile". When you install the Agile PLM system, the default virtual path is "Agile".
-f filetype	This is the type of file that is imported. If this option is omitted, the client determines the filetype based on the MIME type of the import source file.
-h host	This is the Agile PLM server machine. The default is localhost.
-i input-file	This is the source data file.
-l port	This is the port on which the Agile PLM server is listening. The default is 80.
-m map	A textual mapping definition. Arguments should take the form of `<source-path>=<target-path>`.
-n option	This is the an import server option. Arguments take the form of `<group>|<option>=<value>`. Please see the Import XML Schema documentation for more information on available options.
-o output-file	This is the output file name. The default is `log.xml`.
-p password	This is the user's password.
-P protocol	This is the URL protocol. Valid values are either HTTP (the default) or HTTPS.
-t type	This is the type of import operation(s) to run. At least one type must be specified. The format of a type argument is `type[.<child-type>]` (for example., `items.bom, manufacturerParts.attachments`, and `prices.priceLines`). Please see the Import XML Schema documentation for a complete set of available import types.
-T timeout	This is the time to wait for a response (in minutes). It defaults to 15 minutes.
-u user	This is the Agile PLM `username`.

Option	Description
-x transform	This is the a previously saved transformation definition file. For information on how to use the Import wizard to create a transformation definition file, refer to *Agile PLM Import and Export Guide*.

These examples show how to run the importer.ImportData client.

- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\data\bom2.txt -f DelimitedTextFile -t items -n "BusinessRuleOptions|ChangeMode=Authoring" "TextParser|FieldDelimiter=," -o D:\data\result.xml -m Parent="Part.Title Block.Number" Child="Part.Title Block.Description"`
- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\data\bom2.txt -f DelimitedTextFile -t items -n "BusinessRuleOptions|ChangeMode=Authoring" "TextParser|FieldDelimiter=," -o D:\data\result.xml -m Parent="Part.Title Block.Number" Child="Part.Title Block.Description" Type="Part.Title Block.Part Type"`
- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\data\Book1.xls -f ExcelFile -t items -m num="Part.Title Block.Number" desc="Part.Title Block.Description" type="Part.Title Block.Part Type" -o D:\data\result.xml -n "ExcelFileParser|SelectWorksheet=1"`
- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\SourceFiles\Source\Item\item_tab.txt -a D:\SourceFiles\Mapping\Item\item_tab.xml -t items -f DelimitedTextFile -o D:\SourceFiles\Baseline\Item\item_tab_import.xml -n "BusinessRuleOptions|ChangeMode=Authoring" "TextParser|FieldDelimiter=tab"`
- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\SourceFiles\Source\price_lines_import.xls -a D:\SourceFiles\Mapping\price_lines_import.xml -f ExcelFile -t prices.priceLines -o D:\SourceFiles\Baseline\price_lines_import.xml -n "BusinessRuleOptions|ChangeMode=Redline" "BusinessRuleOptions|ChangeNumber=PCO00005"`
- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\SourceFiles\Source\AML_PC.txt -a D:\SourceFiles\Mapping\AML_PC.xml -t items.aml items.bom -f DelimitedTextFile -o D:\SourceFiles\Baseline\AML_PC.xml -n "BusinessRuleOptions|ChangeMode=Redline" "BusinessRuleOptions|ChangeNumber=C00041" "Template|TemplateType=com.agile.imp.template.TemplateParentChildFilter "`
- `runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\SourceFiles\Source\bom_RefDelimiter.txt -a D:\SourceFiles\Mapping\bom_RefDelimiter.xml -t items.bom -f DelimitedTextFile -o D:\SourceFiles\Baseline\new_bom.xml -n "BusinessRuleOptions|ChangeMode=Authoring" "BusinessRuleOptions|ReferenceDesignatorRangeCharacter=-" "BusinessRuleOptions|ReferenceDesignatorDelimiterCharacter=,"`

- runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\SourceFiles\Source\bom_Level.txt -a D:\SourceFiles\Mapping\bom_Level.xml -t items.bom items.aml -f DelimitedTextFile -o D:\SourceFiles\Baseline\bom_Level.xml -n "BusinessRuleOptions|ChangeMode=Redline" "Template|TemplateType=com.agile.imp.template.TemplateLevelFilter" "BusinessRuleOptions|ChangeNumber=C00013"
- runner importer.ImportData -h agilesvr -u aisuser -p agile -l 7778 -i D:\SourceFiles\Source\Item\item_comma_category.txt -a D:\SourceFiles\Mapping\Item\all_mapping_comma.xml -o D:\SourceFiles\Baseline\all_mapping_comma.xml -t items -f DelimitedTextFile -n "BusinessRuleOptions|ChangeMode=Authoring" "TextParser|FieldDelimiter=," "TextParser|LocationOfHeaderRow=3" "TextParser|FileEncoding=ISO8859_1" "ParsingAndValidationOptions|MultilistDelimiterCharacter=;" "ParsingAndValidationOptions|WhitespaceValidationAction=Reject" "ParsingAndValidationOptions|CaseValidationAction=Convert" "ParsingAndValidationOptions|LengthValidationAction=Reject" "TextParser|TextQualifier='"

importer.ImportSupplierResponse Usage

Usage: `importer.ImportSupplierResponse <options>`

Option	Description
-e virtual-path	This is the Agile virtual path. For example, if you access Agile Web Client via http://www.sample.com/Agile/PLMServlet, the virtual path is "Agile". When you install the Agile PLM system, the default virtual path is "Agile".
-h host	This is the Agile server machine. The default is `localhost`.
-i input-file	This is the source data file.
-l port	This is the port on which the Agile server is listening. The default is 80.
-o output-file	This is the output file name. The default is log.xml.
-p password	This is the user's password.
-P protocol	This is the URL protocol. Valid values are either HTTP (the default) or HTTPS.
-r RFQ-number	This is the RFQ into which you are importing the supplier's response.
-s supplier-number	This is the supplier number. It is needed only when a buyer imports an RFQ response for an off-line supplier. If the supplier number is not specified, the import server retrieves the supplier number from the specified input file.
-T timeout	This is the time to wait for a response (in minutes). It defaults to 15 minutes.
-u user	This is the Agile `username`.

These examples show running the `importer.ImportSupplierResponse` client.

- runner importer.ImportSupplierResponse -h agilesvr -u joesupplier -p agile -l 7778 -i D:\SourceFiles\Source\RFQ00256.csv -r RFQ00256

- `runner importer.ImportSupplierResponse -h agilesvr -u joebuyer -p agile -l 7778 -i D:\SourceFiles\Source\RFQ00013.csv -o D:\SourceFiles\Source\Response.xml`

importer.ValidateData Usage

Usage: `importer.ValidateData <options>`

The options are exactly the same as the importer.ImportData Web service. See the command "importer.ImportData Usage" on page 2-9.

Creating a Web Service Client

Using the `ExportData` sample, this section provides the procedures to create a Web service client application for AIS.

Generating the SOAP Request

You can generate an appropriate SOAP request using client-side stubs. You can also generate client-side stubs. The Web-Service-aware code libraries are able to generate client-side stubs on your behalf. This entails using a code generation utility along with the WSDL for the desired Web service.

AIS provided samples that make use of Axis in order to connect with the AIS Web service engine. Axis provides a WSDL2Java utility that you can use for this purpose; other Web-Service-aware libraries will have their own client-side stub generation facility (for example, `.Net` includes a `wsdl.exe` utility). In the case of the samples, the client-side stub generation occurs during the samples' build process. Within the `build.xml` file is the following `Ant` target:

```
<target name="generate-export-stubs" depends="init"
unless="exp-stubs.present">
<echo>Generating export Java client-side stubs from
WSDL...</echo>
<java fork="true"
classname="org.apache.axis.wsdl.WSDL2Java"
failonerror="true">
<classpath refid="build.classpath"/>
<arg line="-o ${built.scratch.dir}/gen"/>
<arg line="-p export"/>
<arg line="${ais.url}/Export?wsdl"/>
</java>
</target>
```

Axis also includes an Ant task definition which you can use instead of the above <java> task. For more information, visit the Axis Ant Tasks site by pasting this URL in your browser http://ws.apache.org/axis/java/ant/ant.htmlhttp://ws.apache.org/axis/java/ant/ant.html.

The above Ant target is responsible for generating the export-related client-side stubs. This invocation retrieves the Export WSDL from `${ais.url}/Export?wsdl`, generates Java code in the export Java package, and places the source code within the `${built.scratch.dir}/gen` directory. For more information on the WSDL2Java utility, refer to Axis documentation on the Axis Website at http://ws.apache.org/axis/.

Once the client-side stubs have been generated, the user can use the generated object definitions in order to more easily generate the appropriate SOAP request. These stubs enable the user to focus on the capabilities of the target Web service operation without

the need to construct a valid SOAP request. In the `ExportData.java` sample, you can see that the run method contains all the code used to generate the SOAP request. However, instead of explicitly constructing a SOAP request, the code is concerned with setting up a Java data structure, which will be provided as the parameter to a stub method invocation. The code is more concerned with functionality than formatting, which makes it easier to read, write, and maintain.

Agile and Non-Agile Web Service Clients

The sample.zip folder contains the source code of an Import/Export Web service client. If you use this client, there is no need to modify the code for the client to connect to Axis v1.4. However if you plan to use an in-house, or a third party client, you must modify the code and work with Axis v1.4.

Submitting the SOAP Request

The next step in the 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 only requires pointing the stubs to the desired server and invoking a method on the stubs. You do not need to worry about opening a connection or manually marshaling your data onto the wire (Marshaling is the act of taking data from the environment you are in and exporting it to another environment), because the generated stubs will handle these details.

The `ExportData.java` sample illustrates the above statement in two places:

- The `getExportStub` method is responsible for pointing the client-side stubs to the desired Web service engine.
- The `stub.exportData` method invocation found within the run method is responsible for actually submitting the request to the Web service engine. The actual submitting of the request and all the minutiae that entails are managed by the stubs themselves; you do not need to worry about the connecting, submitting, or marshaling particulars.

The details on how you point the stubs to the desired Web service engine and submit the request will vary from code library to code library. For more information, refer to the documentation for your Web-Service-aware code library. XXX

Processing the SOAP Response

Similar to submitting a SOAP request, processing a SOAP response is handled with the generated client-side stubs. Without these generated stubs, you must parse the XML-based SOAP response and resolve the many formatting and unmarshaling issues that arise. However, when working with generated stubs, all these details are taken care of so that you will receive the Java objects in a proper form.

The ExportData.java sample illustrates this point clearly. In this sample, you can see that the result of the stub.exportData method is a javax.activation.DataHandler, which is a convenient way of encapsulating a binary data stream. Rather than requiring you to parse an XML document and interpret the returned data, the stubs automatically do this and return the response's attachment as a `DataHandler` object.

The details on how SOAP responses are processed vary from code library to code library. For more information, consult the documentation for your Web-Service-aware code library.

3

Exporting Data

This chapter includes:

- Understanding the Web Service Export Function
- Using the exportData Web Service Operation

Understanding the Web Service Export Function

The following two export Web service operations are delivered as part of AIS:

- `exportData` - A Web service operation that extracts data from an Agile PLM system in one of several data formats.
- `exportPartList` - A Web service operation that takes a multilevel BOM and *flattens* it into a list of the manufacturer parts and their quantities in the BOM, and returns the data in aXML format.

Using the exportData Web Service Operation

The exportData Web service operation is capable of extracting Agile data in one of several structured formats. This operation can be used to provide integration functionality between your Agile PLM system and other, third-party systems.

This section illustrates how to format an XML request in order to use the exportData Web service operation. For more information on the XML schema that describes an exportData request, see the Export XML Schema documentation in"AIS Folders" on page 1-2. To view the information, select **`documentation > schemas > export.htm`**.

The `exportDataRequest` XML element describes the XML format that you must use when submitting an exportData request to AIS. This enables you to specify the following types of data:

- Queries - One or more queries that define what objects is exported.
- Filters - One or more filters that define what data from the selected objects is exported.
- Formats - The format that is used for the exported data.
- Sites - Manufacturing sites for which data should be exported. By default, data for all sites is exported.
- Export - Approval Matrix data is exported.
- Import - Approval Matrix data is imported.

Working with Queries

Using the `exportData` Web service operation, you can specify parameters related to the object query:

- The query itself (Required)
- The type of object being queried (Required)
- The site to apply to all objects matched by the query (Optional)
- The revision to apply to all objects matched by the query (Optional)

You can specify multiple queries at once, returning multiple result sets. More information on query parameters can be found in the Agile API reference documentation. However, the following section provides a brief introduction to the criteria syntax.

Specifying Query Criteria

This section introduces the basics of Agile SDK query syntax. For complete information on how to construct complex search criteria, refer to *Agile SDK Developer Guide - Using Agile APIs*.

The value for the criteria parameter for the `exportData` and `exportPartlist` is a single string consisting of one or more search conditions. Strings that appear within the search criteria should be enclosed in single quotes (').

Working with Sites

Companies that practice distributed manufacturing use several different sites to manufacture their products. The exportData Web service operation supports exporting data to all manufacturing sites, or to a specific site. Manufacturing sites affect how items and changes are exported. In the case of items, BOMs and AMLs can vary per site. For changes, the Affected Items table specifies which manufacturing sites are affected.

By default, the exportData Web service operation extracts information for all sites. If you specify a manufacturing site, only the data associated with that site is exported. All objects that are not associated with that manufacturing site are filtered out of the query results.

The following examples illustrate using exportData to extract all parts and documents in system and exportPartlist to extract only parts with AMLs.

Example: Using exportData to extract all parts and documents in the system

```
runner export.ExportData-h localhost -u admin -p agile -l 7001 -e Agile -c "[Title
Block.Number] is not null " -F "Item.TitleBlock" -t Item -o outpart.pdx
```

Example: Using exportPartlist to extract only parts with AMLs

```
runner export.ExportPartlist -h localhost -u admin -p agile -l 7001 -e Agile -c
"[Title Block.Number] is not null " -F "Item.TitleBlock" -o outpart.pdx
```

The following XML snippets illustrate different ways to specify a manufacturing site:

```
...
<site>
    <site-name>Taipei</site-name>
  </site>
  ...
  <site>
```

```
    <site-id>6</site-id>
  </site>
  ...
  <!--The following is optional since the default is all sites -->
  <site>
    <all/>
  </site>
```

...

Working with Filters

The `exportData` Web service operation enables you to define the information that you want to query from the selected objects. These parameters are captured by specifying one or more filters. Filters are either predefined in the Agile PLM system, or they are defined in an ad hoc fashion by the AIS client.

You can specify multiple filters and their effect is cumulative. The resulting filter is the combination of all specified filters. For example, if one filter includes an item's PageTwo information and a separate filter includes the item's History information, the effective filter includes both PageTwo and History information.

Predefined Filters

Agile provides several predefined filters to refine query results. You can specify a predefined filter in one of the following three different ways: --

- By ID - Specify the numeric ID of a defined filter with the Agile administrative data. This information can be found using the Agile API to inspect the Agile administrative data. Use the ID of a defined filter to reduce the risk of a name change that can adversely affect your code.
- By name - Specify the name of a defined filter found in the Agile administrative data. This is an easy way to reference previously defined filter definitions.
- By object type -Specify different information sets for each type, from the set of available filters, one for each object type.

> **Note:** If you have administrator privileges to the Agile PLM system, you can define new filters. Log into Agile Java Client and choose Admin > System Settings > Agile Contents Service > Filters.

For more information on predefined filters, see the Export XML Schema documentation in AIS Folders. To view the information, select **`documentation > schemas > export.html`**.

Ad Hoc Filters

Ad hoc filters are defined for a particular purpose and are not stored in the Agile PLM system. The Export XML Schema defines several <filters> elements, such as `ItemFilter`, `ChangeFilter`, `ManufacturerFilter`, and `ManufacturerPartFilter`. The general usage for ad hoc filters is to specify the filter type, such as ItemFilter, and then supply boolean values for each table that you want included by the filter. For example, the following ad hoc filter includes the `TitleBlock` and `PageTwo` tables for items:

```
<filters>
<ItemFilter TitleBlock="true" PageTwo="true"/>
  </filters>
```

Most tables require simple boolean values. However, other tables support enumerated values that enable you to include the associated objects. For example, the BOM table supports the following enumerated values for filters: DoNotInclude (the default), TableOnly, SingleLevel, and Recurse.

> **Note:** The filter type that you specify depends on the output format. If you extract data to a PDX file, the filter type should be a superclass filter such as ItemFilter or ChangeFilter. If you extract data to an aXML file, the filter type should be a class filter such as PartFilter or ECOFilter.

An exportData Filter Example

The following code segment illustrates how to combine a predefined filter, "Default Part Filter," with an ad hoc filter that extracts all Item data, including attachments that may result from the query defined in the query example above.

```
...
<filters>
    <!--The following is a predefined filter specified by name-->
    <filter-name>Default Part Filter</filter-name>
    <!--The following is an ad hoc filter -->
    <ItemFilter TitleBlock="true" PageTwo="true"
             PageThree="true" History="true"
             Attachments="TablesAndFiles"
             BOM="Recurse" Changes="true"
             WhereUsed="true"
             AML="TableOnly" Site="true"/>
 </filters>

...
```

Working with Formats

The `exportData` Web service operation can export data in either PDX or aXML format. For a description of these formats, see "Web Services Operations" on page 1-4.

For more information on how to specify the supported formats, see the Export XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select documentation > schemas > export.htm.

An exportData Format Example

The following illustrates how to extract data in PDX format:

```
...
    <format>PDX</format>
  ...
A Sample exportData Web Service Operation
The following is a sample exportDataRequest, which demonstrates a complete
exportData Web service operation request:

<exportDataRequest>
  <queries>
    <query>
      <criteria>[Title Block.Number] == '1000-02'</criteria>
    <objectType>
     <predefined>Item</predefined>
```

```
    </objectType>
   </query>
  </queries>
 <site>
    <site-name>Taipei</site-name>
 </site>
 <filters>
 <!--The following is a predefined filter specified by name-->
 <filter-name>Default Part Filter</filter-name>
 <!--The following is an ad hoc filter -->
 <ItemFilter TitleBlock="true" PageTwo="true"
     PageThree="true" History="true"
     Attachments="TablesAndFiles"
     BOM="Recurse" Changes="true"
     AML="TableOnly" Site="true"/>
  </filters>
 <format>PDX</format>
</exportDataRequest>
```

The above XML sample is not a complete or valid SOAP request. Rather, this XML document represents the contents of a SOAP request body. In general, you do not need to manually generate the above XML document. Instead, the client-side stubs generated by a Web-Service-aware code library take care of creating an appropriately formatted XML document and placing it within a SOAP request, and this sample is an illustration of what the XML request generated by client-side stubs.

Note: To generate the above XML in PDF format, the code that is submitted to the stubs must include at least one such parameter `-c "[Title Block.Number] == '1000-02' " -f "Default Part Filter" -t Item -s Taipei`.

Working with Tables in Export

With the exception of the PDX format, the Export operation supports exporting Job Function and Functional Team attributes as well as the Acknowledge workflow action. Also supports Functional Teams class tables in aXML, Text (csv), and Excel (xls).

Example 3–1 Using exportData to export tables supported by the Functional Teams class

```
runner export.ExportData -h server -l port -e virtualpath -u username -p password
-c "[General Info.Name] equal to 'AIS_KFUNCTEAM1'" -t UserGroup -F
"UserGroup.GeneralInfo" "UserGroup.FunctionalTeam" "UserGroup.JobFunction"
"UserGroup.Discussions" "UserGroup.ActionItems" -o D:\UserGroups.axml -a aXML
```

Using the exportPartlist Web Service Operation

The exportPartlist Web service operation takes a multilevel BOM and "flattens" it into a list of the manufacturer parts in the BOM and their quantities and returns the data in aXML format. That is, it enables you to extract a rolled up set of parts, and the related Quantities Per Top Level Assembly (QPTLA). The value of the QPTLA is computed as the sum over recursive products starting from the top of the BOM tree. This Web service calculates the QPTLA for each unique item-revision pair, and returns the results in the Part Quantities element of the resulting aXML output.

Note: The exportPartlist Web service exports data only for items with AMLs (approved manufacturer parts and their associated manufacturers). Items without AMLs are ignored.

Working with exportPartlist Queries

The exportPartlist Web service is similar to exportData in the way it accepts query definitions. The main difference is that you do not need to specify the object type against which the query is operating. This is because the queries related to a part list must always be queries against items.

Working with exportPartlist Filters

Filters are specified for the exportPartlist Web service operation similar to the exportData Web service operation. The only difference is which filters can be specified. Because exportData only operates over items, manufacturer parts (that is, AML) and manufacturers (AML's related manufacturers), the object-related filters are restricted to those three data types.

An exportPartlist Example

The following is an example of the exportPartlistRequest element. It is a simple adaptation of the previous exportData sample and demonstrates a complete exportPartlist Web service operation request.

Example 3–2 Using the exportPartlistRequest element

```
<exportPartlistRequest>
<queries>
<query>
<criteria>[Title Block.Number] == '1000-02'</criteria>
</query>
</queries>
<site>
<site-name>Taipei</site-name>
</site>
<filters>
<!--The following is a predefined filter specified by name-->
<filter-name>Default Part Filter</filter-name>
<!--The following is an ad hoc filter -->
<ItemFilter TitleBlock="true" PageTwo="true" PageThree="true" History="true"
Attachments="TablesAndFiles" BOM="Recurse" Changes="true" WhereUsed="true"
AML="TableOnly" Site="true"/>
</filters>
</exportDataRequest>
```

The following entries are removed from the previous exportData sample to make this adaptation:

- The query element does not include an objectType element. This is because the exportPartlist Web service operation only queries against item objects.
- The format element is not included in the exportPartlistRequest. This is because the exportPartlist Web service operation only exports data in the aXML format.

The preceding XML example is not a complete or valid SOAP request. Rather, this XML document represents the contents of a SOAP request body. Generally, you do not need to generate the above XML document manually. Instead, the client-side stubs

generated by a Web-Service-aware code library create an appropriately formatted XML document and place it within a SOAP request. The above sample is an illustration of what the XML request generated by client-side stubs would resemble.

Note: To generate the above XML in PDF format, the code that is submitted to the stubs must include at least one such parameter `-c "[Title Block.Number] == '1000-02' " -f "Default Part Filter" -t Item -s Taipei`.

4

Importing Data

This chapter includes:

- Overview
- Understanding the Web Service Import Feature
- Using the importData Web Service Operation
- Importing Data Values

Overview

You can use the `importData` Web service operation of AIS to import data into the Agile PLM databases. The source for the import data can be an Agile database, a third party Product Data Management (PDM) system, or an Enterprise Resource Planning (ERP) system. The Agile server stores information about customer-specific items, such as parts that the company uses to build its products. It also maintains the relationships that assembly parts have with BOM components and that parent items have with approved manufacturers.

For more information on importing data into the Agile PLM system, refer to the *Agile PLMImport/Export User Guide*.

Understanding the Web Service Import Feature

The following Web service import operations are delivered as part of the AIS:

- `importData` - A Web service operation that imports data into the Agile PLM system.
- `importSupplierResponse` - A Web service operation that imports an RFQ response from a supplier.

Note The `ImportSupplierResponse` Web service operation is deprecated as of Agile 9.0 SP1. Instead, invoke the importData Web service operation and construct a valid `importSupplierResponseRequestType` XML data structure. For more information, see "Importing Supplier Responses" on page 4-8. Although the `oldImportSupplierResponse` Web service operation is supported for this release, Oracle recommends migrating your code to the new API.

- `validateData` - A Web service operation that validates source data with compliance rules

Using the importData Web Service Operation

The importData Web service operation exposes all Import Server functionality through a Web service interface that you can access programmatically. This section documents formatting an XML request in order to use the importData Web service operation. For more information on the XML schema that describes an importData request, refer to the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

The `importDataRequest` XML element describes the XML format you must use when submitting an importData request to AIS. It supports the XML data structure types.

Example 4–1 Supported data structure types

```
<importDataRequest xsi:type="importDataRequestType">
...
</importDataRequest>
<importDataRequest xsi:type="importSupplierResponseRequestType">
...
</importDataRequest>
```

Specifying Data Types

The importDataRequest XML element allows you to specify several different types of data, including:

- Data Source - The source of the data to be imported.
- Operations - Which import operations should be performed.
- Mapping - How incoming data should be mapped into the Agile PLM system.
- Transformation - How incoming data should be transformed before importing into the Agile PLM system.
- Options - Other options that affect the behavior of the import server.

Working with Data Sources

A data source is defined by two pieces of information: the URL that references the data to be imported and a data type that defines what kind of data is being imported. The URL specified can be a reference to either an attachment sent along with the SOAP request, or an external resource. If the URL references an attachment, then the SOAP request can follow either the SwA (SOAP With Attachments) or DIME (Direct Internet Message Encapsulation) encoding rules. For more information on these parameters, refer to the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

The following XML example illustrates how to specify a PDX data source that is sent along with the SOAP request.

Example 4–2 Specifying a PDX data source sent with a SOAP request

```
<importDataRequest xsi:type="importDataRequestType">
<dataSource>
<attachmentRef href="cid:E36C913548344EDA1B7FC20CEDCEDEB3"/>
typeIPC2571</type>
</dataSource>
...
```

In the above snippet, the value of the HREF attribute is not intuitive, but it is of the form expected when referencing an attachment sent as part of the SOAP request.

Note: The HREF value is generated by the stub.

Working with Operations

By specifying one or more import operations, you can define what data is imported into the Agile PLM system. The following table lists valid import operations.

Operation	Child Attributes
currencyConversion	n/a
customers	n/a
declarations	items, manufacturerParts, partFamilies, itemSubstances, mfrpartSubstances, partFamilySubstances, specifications,attachments
items	aml, bom, sites, attachments, composition, substances, suppliers, specifications, relationships
manufacturerParts	attachments, composition, substances, suppliers, specifications, relationships
manufacturers	attachments, relationships
partgroups	parts, suppliers, specifications, relationships, attachments
prices	priceLines, attachments
productServiceRequests	affectedItems, relatedPSR, relationships, attachments
projectItems	aml, bom, attachments
qualityChangeRequests	affectedItems, relationships, attachments
quoteHistories	quoteHistoryLines
specifications	substances, attachments
substances	materialCompositions, attachments
suppliers	supplier, manufacturerOfferings, commodityOfferings
users	usergroup
usergroups	user

Depending on what you specify, the import server performs the desired import operations and ignores data that is not relevant to the selected import operation. For more information on import operations, see the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **`documentation > schemas > import.htm`**.

The following code snippet illustrates how to import manufacturers, manufacturer parts, and items. For items, the BOM and AML tables are also imported.

Example 4–3 Importing manufacturers, manufacturer parts, and items

```
<operations>
<manufacturers attachments="false"/>
<manufacturerParts attachments="false"/>
<items aml="true" bom="true" sites="false" attachments="false"/>
```

```
</operations>

...
```

Working with Mappings

The specified mappings determine how the incoming data is mapped into the Agile PLM system. You can specify mappings either by referencing a previously defined mapping definition file, or by specifying the mappings via the submitted XML data structure. Referencing a previously defined mapping definition file occurs in much the same way as a data source is referenced (that is, via an HREF attribute on the appropriate element). Specifying a mapping via the XML data structure requires specifying the source and target attributes in the appropriate format.

For more information on these parameters, see the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

The following snippet illustrates how to map a field from the incoming PDX package onto the TitleBlock of an item.

Example 4–4 Mapping a field from incoming PDX package to Item's TitleBlock

```
...
<mapping>
<entry>
<source>/ProductDataeXchangePackage/Items/Item@itemIdentifier</source>
<target>Part.Title Block.Number</target>
</entry>
</mapping>
...
The following snippet illustrates how you can reference a previously defined
mapping definition file.
...
<mapping>
<attachmentRef href="cid:E36C913548344EDA1B7FC20CEDCEBEEF"/>
</mapping>
...
```

In the above snippet, the HREF attribute which is generated by the stub is not very intuitive, but it is of the form expected when referencing an attachment sent as part of the SOAP request.

> **Note:** Agile PLM allows you to define an unlimited number of new flex fields for each type of business object. Both the Agile Import wizard and AIS now support user-defined flex fields. Therefore, you can import data to user-defined flex fields.

Working with Transforms

Transforms are used to specify the way data is transformed as it is imported into the Agile PLM system. To specify Transforms, use the previously defined transformation definition files as shown in the following example.

Example 4–5 Specifying a Transform

```
...
<transform href="cid:E36C913548344EDA1B7FC20CEDCE0123"/>
...
```

In the above snippet, the HREF attribute which is generated by the stub is not very intuitive, but it is of the form expected when referencing an attachment sent as part of the SOAP request. For more information on this parameter, see the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

Working with Options

The import server provides several options that you can set in order to alter the behavior of the import server. These options are grouped together into related option groups, which makes it easier to distinguish the purpose of the related options. For more information on these parameters, see the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

The following snippet illustrates how to set several Business Rule, Parsing and Validation options:

Example 4–6 Setting usiness Rule, Parsing and Validation options

```
...
<options>
    <BusinessRuleOptions>
    <ChangeMode value="Authoring"/>
    <MultiRowUpdateMode value="AddUpdateOnly"/>
   </BusinessRuleOptions>
   <ParsingAndValidationOptions>
   <CaseValidationAction value="Convert"/>
   </ParsingAndValidationOptions>
</options>
```

ChangeType and ChangeAutoNumber Options

The import server supports setting the following `ChangeType` and `ChangeAutoNumber` options when importing items in the Redline mode. This in addition to setting the same for a ChangeAutoNumber. You have the option to specify a non-existing change in AIS, and the Import server generates the change for the affected ChangeType, `ChangeNumber` or ChangeAutoNumber. When a change order is initiated, the server records a message that includes the type and number of the change in the AIS log file. For more information on these parameters, see the Import XML Schema documentation in "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

- `ChangeType` - This option supports specifying the subclass name or ID of the change order for the ECO,SCO, or MCO. If the change type is invalid, the Import server will reject the entire Import operation and will record a fatal message in the AIS log file.
- ChangeAutoNumber - This option supports generating change numbers with the specified Autonumber. If the specified ChangeAutoNumber is invalid, the Import server will reject the entire import operation and will record a fatal message in the AIS log file.

Note: Do not set the ChangeNumber option if you have already invoked the ChangeAutoNumber option.

The following example illustrates how to set the ChangeMode, ChangeType, and ChangeAutoNumber options in the aXML file.

Example 4–7 ChangeMode, ChangeType, and ChangeAutoNumber settings in aXML

```
...
<options>
   <BusinessRuleOptions>
   <ChangeMode value="Redline"/>
   <ChangeType value="ECO"/>
   <ChangeAutoNumber value="ECO AutoNumber"/>
   <MultiRowUpdateMode value="AddUpdateOnly"/>
   </BusinessRuleOptions>
   <ParsingAndValidationOptions>
   <CaseValidationAction value="Convert"/>
   </ParsingAndValidationOptions>
</options>
...
```

The parameter in the client-side code that is submitted to generate this XML must contain:

```
-n "BusinessRuleOptions|ChangeMode=Redline" "BusinessRuleOptions|ChangeType=ECO"
"BusinessRuleOptions|ChangeAutoNumber=ECO AutoNumber"
"BusinessRuleOptions|MultiRowUpdateMode=AddUpdateOnly"
"ParsingAndValidationOptions|CaseValidationAction=Convert"
```

Options to Import Non-Existing Objects

You have the option to accept or reject importing non-existing objects during an import operation. This behavior is supported by the BehaviorUponNonExistingObjects option. This option has two values, Accept and Reject. Accept creates the non-existing objects during import and Reject skips creating these objects.

You can find detailed information about BehaviorUponNonExistingObjects in the documentation folder in the AIS_samples.zip file. To access this file, see "AIS Folders" on page 1-2. To view the information, select **documentation > schemas > import.htm**.

Invoking the ImportDataRequest Operation

The following is a complete example of invoking importDataRequest. It shows what a fully configured importData operation request will resemble.

Example 4–8 An ImportData Example

```
<importDataRequest xsi:type="importDataRequestType">
<dataSource>

<attachmentRef href="cid:E36C913548344EDA1B7FC20CEDCEDEB3"/>
<type>IPC2571</type>
</dataSource>

<operations>
```

```
<manufacturers attachments="false"/>
    <manufacturerParts attachments="false"/>
    <items aml="true" bom="true" sites="false" attachments="false"/>
  </operations>
  <mapping>
    <attachmentRef href="cid:E36C913548344EDA1B7FC20CEDCEBEEF"/>
  </mapping>
  <options>
    <BusinessRuleOptions>
      <ChangeMode value="Authoring"/>
      <MultiRowUpdateMode value="AddUpdateOnly"/>
    </BusinessRuleOptions>
    <ParsingAndValidationOptions>
      <CaseValidationAction value="Convert"/>
    </ParsingAndValidationOptions>
  </options>
</importDataRequest>
```

The above XML document is not a complete or valid SOAP request. Rather, this XML document represents the contents of a SOAP request body. Generally, you do not need to generate the above XML document by manually. Instead, the client-side stubs generated by a Web-Service-aware code library will usually create an appropriately formatted XML document and places it within a SOAP request. The above sample is simply an illustration of what the XML request generated by client-side stubs might look like.

Following is the client-side code that is submitted to generate this XML. For additional examples, see "importer.ImportSupplierResponse Usage" on page 2-11.

```
runner importer.ImportData -p http -h localhost -e web -u admin -p agile -l 8888
-f ExcelFile -i D: \source.xls -a D: \mapFile.xml -t items.aml items.bom
manufacturers manufacturerParts -n "BusinessRuleOptions|ChangeMode=Authoring"
"BusinessRuleOptions|MultiRowUpdateMode=AddUpdateOnly"
"ParsingAndValidationOptions|CaseValidationAction=Convert"
```

Using the validData Web Service Operation

This operation exposes the validation service through a Web service interface that you can invoke programmatically. This operation validates the source data for compliance with server rules that govern length, size, and other formats before importing them into the Agile PLM system. For information on programmatic support, refer to the Agile PLM SDK Developer Guide. For information on the UI implementation, refer to *Agile PLM Import/Export User Guide*.

The `validateData` operation uses the same `importDataRequestType` used by the `importData` Web service operation. For procedures to specify the importDataRequestType, see "Using the importData Web Service Operation" on page 4-2. For more information on the XML schema that describes the importData request, refer to the Import XML Schema.

> **Note:** The `validateDataReqeustType` is a subclass of the importDataRequestType, but it does not define any additional methods. In that way, the two are exactly the same. In future releases, the validateData operation will use validateDataReqeustType instead of importDataRequestType.

Importing Supplier Responses

To import supplier responses using the `importDataRequest` Web service operation, specify "`importSupplierResponseRequestType`" for the `xsi:type element`. The importSupplierResponseRequestType is much simpler than `importDataRequestType` because it is much more constrained. You don't need to specify import operations, mapping files, transformation files, or options to import an RFQ response. The importSupplierResponseRequestType XML element allows you to specify three types of data:

- Data Source - This is the source of the data to be imported.
- RFQ Number - This is the alphanumeric identifier of the RFQ that is associated with the response.
- Supplier Number - This is the supplier number is needed only when a buyer imports an RFQ response for an off-line supplier. If the supplier number is not specified, the import server retrieves the supplier number from the specified input file.

For more information on importSupplierResponseRequestType parameters, see the Import XML Schema documentation in "AIS Folders" on page 1-2.

The following is a complete sample for `importSupplierResponseRequest`. It demonstrates how a fully configured `importSupplierResponse` operation request can appear.

Example 4–9 A fully configured importSupplierResponse request

```
<importDataRequest xsi:type="importSupplierResponseRequestType">
   <dataSource>
      <attachmentRef href="cid:E36C913548344EDA1B7FC20CEDCEDEB3"/>
   </dataSource>
     <rfqNumber value="RFQ00123"/>
</importDataRequest>
```

The above XML document is not a complete or valid SOAP request. It is a depiction of the XML request that is generated by the client-side stubs.

Working with Tables in Import

The Import operation supports Importing Job Function and Functional Team attributes and Functional Teams class tables in aXML, Text (csv), and Excel (xls) formats. The only exception is the PDX format which the Import operation does not support.

Example 4–10 Importing a Job Functional Team class Table - Job Functions

Source file: **imp_jobfunctions.txt** with the following contents:

```
Name,Type,Status,JobName,Users/User Groups,
AIS_KFUNCTEAM1,Functional Team,Active,Product Manager,kuser1,
AIS_KFUNCTEAM2,Functional Team,Active,Lead Developer,kuser2,
AIS_KFUNCTEAM3,Functional Team,Active,Developer,kuser3,
```

Solution

```
runner importer.ImportData -h server -l port -e virtualpath -u username -p
password -i D:\imp_jobfunctions.txt -f DelimitedTextFile -t usergroups.jobfunction
-n "TextParser|FieldDelimiter=," -o D:\result_jf.xml -a D:\mapping_jf.xml
```

Example 4–11 Importing a Functional Team class Table - Discussion

```
Source file: imp_discussion.txt with the following contents:
Name,Type,Status,Subject,Number
AIS_KFUNCTEAM1,Functional Team,Active,DISC1,D00001
AIS_KFUNCTEAM2,Functional Team,Active,DISC1,D00001
AIS_KFUNCTEAM3,Functional Team,Active,DISC1,D00001
```

Solution

```
runner importer.ImportData -h server -l port -e virtualpath -u username -p
password -i D:\imp_discussions.txt -f DelimitedTextFile -t usergroups.discussion
-n "TextParser|FieldDelimiter=," -o D:\result_disc.xml -a D:\mapping_disc.xm
```

Example 4–12 Importing a Functional Team class Table - Action Items

```
Source file: imp_actionitems.txt with the following contents:
Name,Type,Status,Subject,AssignedTo
AIS_KFUNCTEAM1,Functional Team,Active,ActionItem1,admin
AIS_KFUNCTEAM2,Functional Team,Active,ActionItem2,admin
AIS_KFUNCTEAM3,Functional Team,Active,ActionItem3,admin
```

Solution

```
runner importer.ImportData -h server -l port -e virtualpath -u username -p
password -i D:\imp_actionitems.txt -f DelimitedTextFile -t usergroups.actionitem
-n "TextParser|FieldDelimiter=," -o D:\result_action.xml -a D:\mapping_action.xml
```

Importing Data Values

The Import Web service supports a variety of date formats based on several different criteria, including user preferences and locale.

Note: The upper limit for dates is today's date + 150 years. Date values later than that are invalid and cannot be imported.

Setting the Preferred Date Format and Time Zone

Each Agile user can select a preferred date format.

To change date format preferences for your Agile account:

1. In Agile Web Client, select Settings > User Profile > Preferences > Edit.
2. Select the desired date format in the Preferred Date Format field.
3. Select a GMT time zone in the Time Zone field.
4. Click Save.

Supported Date Formats

The Import Web service supports all combinations of date and time formats available in the java.text.DateFormat class as well as additional formats. DateFormat provides many date and time formatting styles based on locale. The following table shows date formats available for the U.S. locale, evaluated in order:

Date Format	Example
MMM-dd-yyyy HH:mm:ss	Jul-10-2001 14:08:35

Date Format	Example
MMM-dd-yyyy HH:mm	Jul-10-2001 14:08
MMM-dd-yyyy hh:mm:ss a	Jul-10-2001 02:08:35 PM
MMM-dd-yyyy hh:mm a	Jul-10-2001 02:08 PM
MMM-dd-yyyy	Jul-10-2001
dd-MMM-yyyy HH:mm:ss	10-Jul-2001 14:08:35
dd-MMM-yyyy HH:mm	10-Jul-2001 14:08
dd-MMM-yyyy hh:mm:ss a	10-Jul-2001 02:08:35 PM
dd-MMM-yyyy hh:mm a	10-Jul-2001 02:08 PM
dd-MMM-yyyy	10-Jul-2001
EEEE, MMMM d, yyyy	Thursday, June 25, 1998
EEEE, MMMM d, yyyy h:mm a	Thursday, June 25, 1998 1:32 PM
EEEE, MMMM d, yyyy h:mm:ss a	Thursday, June 25, 1998 1:32:19 PM
EEEE, MMMM d, yyyy h:mm:ss a z	Thursday, June 25, 1998 1:32:19 PM GMT-05:00
MMMM d, yyyy	June 25, 1998
MMMM d, yyyy h:mm a	June 25, 1998 1:32 PM
MMMM d, yyyy h:mm:ss a	June 25, 1998 1:32:19 PM
MMMM d, yyyy h:mm:ss a z	June 25, 1998 1:32:19 PM GMT-05:00
MMM d, yyyy	Jun 25, 1998
MMM d, yyyy h:mm a	Jun 25, 1998 1:32 PM
MMM d, yyyy h:mm:ss a	Jun 25, 1998 1:32:19 PM
MMM d, yyyy h:mm:ss a z	Jun 25, 1998 1:32:19 PM GMT-05:00
M/d/yy	6/25/98
M/d/yy h:mm a	6/25/98 1:32 PM
M/d/yy h:mm:ss a	6/25/98 1:32:19 PM
M/d/yy h:mm:ss a z	6/25/98 1:32:19 PM GMT-05:00

Each date format is specified using a time pattern string where

y = year M = month in year d = day in month h = hour in AM/PM (1~12) m = minute in hour s = second in minute E = day in week a = AM/PM marker z = time zone ' = escape for text '' = single quote

The count of each letter such as "M" in the time pattern determines the format. For example, three "M" characters indicate that the month is represented as text instead of a number; less than three "M" characters means that the month is represented by a number.

For more information about Java date formats and time pattern syntax, see Oracle documentation for the `SimpleDateFormat` and `DateFormat` classes at: `http://docs.oracle.com/javase/1.5/docs/`

Specifying Time Zones

Date values can specify a GMT time zone. If a date value omits the time zone, the user's time zone preference is used. Time zones must be entered in the following format:

GMT Sign hh:mm

where:

GMT = Greenwich Mean Time

Sign = + or -

h = hour in AM/PM (1 to12)

m = minute in hour

For example, "GMT-05:00" and "GMT+02:00" are valid time zones.

Note: Do not use three-character codes (such as PST or PDT) to specify time zones. Three-character time zone codes are unreliable because some are used for multiple time zones. Consequently, the Agile server might resolve a three-character time zone code to an incorrect time zone.

aXML and PDX Package Date Formats

For aXML and PDX packages, the Import Web service operation supports a subset of the ISO String date format: yyyy/MM/ddTHH:mm:ssZ

Note: The T and Z characters are optional.

Importing XLSX File Formats

The AIS Import function supports importing XLSX or Microsoft 2007/2010 Excel file types by using a command similar to the one shown below:

Example 4–13 A sample command to import XLSX file types using AIS

```
runner importer.ImportData -h server -l port -e
virtualpath -u username -p password -i
D:\Item_BOM_LT_2010.xlsx -f ExcelXLSXFile -t items
-n "BusinessRuleOptions|ChangeMode=Authoring"
"ExcelFileParser|SelectWorksheet=1"
```