Agile Integration Services™ Developer Guide

Release 9.2.1

Part No. E11129-01

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

Copyrights and Trademarks

September 12, 2007

REVISIONS

Revision	Date	Pages Affected	Description
A	5/10/06	All	Initial release of the manual for Agile 9.2.1.
B	09/12/06	Chapter 5	Removed migration chapter.

CONTENTS

PREFACE

The Agile documentation set includes Adobe® Acrobat™ PDF files. The Oracle Technology Network (OTN) Web site contains the latest versions of the Oracle|Agile PLM PDF files. You can view or download these manuals from the Web site, or you can ask your Agile administrator if there is an Oracle|Agile Documentation folder available on your network from which you can access the Oracle|Agile documentation (PDF) files.

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

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

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

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

Readme

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

Agile Training Aids

Go to the Agile Training Web page for more information on Agile Training offerings.

CHAPTER 1
Introduction

This chapter contains information about AIS and web services. It has the following sections:

- *Understanding AIS*
- *Understanding Web Services*
- *AIS Web Service Operations*
- *Web Service Extensions*
- *Security*

Understanding AIS

Agile Integration Services (AIS) are a collection of predefined Web services in the Agile Integration Framework to enable communication between the Agile server and disparate systems, 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. Using AIS to exchange content with other systems simplifies the process for aggregating raw product content, and makes critical product content available in real time to other core systems.

AIS Web services provide import and export capabilities, which can be used to:

- Make product content available to Enterprise Application Integration (EAI) systems.
- 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 custom applications.
- Import product content data from ERP and other supply chain applications.

Key Features

AIS includes the following key features:

- **Programmatic access to data** — AIS provides programmatic access to data stored in Agile PLM systems.
- **Accessibility** — AIS provides accessibility of Agile product content outside of corporate firewall using standard HTTP(S) technology.
- **WSX Framework** — Agile Web Service Extensions (WSX) are based on the AXIS package. Agile certifies client applications using AXIS package client.

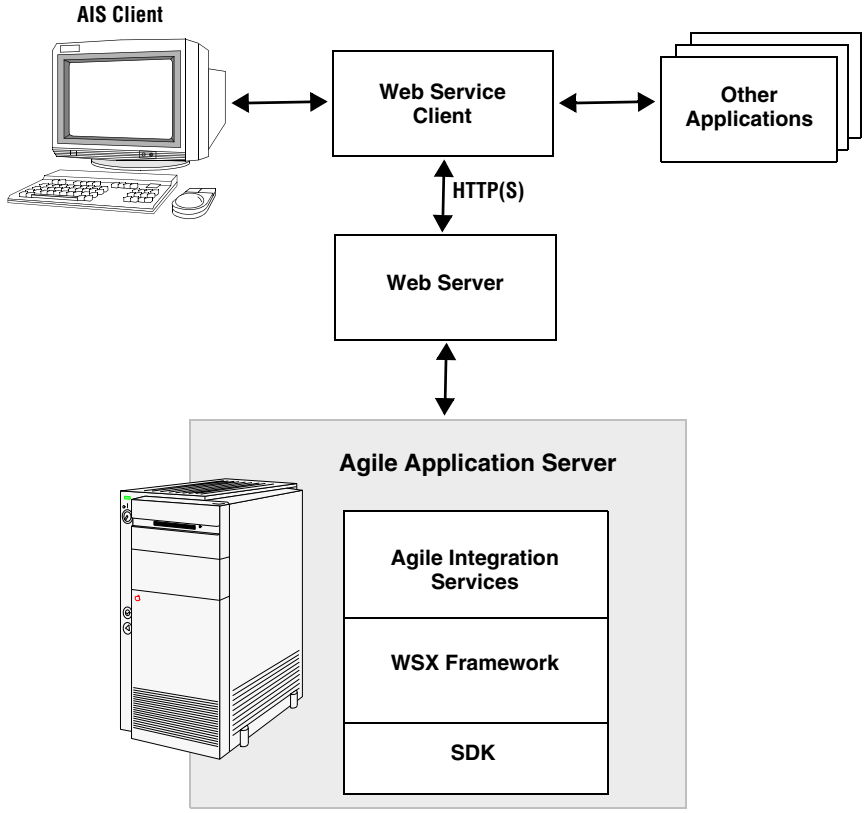
  Note Export and Import attachment types are incompatible with .Net attachment types.

- **Multiple output format support** — AIS supports aXML and PDX 1.0.
- **Security** — AIS communicates with XML-compliant applications using Internet-standard communication and security protocols (HTTP and SSL), so the interface is both firewall-friendly and secure.
- **Multilevel BOM support** — Lets you convert a multilevel BOM into individual parts data in XML

AIS Architecture

To connect to AIS, you use standard web service invocation methodologies.

Figure 1-1: AIS architecture



AIS Folders

When you install the Agile Application Server, Agile Integration Services are installed on the server if you purchased a separate Agile Integration Services license. In addition to its server component, AIS also includes the following folders:

- **bin** — Contains binaries and files related to migrating from a previous AIS release.
- **lib** — Contains common JAR files used by samples and binaries.

AIS documentation and source files for sample clients are available from the Agile Documentation Web site, which can be accessed from the Help menu in the Agile Java Client and the Agile Web Client.

Understanding Web Services

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

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

The key benefits of web services are:

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

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

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

Web Services Architecture

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

- Web service roles:
  - **Service provider**—provides the service by implementing it and making it available on the Internet.
  - **Service requestor**—user of the service who accesses the service by opening a network connection and sending an XML request.
  - **Service registry**—a centralized directory of services where developers can publish new services of find existing ones.
- Web services protocol stack:
  - **Service transport layer**—uses HTTP to transport messages between applications. Other transports will be supported in future AIS releases.
  - **XML messaging layer**—encodes messages in XML format by using SOAP, a platform-independent XML protocol used for exchanging information between computers. It defines an envelope specification for encapsulated data being transferred, the data encoding rules, and RPC conventions.
  - **Service description layer**—describes the public interface to a specific web service by using the Web Service Description Language (WSDL) protocol. WSDL defines an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages, which contain either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a network protocol and message format. WSDL allows description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. A WSDL document defines services as collections of network endpoints (called ports). A *port* is defined by associating a network address with a 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 AIS web services do not currently support UDDI or other service discovery layers.

Further Information about Web Services

Here are some helpful Web sites to explore:

- **WebServices.Org** — http://www.webservices.org/
- **Web Services Architect** — http://www.webservicesarchitect.com/
- **Web Services Journal** — http://www.sys-con.com/webservices/
- **webservices.xml.com** — http://webservices.xml.com/
- **O'Reilly Web Services** — http://webservices.oreilly.com/
- **Apache Axis** — http://ws.apache.org/axis/
- **Java Web Services Developer Pack 1.1** — http://java.sun.com/webservices/webservicespack/html
- **Sun ONE Web Services Platform Developer Edition** — http://sunonedev.sun.com/building/development/developer_platform_overview.html
- **Microsoft .Net Framework** — http://msdn.microsoft.com/netframework/
- **SOAP::Lite for Perl** — http://www.soaplite.com
- **Soap Tutorial** — http://www.w3schools.com/soap/default.asp

AIS Web Service Operations

AIS enables you to export data programmatically into structured XML documents and import data into the Agile PLM system by providing 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 the DTD, see following Web page:
    http://webstds.ipc.org/2571/2571.htm
  - **Agile XML (also known as aXML)** — Agile XML format is an XML representation of Agile's business schema. aXML contains all product content managed in Agile including items, change details, manufacturer information, cost, drawings and other files. As a representation of schema elements across all Agile products, aXML will evolve with Agile's business schema over time.

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

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

    For changes to the aXML format changes from Agile 9.2 to Agile 9.2.1, please see the *Agile PLM 9.2.1 Readme* document.
- **exportPartList** — A web service operation that takes a multilevel BOM and "flattens" it into a list of the items and associated 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 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—the 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.
- **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 4-5.

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 Service Extensions

You can use the Agile SDK to develop web service extensions (WSX) that leverage the functionality of AIS while at the same time 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, see the *Agile SDK Developer Guide*.

Security

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.

Username 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, see the following Web page:
http://java.sun.com/j2se/1.3/docs/guide/security/index.html

CHAPTER 2
Using AIS Web Services

This chapter contains information about how to use AIS web services. It has the following sections:

- *Getting Started*
- *Checking Your AIS System*
- *Using the AIS Java Samples*
- *Creating a Web Service Client*

Getting Started

This section describes the tools you can use to develop client applications, the languages that can generate and process XML and process HTTP request/response messages, and the general steps to access the prebuilt AIS web services. The AIS samples will be used to illustrate the indicated steps.

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/response messages.

AIS 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 AIS 9.1 uses AXIS 1.1. This affects the classnames that are generated for the Java samples included with AIS. For detailed information about AXIS, its features, and how to use it, see the Apache Web site: http://ws.apache.org/axis/

Client Programming Languages

Agile tests and certifies Java for use in developing AIS clients.

Note WSDL is only supported with J2EE.

Here are some Web sites where you can find more information:

- **Apache Axis** — Open source SOAP implementation for Java. See the following Website: http://ws.apache.org/axis/
- **Java Web Services Developer Pack (JWSDP)** — Sun's Java implementation of the SOAP protocol. See the following Website: http://java.sun.com/webservices/webservicespack.html

- **Microsoft .Net** — An XML Web services platform for Microsoft Windows that can be used to create web service clients. See the following Website:
  http://msdn.microsoft.com/net
- **SOAP::Lite for Perl** — A Perl implementation of the SOAP protocol. See the following Website:
  http://www.soaplite.com/

  Note For a comprehensive list of other SOAP implementations, see the following Website:
  http://www.soapware.org/

Accessing AIS Web Services

In general, to access AIS web services, 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 AIS via HTTP or HTTPS** — Once an appropriate set of client-side stubs has been generated, a client application can use these stubs to submit a request.
3. **Process the SOAP response** — The client-side stubs usually are responsible for processing the SOAP response and converting the response into an appropriate set of return objects.

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

Checking Your AIS System

Before trying to compile and run the sample AIS web service clients, make sure web services are working properly on the Agile PLM server. For more information, see the Agile PLM installation guide.

Using the AIS Java Samples

AIS provides several sample Java web service clients that you can download. The AIS samples use Axis to connect with the AIS web service engine to generate client-side stubs. The sample clients can be used to export and import data. They provide command-line interfaces to the ExportData, ExportPartlist, and ImportData web service operations.

Note Client stubs for import operations use the following XML namespace :
http://www.agile.com/2004/01/ais/import

It's important to remember that the AIS Java samples do not expose all AIS functionality. They are merely sample clients. For example, the samples let you run only one query at a time, but AIS lets you run multiple queries and then aggregate the results. You may choose to develop AIS clients with additional functionality. The samples provide source code you can use to learn how to develop your own AIS clients. For complete information about the functionality supported by the Export and Import web services, see the Export and Import XML Schema documentation.

Before you can build and run the AIS samples, download the following required tools:

- Java 2 SDK SE Version 1.3.1. You can download this software at the following Website:
  http://java.sun.com/j2se/1.3/download.html
- The Apache Project's Ant build tool, version 1.6, available at: http://ant.apache.org/

Installing the Java SDK

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

To install the Java SDK on Windows:

1. Double-click the distribution and follow the installation procedures.
2. Set the system variable JAVA_HOME to point to the home directory of your Java SDK installation (for example, D:\j2sdk131).

To install the Java SDK on Solaris:

1. Execute the distribution (for example, $ ./ j2sdk-1_3_1-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>/j2sdk131).
3. Execute your .profile or .cshrc (depending on your shell) file to reinitialize your environment settings.

Installing Ant

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

To install the Ant on Windows:

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

   The Ant distribution for Windows is a zip file (for example, apache-ant-1.6.0-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 installation 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 reinitialize your environment settings.
3. From a command prompt, verify that Ant can be invoked by entering the following command:

   ```
   $ANT_HOME/bin/ant -version
   ```

   The following output should be displayed:

   ```
   Apache Ant version 1.6.0 compiled on date
   ```

Building The Java Samples

Building the Java samples is very straightforward. You need the Ant build tool, which is available for download at http://ant.apache.org/. For information on installing Ant, see “Installing Ant” on page 2-3.

Run Ant within the samples directory, specifying the URL to your AIS installation. After you build the samples, the AIS samples directory contains a runner file, which can be used to run the samples. It contains all the necessary CLASSPATH initialization for the samples.

Note If you generated client stubs for the AIS samples from the WSDL, they will work fine 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. Open a command prompt window and go to the AIS samples folder, which contains a file named build.xml.
2. Enter the following command:

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

where

- *hostname* is the name of the Agile server
- *port* is the application server port. If you are using Oracle Application Server to host the Agile PLM system, enter 7777. If you are using BEA WebLogic Server, enter 7001.
- *virtualPath* is the virtual path for your Agile server. The virtual path is specified when the Agile PLM system is installed. The default virtual path is "Agile". For more information, see the *Agile Installation and Upgrade Guide*.
- *target* 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* is the Agile PLM user ID
- *password* is the Agile PLM password

Note Agile PLM 9.2.1 requires username and password for the building of Java samples. The makefile will error if all three parameters are not set.

To build AIS Java samples on Solaris:

1. Go to the AIS samples directory, which contains a file named build.xml.
2. Enter the following command:

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

If you are connecting to a secure URL that uses SSL, enter the following command instead:

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

For descriptions of *hostname*, *port, virtualPath*, *username*, *password*, and *target*, see the previous section.

After you build a sample, it is ready to run. See the comments (or JavaDoc) in each sample for more information.

Building the Java Samples Using SSL

To build the Java AIS samples on a server that has enabled Secure Sockets Layer (SSL), follow these steps:

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>
```

Note The Readme.txt file installed with the AIS samples includes more information about how to obtain a certificate, how to intall it in your Java environment, and how to build and run the AIS samples on an SSL-enabled system.

Running the Java Samples

Once you perform the build, the AIS samples directory will contain the following file that contains all the necessary CLASSPATH initialization and can be used to simplify the process of actually invoking a sample application:

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

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

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

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

export.ExportData Usage

Usage: `export.ExportData <options>`

Table 2-1: export.ExportData options

Option	Description
-a axml	Selects aXML output format instead of the default PDX output format.
-c *criteria*	The search criteria for locating objects to export. The criteria must be formatted using the Agile SDK query language. For more information, see the *Agile SDK Developer Guide*.
-e *virtual-path*	The Agile PLM virtual path. For example, if you access the 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*	Predefined filter name or ID. If you have administrator privileges, you can define Agile PLM filters using the Agile Java Client.

-F *filter-flag*	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*. *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. Here is an example of a filter flag:
	`-F "Item.TitleBlock" "Item.Attachments.TableAndFiles" "Item.BOM.Recurse"`
	
	If you are extracting data to a PDX file, 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`
	
	If you are extracting data to an aXML file, the filter flag should 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, see the Export XML Schema Documentation.

-h *host*	The Agile PLM server machine. Default: localhost
-l *port*	Port on which the Agile PLM server is listening. Default: 80
-o *output-file*	The output file name. Default: either out.pdx or out.axml, depending on the output format.
-p *password*	User's password.
-P *protocol*	URL protocol. Either http (the default) or https.

Table 2-1: export.ExportData options (continued)

Option	Description
-s *site*	The manufacturing site for which data is extracted. If you do not specify a manufacturing site, data is extracted for all sites.

-t *type*	Type of object being queried. Enter either the class name or predefined object type. Default:Items. For a list of predefined object types, see the Export XML schema documentation.
	The Export XML schema, `export.xsd`, can be obtained by installing the documentation and samples:
	`C:\Agile\Agile921Docs\integration\ais\documentation\schemas\export.xsd`
	The predefined types listed in export.xsd, map 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.
	Note: 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 predefined 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*	Amount of time to wait for a response (in minutes). Defaults to 15 minutes.
-u *user*	Agile PLM username.

Note The export.ExportData client does not have an option for specifying 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, see the Export XML Schema documentation.

Here are some examples showing how to run the export.ExportData client.

```
runner export.ExportData -h agilesvr -u aisuser -p 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 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 agile -l 7001 -c "[General
Info.Name] equal to 'ACT'" -f "Default Manufacturer Filter" -t Manufacturer
```

Note Substitute the appropriate port number, for example for BEA Weblogic, use port number `7001` and for Oracle, use port number `7777`.

Note For readability, the above examples use attribute names, such as [Title Block.Number], instead of IDs. However, Agile strongly recommends using attribute IDs instead. If you choose to use attribute names to run AIS samples, make sure you fully qualify the attribute names to avoid ambiguity.

export.ExportPartlist Usage

Usage: `export.ExportPartlist <options>`

Table 2-2: export.ExportPartlist options

Option	Description
-c *criteria*	The search criteria for locating objects to export. ExportPartlist 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, see the *Agile SDK Developer Guide*.
-e *virtual-path*	The Agile PLM virtual path. For example, if you access the 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*	Predefined filter name or ID. If you have administrator privileges, you can define Agile PLM filters using the Agile Java Client.
-F *filter-flag*	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*. *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. Here is an example of a filter flag: `-F "Part.TitleBlock" "Part.Attachments.TableAndFiles" "Part.BOM.Recurse"`
-h *host*	The Agile PLMserver machine. Default: localhost
-l *port*	Port on which the Agile PLM server is listening. Default: 80
-o *output-file*	The output file name. Default: out.axml
-p *password*	User's password.
-P *protocol*	URL protocol. Either http (the default) or https.
-r *revision*	The item revision to export.
-s *site*	The manufacturing site for which data is extracted. If you do not specify a manufacturing site, data is extracted for all sites.
-T *timeout*	Amount of time to wait for a response (in minutes). Defaults to 15 minutes.
-u *user*	Agile PLM username.

Here are some examples showing 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>`

Table 2-3: importer.ImportData options

Option	Description
-a *mapfile*	A previously saved mapping definition file.
-e *virtual-path*	The Agile PLM virtual path. For example, if you access the 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*	Type of file being imported. If this option is omitted, the client tries to determine the filetype based on the MIME type of the import source file.
-h *host*	The Agile PLM server machine. Default: localhost
-i *input-file*	The source data file.
-l *port*	Port on which the Agile PLM server is listening. Default: 80
-m *map*	A textual mapping definition. Arguments should take the form of <source-path>=<target-path>.
-n *option*	An import server option. Arguments should take the form of <group>|<option>=<value>. Please see the Import XML Schema documentation for more information on available options.
-o *output-file*	The output file name. Default: log.xml
-p *password*	User's password.
-P *protocol*	URL protocol. Either http (the default) or https.
-t *type*	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*	Amount of time to wait for a response (in minutes). Defaults to 15 minutes.
-u *user*	Agile PLM username.
-x *transform*	A previously saved transformation definition file. For information on how to use the Import wizard to create a transformation definition file, see the *Agile PLM Import and Export Guide*.

Here are some examples showing 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>`

Table 2-4: importer.ImportSupplierResponse options

Option	Description
-e *virtual-path*	The Agile virtual path. For example, if you access the 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*	The Agile server machine. Default: localhost
-i *input-file*	The source data file.
-l *port*	Port on which the Agile server is listening. Default: 80
-o *output-file*	The output file name. Default: log.xml
-p *password*	User’s password.
-P *protocol*	URL protocol. Either http (the default) or https.
-r *RFQ-number*	The RFQ into which you are importing the supplier’s response.

Table 2-4: importer.ImportSupplierResponse options (continued)

Option	Description
-s *supplier-number*	Supplier number. 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.
-T *timeout*	Amount of time to wait for a response (in minutes). Defaults to 15 minutes.
-u *user*	Agile username.

Here are some examples showing how to run 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
-r RFQ00013 -s SUP11003
```

Creating a Web Service Client

Using the ExportData sample, this section illustrates steps for creating a web service client application for AIS.

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 on your behalf. This entails using a code generation utility along with the WSDL for the desired web service.

The provided AIS samples make use of Axis in order to connect with the AIS web service engine. Axis provides a WSDL2Java utility that can be used 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>
```

Note AXIS also includes an Ant task defintion, which you can use instead of the above <java> task. For more information about Axis Ant tasks, see the following Website: http://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, please consult the Axis documentation, which can be found 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. 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 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 it is with formatting, which makes it easier to read, write, and maintain.

Submitting the SOAP Request

The next step in consuming a web service operation is properly submitting the generated SOAP request to the web service engine. When dealing with generated client-side stubs, this step usually becomes as simple as pointing the stubs to the desired server and invoking a method on the stubs. You do not need to worry about opening a connection or manually marshaling your data onto the wire; instead, the generated stubs handle these details on your behalf.

The ExportData.java sample illustrates the above 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. Please consult the documentation for your Web-Service-aware code library for more information.

Processing the SOAP Response

As with submitting the SOAP request, the processing of the SOAP response is usually handled via the generated client-side stubs. Without these generated stubs, you would be responsible for parsing the XML-based SOAP response and dealing with the many formatting and unmarshaling issues that arise. However, when dealing with generated stubs, all of these details are taken care of for you, allowing you to receive properly type Java objects.

The ExportData.java sample illustrates this point nicely. In this sample, you can see that the result of the stub.exportData method is a javax.activation.DataHandler, which is a handy way of encapsulating a binary data stream. Rather than require you to parse an XML document and discern what the returned data is, the stubs automatically do this for you, returning the response's attachment to you as a DataHandler object.

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

Troubleshooting

If you experience trouble with replacing the Oracle XML Parser, refer to the section, “Replacing the Oracle XML Parser” below.

Replacing the Oracle XML Parser

If you have trouble building or running the Java samples, it may be due to a problem with Oracle’s XML parser. Contact Agile Support to get an older version of the file xmlparserv2.jar to use with AIS. Place the xmlparserv2.jar file that you receive from Agile Support in the following location on the Agile Application Server:

<*drive*>:\<*OracleHome*>\lib

To Locate the file, `xmlparserv2.jar` go to the Agile Support website www.agile.com/support and ask for an older version of the file xmlparserv2.jar to use with AIS.

CHAPTER 3

Extracting Agile Objects and Attachments

This chapter contains information about how to use AIS to export data. It has the following sections:

- *Understanding the Export Web Service*
- *Using the exportData Web Service Operation*
- *Using the exportPartlist Web Service Operation*

One of the primary functions of AIS is the ability to extract data into structured documents. This functionality is crucial for various integration scenarios.

Understanding the Export Web Service

There are two export web service operations that 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 in the BOM and their quantities 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, please see the Export XML Schema documentation.

The exportDataRequest XML element describes the XML format you should use when submitting an exportData request to AIS; it allows you to specify four different types of data:

- **Queries** — One or more queries that define what objects should be exported
- **Filters** — One or more filters that define what data from the selected objects should be exported.
- **Formats** — What format should be used for the exported data.
- **Sites** — Manufacturing sites for which data should be exported. By default, data for all sites is exported.

Working With Queries

The exportData web service operation allows you to specify several 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, see the *Agile SDK Developer Guide*.

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 (`'`).

Each search condition contains the following elements:

- **Left operand –** The left operand is always an attribute enclosed in brackets, such as `[Title Block.Number]`. You can specify the attribute as an attribute name (fully qualified name or short name) or attribute ID number. The attribute specifies which characteristic of the object to use in the search.
- **Relational operator** — The relational operator defines the relationship that the attribute has to the specified value. Relational operators are: “equal to” (==), “not equal to” (!=), “greater than” (>), “greater than or equal to” (>=), “less than” (<), “less than or equal to” (=<), “contains,” “does not contain,” “starts with,” “does not start with,” “is null,” “is not null,” “between,” “not between,” “in,” and “not in.”
- **Right operand** — The matching value for the specified attribute in the left operand. The right operand can be a constant expression or a set of constant expressions. A set of constant expressions is needed if the relational operator is “between,” “not between,” “in,” or “not in.”

Here is an example of a search condition:

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

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

```
[Title Block.Number] in ('1000', '2000')
```

You can also string multiple search conditions together using the logical operators “and” or “or.” Here is an example of search criteria that includes two conditions separated by the logical operator “and”:

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

An exportData Query Example

The following XML snippet illustrates how to specify a single query with the following conditions:

- Locate the part with the part number 1000-02.
- Export revision C

If you are exporting items, you can specify revisions by effectivity date, change number, or revision identifier. For more information, see the Export XML Schema documentation. If you don't specify a revision, the latest released revision is exported.

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

Working with Sites

Companies that practice distributed manufacturing use several different manufacturing sites for their products. The exportData web service operation allows you export data for all manufacturing sites or for a specific site. Manufacturing sites affect how items and changes are exported. For 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 not associated with that manufacturing site are filtered out of the query results.

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 allows you to define what information from the selected objects should be queried. 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 that allow you to refine query results. Predefined filters can be specified in one of 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. By using the ID of a defined filter, you reduce the risk of having a name change 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** — The set of possible filters includes one for each object type, allowing you to specify different information sets for each type.

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

For more information on the predefined filters, see the Export XML Schema documentation.

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 ennumerated values that let you include associated objects. For example, the BOM table supports the following ennumerated values for filters: `DoNotInclude` (the default), `TableOnly`, `SingleLevel`, and `Recurse`.

Note The filter type 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 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 "AIS Web Service Operations," on page 1-5.

For more information on how to specify the supported formats, see the Export XML Schema documentation.

An exportData Format Example

The following illustrates how to extract data in PDX format:

```
...
<format>PDX</format>
...
```

An exportData Sample

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"
                WhereUsed="true"
                 AML="TableOnly" Site="true"/>
  </filters>
  <format>PDX</format>
</exportDataRequest>
```

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 hand. Instead, the client-side stubs generated by a Web-Service-aware code library will usually take care of creating an appropriately formatted XML document and placing 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.

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. exportPartlist calculates the QPTLA for each unique item-revision pair, and returns the results in the Part Quantities element of the resulting aXML output.

Note ExportPartlist exports data only for items with AMLs (approved manufacturer parts and their associated manufacturers). Items without AMLs are ignored.

The format of the exportPartlist request is very similar to the exportData request structure and is encapsulated by the exportPartlistRequest XML element. Only the differences between the exportPartlistRequest and exportDataRequest structures will be covered in this section. For more information on the exportPartlistRequest element, please refer to the Export XML Schema documentation.

Working With exportPartlist Queries

exportPartlist accepts query definitions in almost the exact same fashion as exportData. 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 partlist must always be queries against items.

Working With exportPartlist Filters

Filters are specified for the exportPartlist web service operation in much the same way as they are for the exportData web service operation. The only difference is which filters can be specified. Since exportPartlist 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 exportPartlistRequest element, which demonstrates a complete exportPartlist web service operation request. You will quickly notice that this is a simple adaptation of the prior exportData sample.

```
<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>
```

Two things have been removed in order 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 aXML format.

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 hand. Instead, the client-side stubs generated by a Web-Service-aware code library will usually take care of creating an appropriately formatted XML document and placing 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.

CHAPTER 4
Importing Data

This chapter contains information about how to use AIS to import data into the Agile PLM system. It has the following sections:

- *Understanding the Import Web Service*
- *Using the importData Web Service Operation*
- *Importing Date Values*

You can use the AIS importData web service to import data into the Agile PLM system. 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—the 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, please refer to the *Agile PLM Import and Export Guide*.

Understanding the Import Web Service

There are two import web service operations that are delivered as part of 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 datastructure. For more information, see "Importing Supplier Responses" on page 4-5. Although the old ImportSupplierResponse web service operation is supported for this release, you should migrate your code to the new API.

Using the importData Web Service Operation

The importData web service operation exposes all Import Server functionality through a web service interface, which can be accessed programmatically.

This section will help illustrate how to format an XML request in order to use the importData web service operation. For more information on the XML schema that describes an importData request, please see the Import XML Schema documentation.

The importDataRequest XML element describes the XML format you should use when submitting an importData request to AIS. It supports two types of XML datastructures:

```
<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: an URL which 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.

More information on these parameters can be found in the Import XML Schema documentation.

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

```
<importDataRequest xsi:type="importDataRequestType">
  <dataSource>
    <attachmentRef href="cid:E36C913548344EDA1B7FC20CEDCEDEB3"/>
    <type>IPC2571</type>
  </dataSource>
  ...
```

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

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.

Table 4-1: Import operations

Operation	Child Attributes
currencyConversion	n/a
customers	n/a
items	aml, bom, sites, attachments
manufacturerParts	attachments

Table 4-1: Import operations (continued)

Operation	Child Attributes
manufacturers	attachments
prices	priceLines, attachments
productServiceRequests	affectedItems
programs	n/a
projectItems	aml, bom, attachments
quoteHistories	quoteHistoryLines
supplierCommodityOfferings	n/a
supplierManufacturerOfferings	n/a
suppliers	n/a

Depending on what you specify, import server will perform the desired import operations, ignoring certain data if it is not relevant to the selected import operations. For more information on import operations, see the Import XML Schema documentation.

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

```
...
<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. Mappings may be specified 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.

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

```
...
<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 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 allow you to specify how to transform data as it is imported into the Agile PLM system. Transforms are specified via previously defined transformation definition files, as indicated in the following snippet.

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

In the above snippet, the href attribute 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.

Working With Options

The import server contains a number of options which you can set in order to alter its behavior. 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.

The following snippet illustrates how to set several Business Rule and Parsing & Validation options.

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

An importData Sample

The following is a complete sample importDataRequest, which demonstrates how a fully configured importData operation request might appear.

```
<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 hand. Instead, the client-side stubs generated by a Web-Service-aware code library will usually take care of creating an appropriately formatted XML document and placing 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.

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** — The source of the data to be imported.
- **RFQ Number** — The alphanumeric identifier of the RFQ that is associated with the response.
- **Supplier Number** — 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.

The following is a complete sample importSupplierResponseRequest, which demonstrates how a fully configured importSupplierResponse operation request might appear.

```
<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's simply an illustration of what the XML request generated by client-side stubs might look like.

Importing Date 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 the Agile Web Client, click **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:

Table 4-2: Date formats for the U.S. locale

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

Table 4-2: Date formats for the U.S. locale (continued)

Date Format	Example
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 Sun's documentation for the `SimpleDateFormat` and `DateFormat` classes:

http://www.javasoft.com/j2se/1.3/docs/api/index.html

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` = Greenwhich Mean Time

`Sign` = + or -

`h` = hour in AM/PM (1~12)

`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 supports a relaxed version of the ISO String date format: yyyy/MM/ddTHH:mm:ssZ. The T and Z characters are optional.

INDEX

W

X