# Cisco AON Programming Guide

AON Release 2.2
August 2006

**Corporate Headquarters**
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

*Cisco AON Programming Guide*

# CONTENTS


# Introduction

The Application-Oriented Network (AON) system provides extensibility with several sets of application programming interfaces (APIs) and associated classes, packaged as software development kits (SDKs). This guide assumes that you are familiar with commonly accepted java software development conventions and with AON functionality and terminology, and describes the SDKs and their use.

This chapter introduces AON programmable features in the following sections:



This guide presents conceptual information, step-by-step instructions, and specifications, in the following chapters:

- Chapter 2, “Custom Bladelets”
- Chapter 3, “Custom Adapters”
- Chapter 4, “AON Optimization - Fastpath”
- Chapter 6, “XSLT Transformation”
- Chapter 7, “Schema Validation”
- Appendix A, “AONSCommon Specification”
- Appendix B, “AON Data Types”

For more about AON, see *AON Administration and Installation Guide* and *AON Development Studio Guide*.

## AON Extensibility

As part of AON, Cisco provides a set of built-in bladelets and adapters that are designed to meet many commonly encountered message-handling needs. If these do not meet all your company’s operating requirements, you can easily extend AON functionality. You may need to handle messages in a different way, or to convert a specific customer’s message output to a different format. You can address these and other unique message-handling requirements by extending AON functionality to create new custom bladelets and/or adapters.

- Custom Bladelets

  The Custom Bladelet SDK provides a framework to develop new bladelets that can be incorporated into new or existing AON PEPs (policy execution plans). After they are incorporated into AON, the new bladelets work within the system in the same way as other bladelets. For more details about bladelet customization, see Chapter 2, “Custom Bladelets”.

- Custom Adapters

  Your network traffic may include custom protocols and proprietary message types. If the AON includes built-in adapters—for converting frequently-used message protocols—do not meet all of your requirements, you can use the Custom Adapter SDK to create new adapters for the custom protocols and message types. For details, see Chapter 3, “Custom Adapters”.

- Optimized Processing with Custom Bladelets - Fastpath

  AON supports custom bladelets that are developed for optimized processing. This service is described in Chapter 4, “AON Optimization - Fastpath”.

- External Service Extensions

  You can use External Service Extensions APIs to develop custom bladelets and adapters with extended functionality and simultaneously reduce PEP complexity. This enhancement is explained in Chapter 5, “External Services”.

- Transformation

  AON’s XSLT Transformation (Transform and Content Parser) can transform a message or part of a message to fit requirements at the sending end, the receiving end, or both. This feature can be used to transform an XML message to HTML, a non-XML message to XML, and to perform other types of transformations. For details, see Chapter 6, “XSLT Transformation”.

- Schema Validation

  AON can validate incoming XML messages to verify that they adhere to a specific schema or DTD. For details, see Chapter 7, “Schema Validation”.

- AON Common Specification

  A large number of classes and interfaces are used by both the Custom Bladelet and Custom Adapter SDKs. For details, see Appendix A, “AONSCommon Specification”.

- AON Data Types

  AON-specific and Java-based data types are used in AON operations. For details, see Appendix B, “AON Data Types”.

# AON Programming Requirements

Each set of APIs and related set of extension procedures has associated programming requirements as follows:

Core API

  - Must have a well defined execution and exception handling model.
  - Must conform to bladelet SDK specifications (for example, when extending AbstractCustomBladelet, you should adhere to the requirements associated with the Bladelet SDK).
  - Must provide access to PEP context variables.
  - Must provide a mechanism to handle external libraries.
  - Must provide a mechanism to create attribute domains.
  - Must provide a mechanism to define input and output parameters.

- Exception Handling
  - Must follow AON PEP exception management specification.

  - Must provide error logging interfaces.
- Sandbox
  - Must not terminate AON execution.
  - Must restrict access to other AON Java code.
  - Must not invoke other bladelets
  - Must protect AON security features
- Utilities
  - Must provide utility classes for common functions.
  - May expose transformation services.

This guide discusses packaging, programming and design such as the use of the bladelet-info.xml file (mentioned above). For examples, see Using the AON Bladelet Schema, page 2-17 and Bladelet Info File, page 2-18.

# Where to Go Next

For more information about AON, see the following documents:

- *AON Development Studio Guide*
- *AON Installation and Administration Guide.*

CHAPTER 2

# Custom Bladelets

The Application-Oriented Network (AON) includes software development kits (SDKs) for developing custom bladelets. You can use these collections of application programming interfaces (APIs) and associated classes to create custom bladelets for AON. These features are discussed in the sections listed below.



For additional information, see the *AON Installation and Administration Guide* and *AON Development Studio Guide*. See Chapter 4, “AON Optimization - Fastpath” for more about extending custom bladelets.

# Conventional Custom Bladelets

This section introduces conventional custom bladelets. These user-designed bladelets can use the complete set of AON bladelet support features.



## Custom Bladelet SDK

The AON Custom Bladelet SDK contains the Java interfaces and classes needed for building custom bladelets. In addition it contains utility libraries that can be used to handle XML parsing, XPath, and other services. These components are identified below.

The Custom Bladelet SDK is incorporated into each AON release as a set of packages that includes:

- Custombladeletsdk.jar—Java archive file containing:
  - AbstractCustomBladelet—This is the basic custom bladelet class. It implements the CustomBladelet interface and must be extended in all new custom bladelets.

**Note** When you extend AbstarctCustomBladelet, pass the message as a parameter, call 8-per, and override the onCreate method.

  - CustomBladelet—All custom bladelets must implement this interface. However, the SDK provides a basic implementation of the interface. You can extend this basic implementation instead of implementing the interface.
  - CustomBladeletContext—This interface provides context to the custom bladelet. It is mainly used to pass variables, get Policy Execution Plans (PEP) details, set output path and log messages.
  - Manifest—Simple manifest file.
- JavaDocs—These JavaDocs provide summary descriptions of AbstractCustomBladelet, CustomBladelet, and CustomBladeletContext.

**Note** The AONSCommon set of interfaces and classes is used in conjunction with the Custom Bladelet API. For a detailed description, see AONSCommon Specification.

## Setting Up the Custom Bladelet SDK

The Custom Bladelet SDK does not require a complex installation process. However, you must include the SDK archive file in the development environment. Follow the steps listed below.

**Step 1** Install the CustomBladelet package in a separate directory.

**Step 2** Add the SDK's Java archive files (.jar) in the classpath.

—or—

Edit the ant script so that it adds the SDK's .jar in the classpath.

# Designing Custom Bladelets

You should consider the following while designing custom bladelets:

- how the new custom bladelet functions
- what properties to make configurable (by users) in the custom bladlet
- how runtime environment, installed bladelets and existing policies could affect custom baldelet
- the need for Fastpath or regular processing

The AON Development Studio (ADS) allows you to incorporate your newly created bladelets into either new or existing Policy Execution Plans (PEPs). For more information, see Policy Execution Plans, page 2-43 and the AON Bladelet Reference in the *AON Development Studio Guide*.

**Note** AMC is unable to upload custom bladelets larger than 50MB in size. If your custom bladelet exceeds 50MB, please contact your Cisco representative.

## Custom Bladelet Design Requirements

Custom bladelets must conform to the general requirements listed in the following sections:

- Custom Bladelet Class and Display Name, page 2-3
- Custom Bladelet Code Requirements, page 2-3
- Custom Bladelet Model, page 2-3

### Custom Bladelet Class and Display Name

The custom bladelet must be uniquely identified by the following:

- Class—Used internally as an identifier for custom bladelets.
- Display name—A short, user-friendly name for display.

### Custom Bladelet Code Requirements

The custom bladelet code requirements are as follows:

- Java 1.4.2
- Linux
- AON Bladelet Schema

  See Using the AON Bladelet Schema, page 2-17.
- Custom Bladelet API
  - Extend AbstractCustomBladelet *or* Implement ICustomBladelet interface

For details, see the Developing a Conventional Custom Bladelet, page 2-9.

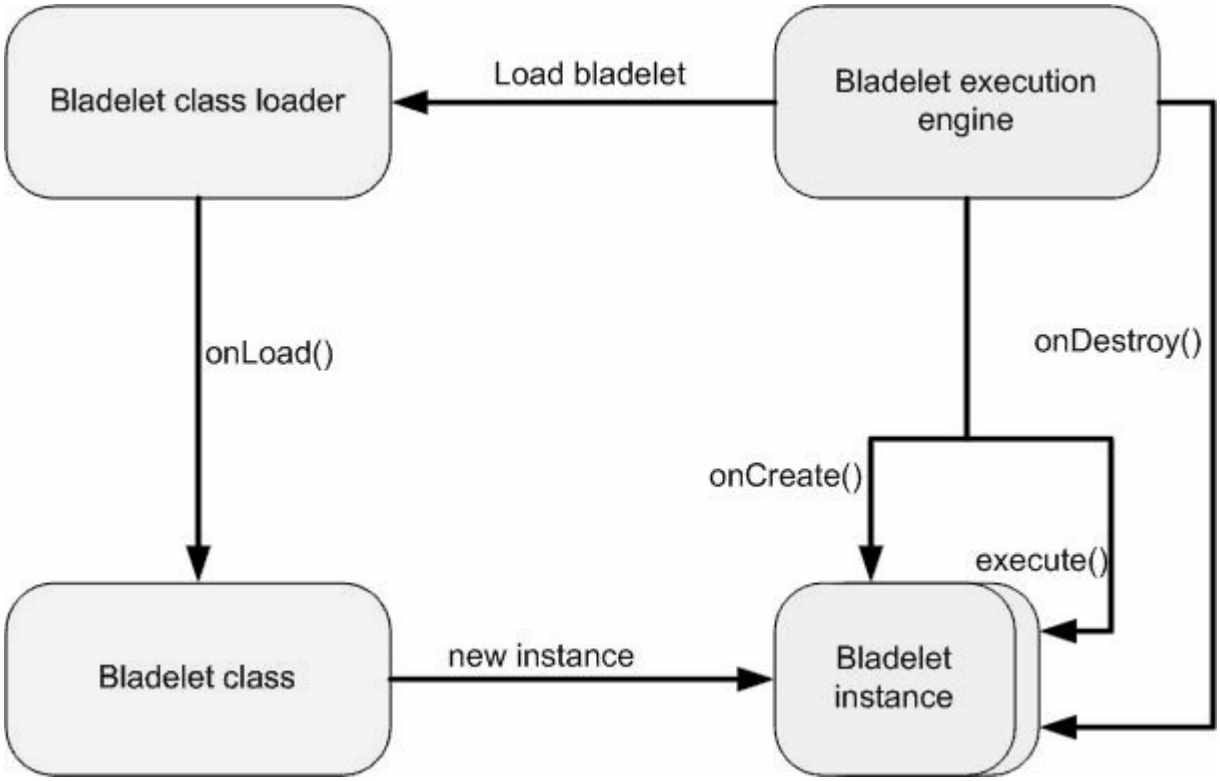
### Custom Bladelet Size

AMC is unable to upload custom bladelets larger than 50MB in size. If your custom bladelet exceeds 50MB, please contract your Cisco representative.

### Custom Bladelet Model

The next figure illustrates custom bladelet activity (state transitions).

*Figure 2-1 Custom Bladelet Activity*




- onLoad—This method is invoked by the container whenever a custom bladelet class is loaded by the class loader. It is called only once in the entire life cycle of the custom bladelet.
- onCreate—In contrast to onLoad, the onCreate method is called once whenever a new instance of a custom bladelet is created.
- execute—This method is invoked when a custom bladelet instance executes. It has the core business logic of the custom bladelet.
  - Data access—Custom bladelets exchange data with other bladelets using Policy Execution Plans variables. These PEP variables are available through the PEP context.
  - Error handling—The custom bladelet uses AON logging and execution handling.
- onDestroy—This method is invoked when the custom bladelet instance execution is complete and the AON execution engine is not using the custom bladelet.
- onUnload—This method is invoked whenever the class loader unloads the class from the virtual machine. This method should free up any resources that were allocated by the onLoad method.

**Caution** Use this method carefully because AON will not know the time that the class is unloaded from the virtual machine (VM).

For more information, see the *AON Installation and Administration Guide*.

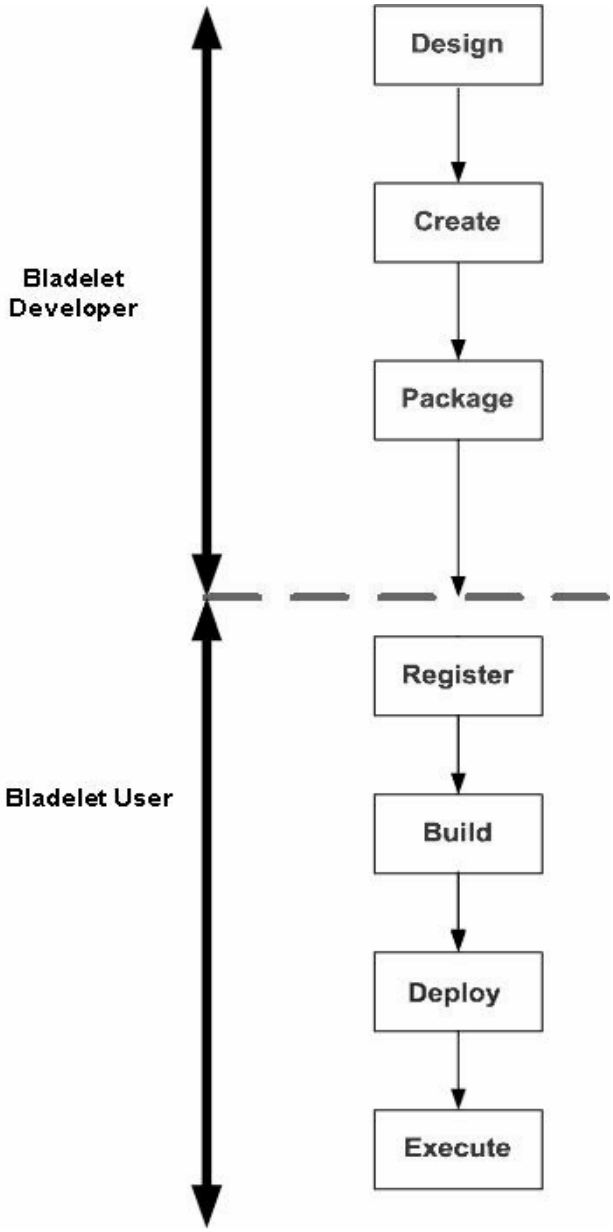
# Bladelet Development Life Cycle

The process or life cycle of custom bladelet development involves the following tasks:

1. Extending bladelet classes and implement bladelet interfaces to achieve desired functionality
2. Implementing the bladelet_info.xml file. This file defines the GUI appearance of the custom bladelet in the AON Developer Studio (ADS).
3. Using the ADS to package the new custom bladelet.
4. Using the ADS to upload the new custom bladelet to AMC.

For detailed steps, see Developing a Conventional Custom Bladelet, page 2-9.

*Figure 2-2 AON Custom Bladelet Development Life Cycle*




The figure depicts processing steps undertaken by bladelet developers and users.

**Bladelet Developer**

- Design—The developer decides what functions the custom bladelet will provide, how it can be configured, what external libraries it requires, and any interoperability issues with existing bladelets.
- Create—The developer uses the Custom Bladelet SDK and a Java editor (such as JBuilder or Eclipse) to write the Java code that will provide bladelet functionality. At a minimum, the developer must extend the AbstractCustomBladelet and override the execute method of this class. The developer can override other methods to control the behavior of the custom bladelet. For a description of this class, see the Code, page 2-33.

- Package—The custom bladelet must be packaged so that it can be recognized by AON. To package the new custom bladelet, the developer uses the ADS to collect all the custom bladelets and related files into an archive (.scar file, for custom bladelet archive). The scar files (containing metadata needed to build PEPs and execute the custom bladelet) are uploaded to the AON device. The package name can be stated in dotted notation so that the package is uniquely identified in the AON device. See Policy Execution Plans, page 2-43.
- Upload to AMC—As a final step, you upload the new custom bladelet package to AMC. See Uploading the Custom Bladelet to AMC, page 2-15.

**Bladelet User**

- Register—The user uploads the packaged custom bladelet (.scar file) to the AON Management Console (AMC). This makes the custom bladelet available to the ADS to build PEPs. Custom bladelets and bladelets that have been loaded in AMC are indistinguishable to the ADS.
- Build—Working with the ADS, the user can use the uploaded custom bladelet and other available bladelets to construct PEPs.
- Deploy—When the PEP has been designed, the user works with the AMC to deploy both the custom bladelet and new PEPs to the AON nodes where they will be executed.
- Execute—Custom bladelets that have been deployed to a device can be executed like any other bladelet in a PEP. The AON execution engine ensures that the custom bladelet code being executed conforms to the sandboxing requirements of the custom bladelet SDK.

The procedures associated with these activities are in Developing a Conventional Custom Bladelet, page 2-9. For a complete description of all user activities, see the *AON Development Studio Guide* and *AON Management Console Guide*.

# Custom Bladelet Use Cases

You can develop a custom bladelet to provide almost any type of AON message processing. For example, two custom bladelet examples are shown in the RnetBladelet1, page 2-47 and the EmailBladelet, page 2-49. These examples show:

- Extracting context variable data from an incoming purchase order (received as an XML message)
- Putting the extracted data into an outgoing e-mail message.

For more information, see Developing a Conventional Custom Bladelet, page 2-9.

# Designing Custom Bladelets to Restrict Permissions - Sandboxing

The AON sandbox feature allows for control, cutomization and restriction of permissions for custom bladelets. At the same time, sandboxing protects AON by restricting certain permissions which cannot be customized.

Bladelet package-level sandbox policies cannot override any restrictions imposed by the Global policy. AON parses the internal AON policy file (which cannot be user updated) that contains default restrictions. At bladelet deployment time, AON parses the sandbox policy file as defined in the SCAR package, and issues warnings if the Custom Bladelet package level sandbox policy overrides any globally imposed restriction.

If there are any violations such as granting the default restrictions, a sandbox warning message is issued. The Custom Bladelet package level sandbox policy file should be named sandbox-policy.xml.

- Sandboxing and Java Based Security Permissions, page 2-8

For more information, see the *AON Development Studio Guide*.

## Sandboxing and Java Based Security Permissions

Generally, the AON sandbox service is based on Java (jdk) security permissions. Permission is a specific action that code is allowed to perform. It includes three elements: Type, Name, and Actions. For example:

```
java.io.FilePermission ("/tmp/a.txt","read");
..............Type.............................Name........Actions
```

Sandbox codebase—This is the location from which a class has been loaded (used in policy files). For example:

```
grant codebase:file:/usr/home" {
permission java.io.FilePermission("/tmp/a.txt","read");}
```

## Sandbox Permission Types

The AON sandbox service (also applicable to Custom Adapters) provides:

- File level access—Controls the access level of files/directories.
  - Consists of a filepath name and a set of actions valid for the name.
  - Controlled actions: read, write, delete, execute.
- Network level access—Controls various network permissions. Contains no actions and only names.
  - Controlled actions: specifyStreamHandler, setDefaultAuthenticator.
- Socket access—Controls access to a network via sockets. Consists of host as "name" and a set of actions.
  - Controlled actions: accept, connect, listen, resolve.
- Property access—Controls property permissions. Consists of a name and a set of actions.
  - Names: java.home, AON.home.
  - Controlled actions: read, write.
- Security access—Controls security permissions. Consists of a name and no actions.
  - Names: getPolicy, setPolicy.
- Runtime access—Controls runtime permissions. Consists of name and no action list.
  - Names: createClassLoader, createSecurityManager, exitVM.

AON sandboxing does not include:

- Thread management
- Memory management
- CPU management
- Administration console

Out-of-the-box, AON sandboxing restricts the following permissions:

- VM exit (system.exit)

- Customers to set their own security manager
- Customers to create their own security manager
- All Security permissions

For more information, see the *AON Installation and Administration Guide*.

## Sandbox Policy: Schema

The schema shown below is used to create a sandbox policy file.

```
<?xml version='1.0' encoding='UTF-8' ?>

<!ELEMENT policy (extension-policy+)>

<!ELEMENT extension-policy (grant+)>

<!ATTLIST extension-policy type CDATA #IMPLIED>

<!-- Restricts the required permissions. .java.policy format.. No change but in xml
representation ->
<!ELEMENT restrict (permission+)>

<!ATTLIST restrict codebase CDATA #IMPLIED>

<!-- The restriction element is similar to .java.policy format. No change but in xml
representation->
<!ELEMENT permission (name,target?,actions?)>

<!ELEMENT name (#PCDATA)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT actions (#PCDATA)>
```

## Sandbox Policy: Sample

The following sample XML code creates a sandbox.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<policy>
<extension-policy type="aons">
                <restrict codebase="file:/{SCRIPTLET}/-">
                <permission>
                        <name>java.net.SocketPermission</name>
                        <target>mailman.cisco.com</target>
                        <actions>connect,resolve</actions>
                </permission>
        </restrict>
        </extension-policy>
</policy>
```

# Developing a Conventional Custom Bladelet

Developing a custom bladelet (either conventional, or optimized for Fastpath) involves the following tasks:

- Creating the Custom Bladelet, page 2-10
- Packaging the Custom Bladelet, page 2-10

- Uploading the Custom Bladelet to AMC, page 2-15

For more information, see the *AON Development Studio Guide*.

# Creating the Custom Bladelet

After installing the SDK, create your custom bladelets using the guidelines outlined in the following steps:

**Step 1** **Write the custom bladelet code.**

Using a Java editor, write code that, at a minimum:

- Conforms to the bladelet-info.xsd format.
- Includes a subclass extended from AbstractCustomBladelet.
- Overrides the execute method of AbstractCustomBladelet.

See the Using the AON Bladelet Schema, page 2-17, Custom Bladelet Samples, page 2-47, and the Code, page 2-33.

**Step 2** **Compile the custom bladelet code**.

Include all library/.jar files, resource files (including policy and attribute domain), and the bladelet-info.xml file.

You can use the Ant scripting tool to compile and later run your new Java custom bladelet program.

**Note** The bladelet-info.xml file, provided with AON, defines the complete set of supplied bladelets. For more information, see Bladelet Info File, page 2-18.

**Step 3** **Package the files**.

Using the AON Development Studio (ADS), package the custom bladelet files as a scar file. For descriptions of the packaging steps, see Packaging the Custom Bladelet, page 2-10.

**Step 4** **Use AMC to upload and register the custom bladelet**

This action uploads the custom bladelet package to the AMC. Follow the steps Uploading the Custom Bladelet to AMC, page 2-15. F or additional information, see the *AON Installation and Administration Guide.*

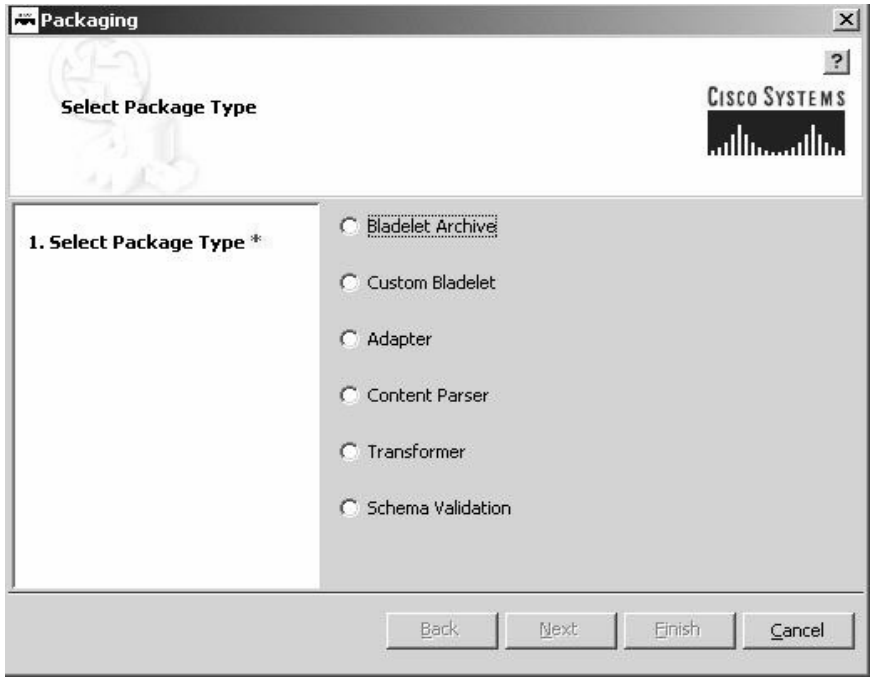
# Packaging the Custom Bladelet

Use the same ADS windows to package all custom bladelets. Later, you use AMC windows to upload the package to AON. Follow the steps listed below.

**Step 1** Using the ADS, select **Tools > Packaging**.

The AON Packaging window appears.

*Figure 2-3 AON Packaging Window*



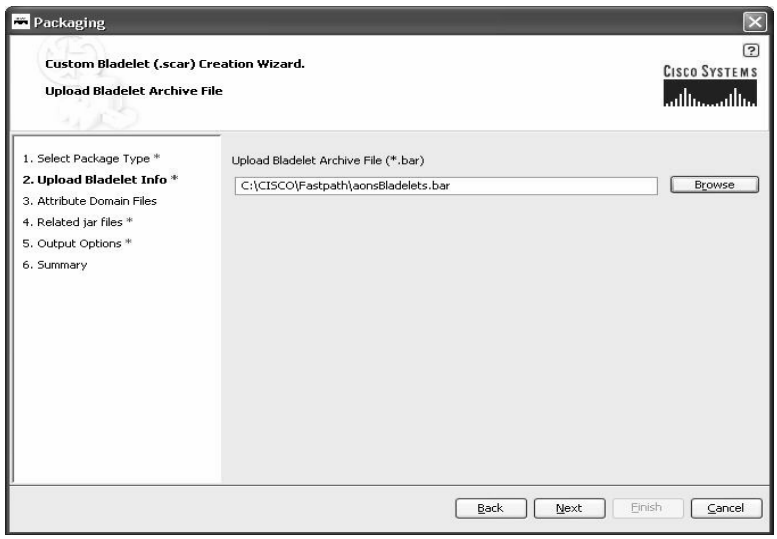

**Step 2** Select **Custom Bladelet** and click **Next**.

The Custom Bladelet (.scar) Creation Wizard - Upload Bladelet Archive file screen appears.

**Note** Each custom bladelet archive file with all its components has the file extension .bar.

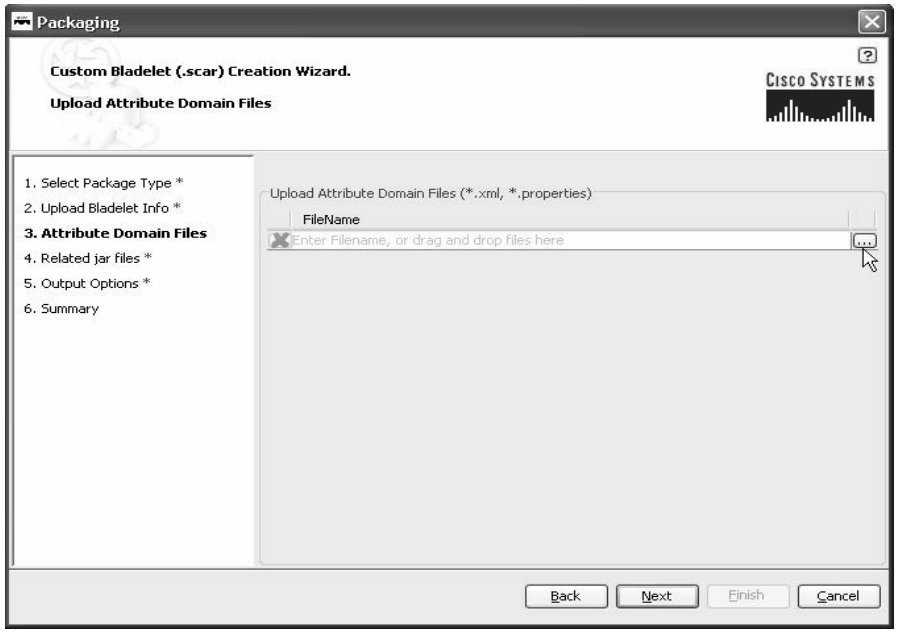
**Step 3** Click **Browse** and locate the bladelet.bar (archive) file for the new custom bladelet, then click **Next**, as in the following screen shot:

*Figure 2-4 Upload Bladelet Archive File window*



The Upload Attribute Domain Files screen appears. For example:

*Figure 2-5 Upload Attribute Domain Files window*

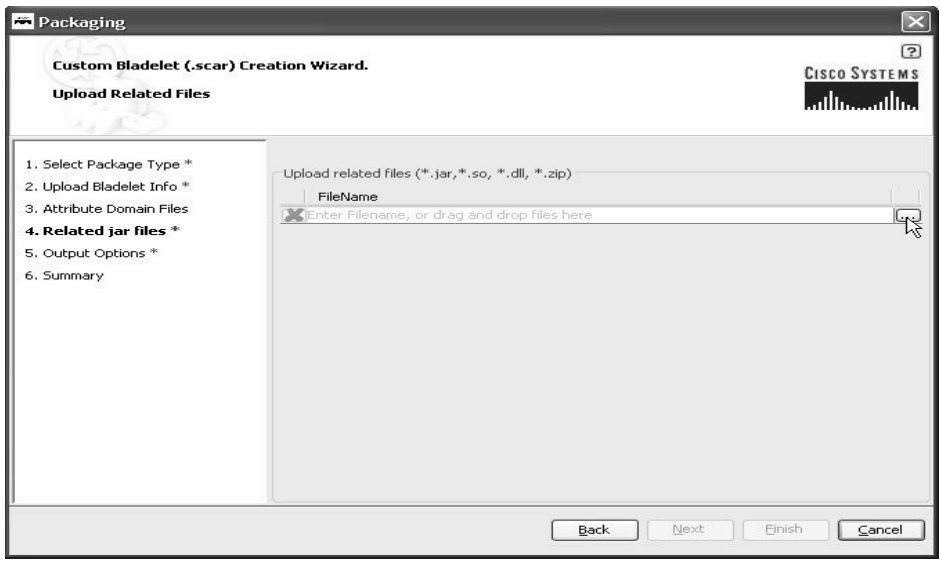



**Step 4** Click the Search button, locate the file(s), then click **Next.**

Search for the .xml and/or .properties files associated with the new custom bladelet.Generally, attribute domain files contain policies. In addition, profiles are associated with attribute domains. Contexts are associated with property sets.

The Upload Related Files screen appears.

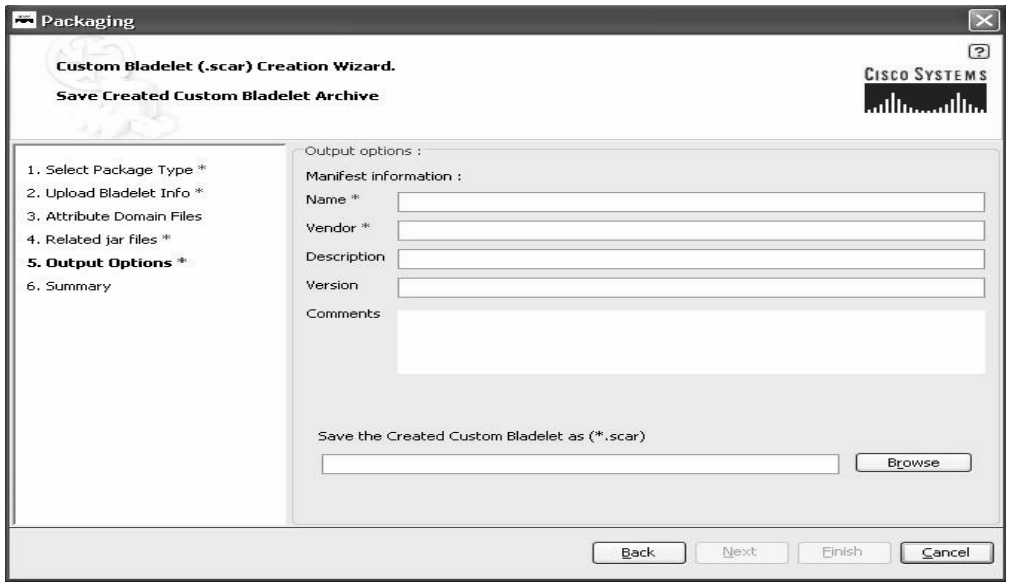
*Figure 2-6 Upload Related Files window*




**Step 5** Click **Search** to locate related files (for example, aonscommon.jar), then click **Next.**

The Save Created Custom Bladelet Archive - Output Options screen appears.

*Figure 2-7 Output Options window*



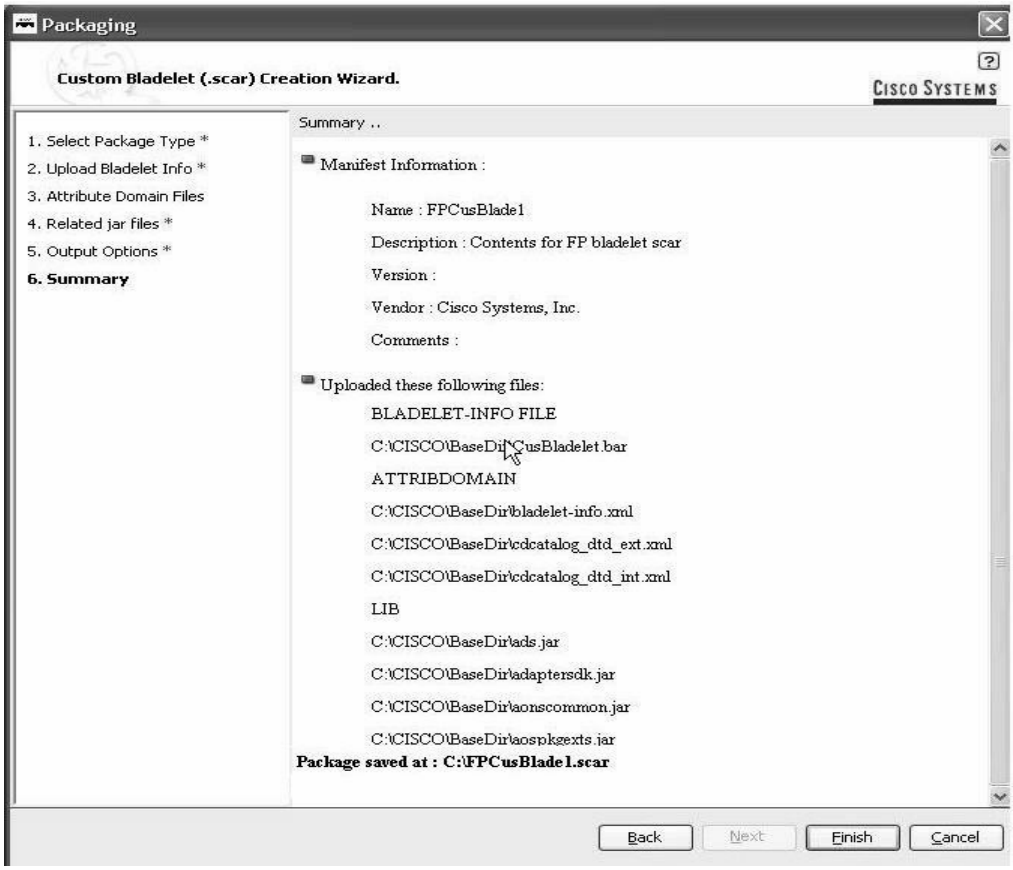

**Step 6** Fill in the manifest information and click **Next**.

*Table 2-1 Manifest Information Fields*

| Field | Description |
|---|---|
| Name | Manifest name. |
| Vendor | Vendor. For example, Cisco Systems, Inc. |
| Description | (Optional) Description of the manifest contents. |
| Comments | (Optional) Comments about the manifest. |
| Save the Created Custom Bladelet as | Automatically filled in the Name field name and current default location. You may use the Browse button to find a different location. |

A Packaging Summary screen appears. For example:

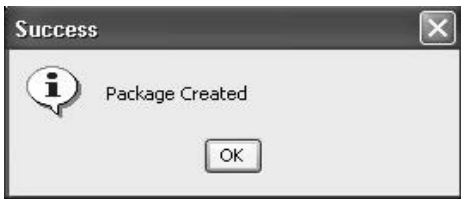
***Figure 2-8 Summary window***




**Step 7** Click **Finish** to complete the package.

A Success message displays.

***Figure 2-9 Success Message dialog***




**Step 8** Click **OK**.

The main ADS window displays.

The new custom bladelet package will contain:

- A Java archive file (.jar) that contains the Java version of the bladelet.
- Additional .jar files and shared objects required by the implementation.
- Optionally, a native shared object that contains the Fastpath bladelet implementation. For more details, see Developing a Custom Bladelet for Fastpath Processing, page 4-1.

Next, you can upload the package to AMC. See Uploading the Custom Bladelet to AMC, page 2-15. To learn more about AMC, see the *AON Installation and Administration Guide*.

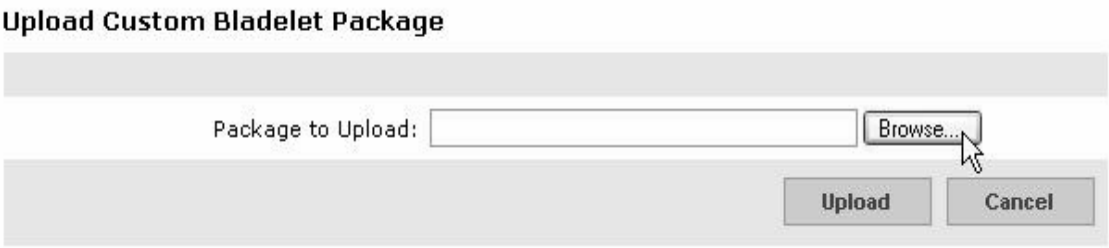
# Uploading the Custom Bladelet to AMC

The AON Management Console is used to upload custom bladelets to AON. Follow the steps described below.

**Step 1** Using AMC, select **Admin > Extensions > Custom Bladelets & Extensions > Upload > Upload Custom Bladelet Package**.

An upload dialog appears.

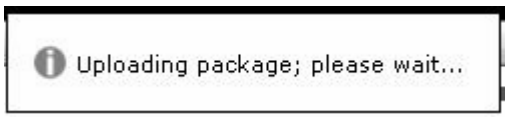
***Figure 2-10 Uploading the Custom Bladelet Package dialog***




**Step 2** Press **Browse** to locate the package (.scar extension) and click **Upload**.

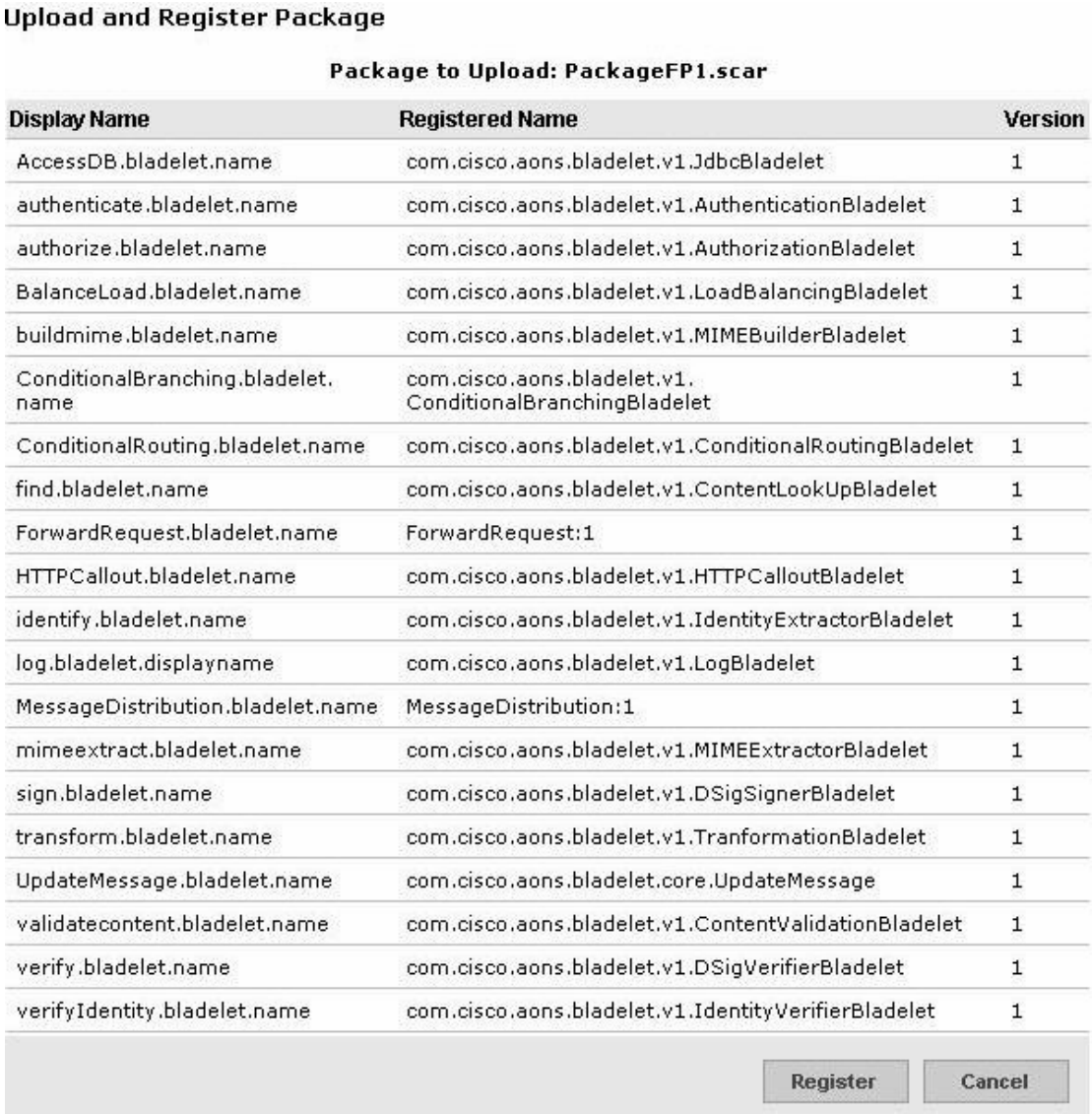
A message appears as the process occurs. The upload may take a minute or more.

***Figure 2-11 Uploading package message***




After the upload is complete, aan Upload and Register Packaage: Package to Upload window appears. For example:

*Figure 2-12 Uploading and Registering the Package*



Upload and Register Package

Package to Upload: PackageFP1.scar

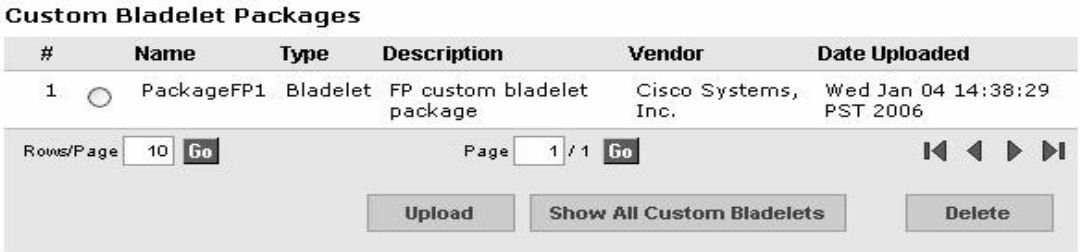
| Display Name | Registered Name | Version |
|---|---|---|
| AccessDB.bladelet.name | com.cisco.aons.bladelet.v1.JdbcBladelet | 1 |
| authenticate.bladelet.name | com.cisco.aons.bladelet.v1.AuthenticationBladelet | 1 |
| authorize.bladelet.name | com.cisco.aons.bladelet.v1.AuthorizationBladelet | 1 |
| BalanceLoad.bladelet.name | com.cisco.aons.bladelet.v1.LoadBalancingBladelet | 1 |
| buildmime.bladelet.name | com.cisco.aons.bladelet.v1.MIMEBuilderBladelet | 1 |
| ConditionalBranching.bladelet.name | com.cisco.aons.bladelet.v1.ConditionalBranchingBladelet | 1 |
| ConditionalRouting.bladelet.name | com.cisco.aons.bladelet.v1.ConditionalRoutingBladelet | 1 |
| find.bladelet.name | com.cisco.aons.bladelet.v1.ContentLookUpBladelet | 1 |
| ForwardRequest.bladelet.name | ForwardRequest:1 | 1 |
| HTTPCallout.bladelet.name | com.cisco.aons.bladelet.v1.HTTPCalloutBladelet | 1 |
| identify.bladelet.name | com.cisco.aons.bladelet.v1.IdentityExtractorBladelet | 1 |
| log.bladelet.displayname | com.cisco.aons.bladelet.v1.LogBladelet | 1 |
| MessageDistribution.bladelet.name | MessageDistribution:1 | 1 |
| mimeextract.bladelet.name | com.cisco.aons.bladelet.v1.MIMEExtractorBladelet | 1 |
| sign.bladelet.name | com.cisco.aons.bladelet.v1.DSigSignerBladelet | 1 |
| transform.bladelet.name | com.cisco.aons.bladelet.v1.TranformationBladelet | 1 |
| UpdateMessage.bladelet.name | com.cisco.aons.bladelet.core.UpdateMessage | 1 |
| validatecontent.bladelet.name | com.cisco.aons.bladelet.v1.ContentValidationBladelet | 1 |
| verify.bladelet.name | com.cisco.aons.bladelet.v1.DSigVerifierBladelet | 1 |
| verifyIdentity.bladelet.name | com.cisco.aons.bladelet.v1.IdentityVerifierBladelet | 1 |

Register Cancel

**Step 3** Click **Register**.

Soon, Custom Bladelet Packages appears.

*Figure 2-13 Registering the Custom Bladelet window*



Custom Bladelet Packages

| # | Name | Type | Description | Vendor | Date Uploaded |
|---|---|---|---|---|---|
| 1 | PackageFP1 | Bladelet | FP custom bladelet package | Cisco Systems, Inc. | Wed Jan 04 14:38:29 PST 2006 |

Rows/Page 10 Go Page 1 / 1 Go

Upload Show All Custom Bladelets Delete

**Step 4** Select the new package and click **Upload**.

The new custom bladelet package is uploaded to AMC. At this point, you may load additional packages, click **Show All Custom Bladelets** to view all uploaded bladelet packages, and click Delete if you want to remove the new package from AMC. F or additional information, see the *AON Installation and Administration Guide.*

# Using the AON Bladelet Schema

All AON bladelets, custom-developed or supplied with AON, are defined in the bladelet information file (bladelet-info.xml) which must conform to the bladelet schema (bladelet-info.xsd) file. After you create a custom bladelet (coded in Java), you create an associated bladelet-info.xml file based on the schema.

The bladelet information file (bladelet-info.xml) is included in the bladelet archive (.bar) file. This XML file describes the bladelet for the AON Development Studio (ADS).

This describes the AON Bladelet Schema, focusing on:

- Bladelet Archive File, page 2-17
- Bladelet Info File, page 2-18
- Bladelet Schema, page 2-24

## Bladelet Archive File

AON bladelets are defined in bladelet archive (.bar) files. This file stores metadata about one or more bladelets. When a message PEP is constructed, the AON Development Studio (ADS) uses information in the bladelet archive file to generate user interface and other code. The .bar file is also used at runtime to load the bladelet name and class name.

The bar file is a zip/jar file that is renamed to .bar by convention. All tools that work with zip or jar files also work with bar files. This archive file must contain the following components:

- Bladelet-info.xml

  The primary component of the archive, the bladelet-info.xml file contains XML elements that conform to the bladelet-info.xsd schema. This file is stored in the META-INFO folder in the bar file. For more information, see the Bladelet Info File, page 2-18.

- Resource bundles

  Each bladelet identifies a single resource bundle that contains all bladelet-referenced strings. This includes error messages generated by bladelet-defined validator classes. The base name of the bundle file is stored in the bladelet-info.xml file. All bladelets in a bar file may share the same resource bundle or a bladelet may have its own resource bundle.

- Validator classes

  Each bladelet defines a single validator class. This class is also included in the bladelet archive. The key for this validator class is stored in the bladelet-info.xml file.

- Icons

  Each bladelet specifies two icons. One icon (16 X 16) is used in the palette; the other (32 X 32) is used as the bladelet representation in the ADS PEP Developer. The bladelet-icon.xml file contains the pathname to the two icons.

- Validator rules

A validator rule is used to validate the user input of any bladelet parameters during PEP construction.It determines whether a bladelet parameter is correctly specified or not.

# Bladelet Info File

The bladelet-info.xml file stores metadata about bladelets. It is stored in the META-INFO folder in the bar file. All bladelet-info.xml files must conform to the bladelet-info.xsd schema, contained in the AON source in the AON/modules/bladeletInfo folder. This section describes the components of the schema with code section samples. For a description of the bladelet-info.xsd schema, see the Bladelet Schema, page 2-24.

## Bladelet Info File Attributes

At the top of the bladelet-info.xml file, the attributes section contain the general file information listed in the table below. Unless they are marked "(Optional)," attributes are mandatory.

Although you must include all mandatory (non-optional) attributes, a null value is acceptable.

***Table 2-2 Bladelet Info File Attributes***

| Attribute | Description |
|---|---|
| name | Name of the bladelet. Internal use. |
| versionId | Version number. |
| displayNameKey | (Optional) Key for the resource bundle for the display name. |
| categoryKey | Key for this bladelet's category. This attribute is used to sort bladelets in the ADS Task Pane display. |
| bladeletClass | Class name for the runtime bladelet. |
| terminal | (Optional) "True," if this bladelet is used to terminate a PEP. |
| allowInException | (Optional) "True," if this bladelet is used in an exception PEP. |
| bundle | Path to the resource bundle for this bladelet. It should be in the same syntax as a java class. For example, com.cisco.AON.BundleName. |
| validatorClass | Class name for the design time bladelet validator for this bladelet. |
| validatorRules | Path to the validator rules for this bladelet. It should be in java resource syntax. For example, com/cisco/aons/ValidationRules.xml. |

## Bladelet Info File Elements

The next section of the bladelet-info.xml file, the elements sections, contain the additional information listed in the table below. Unless they are marked "(Optional)," these elements are mandatory.

***Table 2-3 Bladelet Info File Elements***

| Element | Description |
|---|---|
| icon-ref | (Optional) Contains two sub-elements, palette-\icon and document-icon. Each one has an href attribute that contains the paths to the icon. |
| exceptions | (Optional) Contains any number of exception elements. |

*Table 2-3 Bladelet Info File Elements (continued)*

| Element | Description |
|---|---|
| bladelet-design | Contains a bladelet-parameters element which contains any number of configuration-group elements. See the next section below. |
| bladelet-deployment | Not used at present. |
| bladelet-runtime | (Optional) Contains data exported at runtime such as email. |
| output-paths | (Optional) Indicates that a determinate (“static value) or indeterminate (“dynamic”) number of output paths exist.The output path elements have a label attribute indicating the value that goes into the PEP and a key attribute to calculate the display name. |

## Bladelet Info File Parameters

AON bladelet parameters are displayed in the ADS by clicking on a bladelet icon and selecting Properties generating a screen showing properties (parameter information). The display is partially determined by the bladelet-info.xml file. In the file, parameters are grouped hierarchically:

```
<configuration-group>
    <configuration-subgroup>
        <parameter-group> and <parameter>
```

These groupings are described in the following sections.

### Configuration Group

This section contains components that act as a single selection or as mutually exclusive selections. Configuration groups may include any number of configuration subgroups, parameters, and parameter subgroups. In the sample Properties screen in Figure 2-14 on page 2-23, the Advanced screen area depends on the section of bladelet-info.xml code shown below the figure.

Configuration groups have the attributes identified in the following table.

*Table 2-4 Configuration Group Attributes*

| Attribute | Value | Description |
|---|---|---|
| type | “radio” | (Optional) Determines how the control appears in the GUI. If this is present, each configuration group that has the same group name behaves like a radio selection that only permits one group entry at a time. The various radio values appear a a combo box on the top of the right side. See Figure 2-14Bladelet Properties window, page 2-23. |
| name | string | Internal name for the group. All elements with this name are part of the same group. |
| value | string | Required, if a radio type. This value uniquely identifies a group member. This value is put into the far file (group.<*name*>.value) to identify the selected group member when the type attribute is set to radio. |
| valueKey | string | Required, if a radio type. Key that determines the display name of the radio value. If this attribute is not included in the configuration group, it has a default value. |

*Table 2-4 Configuration Group Attributes (continued)*

| Attribute | Value | Description |
|---|---|---|
| key | string | (Optional) Key the identifies the entry in the resource bundle that will be used as the display name of this group. If this attribute is not included in the configuration group, the group name will be the display name. |
| default | boolean | (Optional) Selects the default value of a radio configuration group. |

## Configuration Subgroup

This section of the bladelet-info.xml file is a child of the configuration-group. Each configuration-group must contain at least one configuration-subgroup. The subgroup determines the items in the right part of the Properties display, shown in the sample Figure 2-14 on page 2-23 as the area labeled "Trust Verification." Custom bladelet designers can create additional subgroups to give the bladelet desired functionality. In most cases; however, it will not be necessary to group parameters below the default subgroup. Configuration subgroups have the attributes listed in the following table.

*Table 2-5 Configuration Subgroup Attributes*

| Attribute | Value | Description |
|---|---|---|
| type | tab | (Optional) Determines how the control is displayed in the GUI. This attribute must be set to tab. |
| name | string | (Optional) Internal name for the subgroup. |
| key | string | (Optional) Resource bundle key. The tab name is based on this key. |

## Parameter Group

A parameter group is an optional child of a configuration subgroup. Each parameter group contains one or more parameters. This level of association groups logically related bladelet parameters together. A configuration subgroup may have any number of parameter groups or parameters. Parameter groups have the attributes listed in the following table.

*Table 2-6 Parameter Group Attributes*

| Attribute | Value | Description |
|---|---|---|
| name | string | (Optional) Internal name for the parameter group. If key (described below) is not specified, this attribute is used as the display name in the GUI. |
| type | radio or check box | (Optional) Type of group (for example, radio). If radio is specified, only one parameter in the group will be valid. For a check box, group 0 or more values may be valid. If this attribute is not included, ADS displays each parameter depending upon its editor attribute. |
| key | string | (Optional) Key in the resource bundle that is used as the displayed title of the parameter group. |

*Table 2-6 Parameter Group Attributes (continued)*

| Attribute | Value | Description |
|---|---|---|
| default | boolean | (Optional) Determines the default value of a radio parameter group. |
| value | string | Required, if this is a radio type.This value uniquely identifies a group member. This value is put into the far file (parameter-group.<*name*>.value) to identify the selected group member when the type attribute is set to radio. |

## Parameters

The parameters (listed below) are set and changed by the ADS user to configure the custom bladelet.

*Table 2-7 Parameter Attributes*

| Attribute | Value | Description |
|---|---|---|
| editor | string | (Optional) Determines how the control is displayed in the GUI. Defaults to the widget that corresponds with the parameter type. This attribute should be used carefully. |
| name | string | Internal name of the parameter. |
| key | string | (Optional) Resource bundle key providing the parameter label. Defaults to the parameter name. |
| type | string | Runtime type of the parameter. Must be a value defined in the datatypes.xml file. |
| allowUserInput | true/false | (Optional) Indicates whether the user can enter values manually for this parameter. Default = true. |
| allowVarBinding | true/false | (Optional) Indicates whether PEP variables can be bound for this parameter. Default = true. |
| defaultValue | string | (Optional) Default value for this parameter. In this attribute is not specified, this parameter is optional. |
| domain | string | (Optional) Domain associated with this parameter. |
| designName | string | (Optional) Unique identifier for this parameter. Required if the bladelet has a non-unique parameter name. For example, if two parameter groups have a parameter with the same name. designName defaults to the attribute name. This value must be unique in the bladelet. |
| cdataType | boolean | (Optional) This parameter value is placed in the PEP XML inside the CDATA tag section. Default = false. |
| output | boolean | (Optional) Indicates whether this parameter is an output (export) parameter. Default = false. |
| tooltipKey | sting | (Optional) Resource bundle key sued for the tooltip of this parameter. |
| optional | boolean | (Optional) Indicates if this parameter is optional or required. Default = false. |
| mapKeyColumn | string | (Optional) Indicates the column that is the key for the map. |

*Table 2-7 Parameter Attributes (continued)*

| Attribute | Value | Description |
|---|---|---|
| validate | boolean | (Optional) Indicates whether this parameter should be validated using the ADS validation framework. Default = true. If false, the system will only use the bladelet-specific validation using the validateClass attribute. |
| conditionRule | boolean | (Optional) If the parameter is a rule type, this parameter indicates whether this should be a condition-only rule or a rule with actions. Default = true. |

#### Allowed-Value Attributes

A parameter may have either allowed-values or column info elements as children, depending on the parameter type. If a parameter type is enum, an allowed-values element with any number of allowed-value sub-elements should be present to provide values for the parameter. The allowed-value element has the attributes listed below.

*Table 2-8 Allowed Value Attributes*

| Attribute | Value | Description |
|---|---|---|
| type | string or int | |
| default | boolean | Indicates to use the default allowed value. Default = false. |
| value | string | Value that goes into the PEP file. |
| valueKey | string | Resource bundle key that determines the text in the combo box. Default to value. |

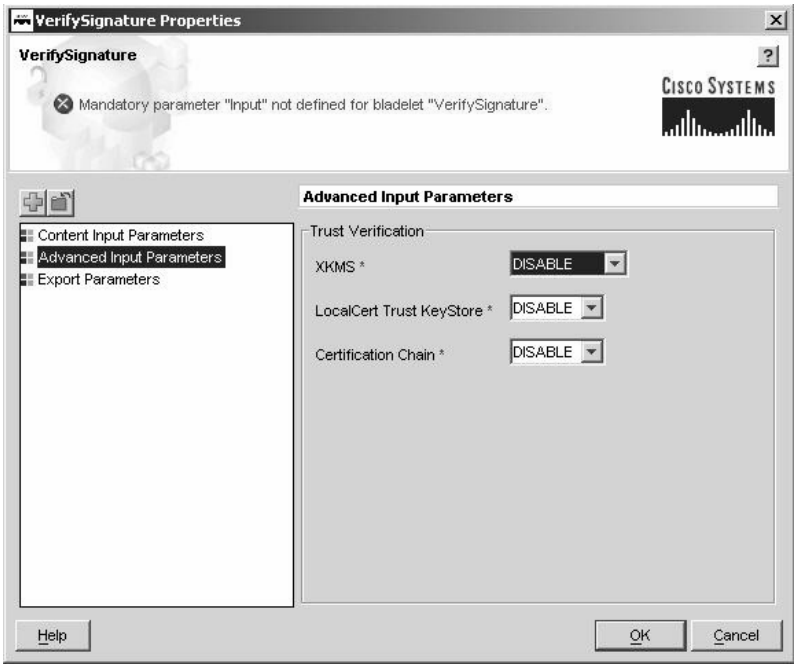
#### Column-Info Element

The column-info element is used if the type of parameter is list, map, or mappedList. This produces a table in the GUI with columns labeled by the individual column-info elements. A map or mappedList have a special column that is used as a row label, indicated by the mapKeyColumn attribute of the parameter. This should be equal to the name attribute of a c column-info element. The column-info element has the same attributes as the parameter element but it cannot be nested inside itself.

## Bladelet Properties Screen and Bladelet Info XML Code Sections

Sample Figure 2-14 has configuration group (Advanced) and configuration-subgroup (Trust Verification) features that are determined by the bladelet-info.xml code section shown below the figure.

*Figure 2-14 Bladelet Properties window*




The following section of the bladelet-info.xml file (Verify bladelet) includes the configuration group and configuration subgroup sections that define items in Figure 2-14. For the entire code sample, see the Verify Bladelet Info XML, page 2-45.

```
- <configuration-group name="Advanced" key="cg.advanced">
- <configuration-subgroup>
- <parameter-group name="Trust Verification"
key="cg.contenttoverify.pg.trustverification">
- <!--
 XKMS Trust Verification Mode - Disabled, Pilot mode or Production mode
-->
- <parameter name="XKMSTrustVerification"
designName="Advanced.TrustVerification.XKMSTrustVerification"
key="cg.advanced.pg.trustverification.p.xkms" type="string" allowVarBinding="false"
allowUserInput="false" editor="combo-box">
- <allowed-values>
<allowed-value type="string" value="Disable"
valueKey="cg.advanced.pg.trustverification.p.xkmstrust.disable" default="true" />
<allowed-value type="string" value="Pilot"
valueKey="cg.advanced.pg.trustverification.p.xkmstrust.pilot" />
<allowed-value type="string" value="Production"
valueKey="cg.advanced.pg.trustverification.p.xkmstrust.production" />
</allowed-values>
</parameter>
- <!--
 Local Certificate Trust Verification - Disabled or Enabled
-->
- <!--
 Whether the certificate that came with signature is explicitly trusted in the local trust
store
-->
- <parameter name="LocalCertTrustVerification"
designName="Advanced.TrustVerification.LocalCertTrustVerification"
key="cg.advanced.pg.trustverification.p.localcerttrust" type="string"
allowVarBinding="false" allowUserInput="false" editor="combo-box">
```

```
- <allowed-values>
<allowed-value type="string" value="Disable"
valueKey="cg.advanced.pg.trustverification.p.localcerttrust.disable" default="true" />
<allowed-value type="string" value="Enable"
valueKey="cg.advanced.pg.trustverification.p.localcerttrust.enable" />
</allowed-values>
</parameter>
- <!--
 Certificate Chain Trust Verification - Disabled or Enabled
-->
- <!--
 Whether the root of the certificate chain that came with signature is explicitly trusted
in the local CA trust store
-->
- <parameter name="CertChainTrustVerification"
designName="Advanced.TrustVerification.CertChainTrustVerification"
key="cg.advanced.pg.trustverification.p.certchain" type="string" allowVarBinding="false"
allowUserInput="false" editor="combo-box">
- <allowed-values>
<allowed-value type="string" value="Disable"
valueKey="cg.advanced.pg.trustverification.p.certchaintrust.disable" default="true" />
<allowed-value type="string" value="Enable"
valueKey="cg.advanced.pg.trustverification.p.certchaintrust.enable" />
</allowed-values>
</parameter>
</parameter-group>
</configuration-subgroup>
</configuration-group>
```

For more information about various bladelet properties windows, see the AON Development Studio Guide.

# Bladelet Schema

The schema (bladelet-info.xsd) for developing new bladelets is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="bladelet-info">
        <xs:annotation>
            <xs:documentation>Root element for the definition of all system
bladelets</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="bladelet" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="groupType">
        <xs:annotation>
            <xs:documentation>Defines the editor types for Configuration
Groups</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="radio"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="subgroupType">
        <xs:annotation>
```

```
            <xs:documentation>Defines the editor types for Configuration
Sub-Groups</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="tab"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="paramType">
        <xs:annotation>
            <xs:documentation>Defines the parameter types supported</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="string"/>
            <xs:enumeration value="cdata"/>
            <xs:enumeration value="list"/>
            <xs:enumeration value="enum"/>
            <xs:enumeration value="object"/>
            <xs:enumeration value="int"/>
            <xs:enumeration value="boolean"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="columnType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="string"/>
            <xs:enumeration value="enum"/>
            <xs:enumeration value="cdata"/>
            <xs:enumeration value="object"/>
            <xs:enumeration value="int"/>
            <xs:enumeration value="boolean"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="enumType">
        <xs:annotation>
            <xs:documentation>Defines the type for an enum value</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="string"/>
            <xs:enumeration value="int"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="allowed-values">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="allowed-value" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="scopeType">
        <xs:annotation>
            <xs:documentation>Defines the scope for exported parameter</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="global"/>
            <xs:enumeration value="output"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="pathsType">
        <xs:annotation>
            <xs:documentation>Defines the enums for number of output
paths</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="static"/>
            <xs:enumeration value="dynamic"/>
```

```
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="allowed-value">
        <xs:complexType>
            <xs:attribute name="type" type="enumType" use="required"/>
            <xs:attribute name="value" type="xs:string" use="required"/>
            <xs:attribute name="default" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="valueKey" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="bladelet">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="icon-ref" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="palette-icon" minOccurs="0">
                                <xs:complexType>
                                    <xs:attribute name="href" type="xs:string"
use="required"/>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="document-icon" minOccurs="0">
                                <xs:complexType>
                                    <xs:attribute name="href" type="xs:string"
use="required"/>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="exceptions" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="exception" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element ref="bladelet-design" minOccurs="1"/>
                <xs:element ref="bladelet-deployment" minOccurs="1"/>
                <xs:element ref="bladelet-runtime" minOccurs="1"/>
                <xs:element name="output-paths" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="output-path" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="number" type="pathsType" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="versionId" type="xs:unsignedInt" use="required"/>
            <xs:attribute name="displayNameKey" type="xs:string" use="required"/>
            <xs:attribute name="categoryKey" type="xs:string" use="optional"/>
            <xs:attribute name="bladeletClass" type="xs:string" use="required"/>
            <xs:attribute name="terminal" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="allowInResponse" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="allowInException" type="xs:boolean" use="optional"
default="true"/>
```

```
            <xs:attribute name="bundle" type="xs:string" use="required"/>
            <xs:attribute name="validatorClass" type="xs:string" use="required"/>
            <xs:attribute name="validatorRules" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="bladelet-runtime">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="config-params" minOccurs="0"/>
                <xs:element name="exported-params" type="exported-paramsType"
minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="output-path">
        <xs:complexType>
            <xs:attribute name="label" type="xs:string" use="required"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="param-value">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="allowed-values" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="create" type="xs:boolean" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="param">
        <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="designName" type="xs:string" use="optional"/>
            <xs:attribute name="scope" type="scopeType" use="optional"/>
            <xs:attribute name="type" type="xs:string" use="optional" default="string"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="bladelet-deployment">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="system-params" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="param-value" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="exported-paramsType">
        <xs:sequence>
            <xs:element ref="param" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="configuration-group">
        <xs:complexType>
            <xs:choice minOccurs="0">
                <xs:element ref="configuration-subgroup" maxOccurs="unbounded"/>
                <xs:choice maxOccurs="unbounded">
                    <xs:element ref="parameter-group"/>
                    <xs:element ref="parameter"/>
```

```
                </xs:choice>
            </xs:choice>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="type" type="groupType" use="optional"/>
            <xs:attribute name="value" type="xs:string" use="optional"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
            <xs:attribute name="valueKey" type="xs:string" use="optional"/>
            <xs:attribute name="default" type="xs:boolean" use="optional"
default="false"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="parameter-group">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="type" type="groupType" use="optional"/>
            <xs:attribute name="value" type="xs:string" use="optional"/>
            <xs:attribute name="valueKey" type="xs:string" use="optional"/>
            <xs:attribute name="default" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="configuration-subgroup">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element ref="parameter-group"/>
                <xs:element ref="parameter"/>
            </xs:choice>
            <xs:attribute name="name" type="xs:string" use="optional"/>
            <xs:attribute name="type" type="enumType" use="optional"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="parameter">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="allowed-values" minOccurs="0"/>
                <xs:element ref="column-info" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="type" type="xs:string" use="required"/>
            <xs:attribute name="conditionRule" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="domain" type="xs:string" use="optional"/>
            <xs:attribute name="editor" type="xs:string" use="optional"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="designName" type="xs:string" use="optional"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
            <xs:attribute name="cdataType" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="allowUserInput" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="allowVarBinding" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="output" type="xs:boolean" use="optional" default="false"/>
            <xs:attribute name="defaultValue" type="xs:string" use="optional"/>
            <xs:attribute name="tooltipKey" type="xs:string" use="optional"/>
            <xs:attribute name="optional" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="validate" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="mapKeyColumn" type="xs:string" use="optional"/>
```

```
        </xs:complexType>
    </xs:element>
    <xs:element name="config-params">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="mode">
                    <xs:complexType>
                        <xs:attribute name="value" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="maxworkers">
                    <xs:complexType>
                        <xs:attribute name="value" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
                <xs:element name="categories">
                    <xs:complexType>
                        <xs:attribute name="value" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="bladelet-design">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="bladelet-parameters">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="configuration-group" minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="exception">
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="key" type="xs:string" use="required"/>
            <xs:attribute name="desc" type="xs:string" use="optional"/>
            <xs:attribute name="descKey" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="column-info">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="allowed-values" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="editor" type="xs:string" use="optional"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="designName" type="xs:string" use="optional"/>
            <xs:attribute name="key" type="xs:string" use="optional"/>
            <xs:attribute name="cdataType" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="validate" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="optional" type="xs:boolean" use="optional"
default="false"/>
            <xs:attribute name="conditionRule" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="allowUserInput" type="xs:boolean" use="optional"
default="true"/>
```

```
            <xs:attribute name="allowVarBinding" type="xs:boolean" use="optional"
default="true"/>
            <xs:attribute name="defaultValue" type="xs:string" use="optional"/>
            <xs:attribute name="tooltipKey" type="xs:string" use="optional"/>
            <xs:attribute name="type" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

# Adding Exception Recovery to a Custom Bladelet

You can enhance a custom bladelet so that exceptions encountered during PEP execution do not stop processing. This involves defining named (recoverable) exceptions in the bladelet definition file (bladelet-info.xml). After you finish, the ADS shows the enhancement as additional exception "output pins" on the custom bladelet. For more information, see the *AON Development Studio Guide*.

To add this feature to a custom bladelet, follow the steps listed below.

---

**Step 1** Using an editor, open the bladelet-info.xml file.

**Step 2** Add an "exceptions" section to the bladelet definitions in the bladelet-info.xml file.

The following example from the bladelet-info.xml file for the custom bladelet "ComputeAggregates"shows an <exceptions>...</exceptions> section.

```
<bladelet name="ComputeAggregates"
           displayNameKey="ComputeAggregates.name"
           versionId="1"
           bladeletClass="com.cisco.aons.visibility.ComputeAggregatesBladelet"
           categoryKey="general.category.key"
           bundle="com.cisco.aons.visibility.visibility"
           validatorClass="com.cisco.aons.visibility.StatisticsBladeletValidator"
           validatorRules="">
  <icon-ref>
     <palette-icon href="com/cisco/aons/visibility/26i_dataaggregator.png"/>
     <document-icon href="com/cisco/aons/visibility/i_dataaggregator.png"/>
  </icon-ref>

       <exceptions>

                 <exception  id="Missing-Aggregate-Exception"
                               key="exception.missing.aggregate.label"
                               desc="Aggregate not defined"
                               descKey="exception.missing.aggregate.desc"
                     />
                </exceptions>

              <bladelet-design>
                   <bladelet-parameters>
```

**Step 3** Add appropriate display strings for named exceptions to the resource bundle.

1. Using an editor, open the properties file (resource bundle).
2. Add appropriate strings to the file.

   The following sample lines are from the properties file (resource bundle) associated with the custom bladelet "ComputeAggregates." In this case, the ADS will display either "Missing Aggregate" or "Missing Aggregate definition" when an exception is encountered.

```
exception.missing.aggregate.label=Missing Aggregate
exception.missing.aggregate.desc=Missing Aggregate definition
```

**Step 4** Use the NamedExtenstionException in the Java class.

1. Using a Java editor, open the java class for the custom bladelet.
2. Add the NamedExtensionException sections.

**Note** The following example shows the NamedExtensionException section from the java class for the custom bladelet "ComputeAggregates."

```
import com.cisco.aons.exception.NamedExtensionException; // this is in aonscommon.jar
private static final String MISSING_AGGREGATES_EXCEPTION =
"Missing-Aggregate-Exception"; // from "id" attribute in bladelet-info.xml

public void execute() throws AONSException, NamedExtensionException // example of
method signature

        // example of throwing a NamedExtensionException
            if (!AggregateUtilities.doesDBTableExist(statId, getLogger())) {
                if (getLogger().isInfoEnabled()) {
                    getLogger().info("Aggregate not found");

                }
             throw new NamedExtensionException("Aggregate does not exist", true,
MISSING_AGGREGATES_EXCEPTION);
            }
```

For descriptions of AONSException and NamedExtensionException, see Exception Package, page A-1.

# Testing the Custom Bladelet

After creating the custom bladelet, you should test it. Follow the high level steps listed below.

**Step 1** Using the AON Development Studio (ADS), include your new custom bladelet along with several other supplied bladelets (for example, Encrypt) into a PEP.

The automatic validation processes built into the ADS tool will identify any problems in bladelet-to-bladelet PEP.

**Step 2** Run the completed PEP with test data.

If it runs without error messages, your custom bladelet is ready for use.

# Conventional Custom Bladelet API Specification

AON provides the Custom Bladelet API as a single package, com.cisco.AON.sdk.custombladelet. As developers create new bladelets, they use various methods in the API to provide message generators. This section introduces the Custom Bladelet API, focusing on each component interface.

Thee Custom Bladelet API package (com.cisco.AON.sdk.custombladelet) contains the components listed below.

- AbstractCustomBladelet, page 2-32

  This is the basic custom bladelet class. It implements the CustomBladelet interface and must be included in all new custom bladelets.

- CustomBladelet, page 2-36

  This is the basic custom bladelet interface. It must be included in all new custom bladelets. It is used to access PEP variables via methods that get and set context data.

- CustomBladeletContext, page 2-38

  This interface provides context to the custom bladelet. It is mainly used to pass variables, get PEP details, set output path and log messages.

# AbstractCustomBladelet

Extending java.lang.Object, the AbstractCustomBladelet is the basic custom bladelet class. This class implements the CustomBladelet interface and must be included in all new custom bladelets. This class incorporates both inherited and uninherited methods.

Inherited Methods—The AbstractCustomBladelet class inherits the following methods from class java.lang.Object:

- clone
- equals
- finalize
- getClass
- hashCode
- notify
- notifyAll
- toString
- wait

Other Methods—The AbstractCustomBladelet class also includes the additional methods listed and described below.

**Note** The components (methods, fields, and constructors) of this abstract class are not operational.

This section describes the following components of AbstractCustomBladelet:

- Code, page 2-33
- Fields, page 2-34
- Constructor, page 2-34
- Methods, page 2-34

## Code

```
package com.cisco.AON.sdk.custombladelet;

import com.cisco.AON.message.IAONSMessage;
import com.cisco.AON.log.Log;
import com.cisco.AON.exception.AONSException;
import com.cisco.AON.util.DomainReader;

/**
 * Abstract implementation of Custom Bladelet interface.
 * All Custom Bladelets should extend AbstractCustomBladelet.
 * Define no operation methods.
*/
public abstract class AbstractCustomBladelet implements CustomBladelet
{
    protected IAONSMessage msg;
    protected CustomBladeletContext context;
    protected DomainReader domainReader;
    public static Log logger;

    public void onLoad() {
    }

    public void onCreate(IAONSMessage msg) {
        this.msg = msg;
    }
/**
* All the extending classes are expected to implement this method
     */
    public abstract void execute() throws AONSException;
public void onDestroy() {
    }

    public void onUnload() {
    }

    public void onException(Exceptions exception) {
    }

    public Log getLogger() {
        return logger;
    }

    public  void setLogger(Log logger) {
    this.logger = logger;
    }

    public void setContext(CustomBladeletContext context) {
     this.context = context;
}

    public CustomBladeletContext getContext() {
     return context;
    }
public void setDomainReader(DomainReader domainReader) {
    this.domainReader = domainReader;
    }
    public DomainReader getDomainReader() {
    return domainReader;
    }
}
```

## Fields

The fields in the AbstractCustomBladelet class are described below.

| Fields | Description |
|---|---|
| context | protected CustomBladeletContext context<br>Context for the bladelet. |
| domainReader | protected DomainReader domainReader<br>Domain reader for the bladelet. |
| logger | public static Log logger<br>Logger for the bladelet. |
| msg | protected IAONSMessage msg<br>Message. |

## Constructor

The single constructor in the AbstractCustomBladelet class is summarized below.

| Constructor | Description |
|---|---|
| AbstractCustomBladelet | public AbstractCustomBladelet() |

## Methods

The methods of AbstractCustomBladelet are described below.

| Methods | Description |
|---|---|
| execute | public abstract void CustomBladelet execute()<br>throws AONSException<br>Specified by execute in interface.<br>All extending classes must implement this method. It performs the required business logic. |
| getContext | public CustomBladeletContext getContext()<br>Specified by getContext in interface CustomBladelet.<br>Gets the context from the custom bladelet. |
| setContext | public void setContext(CustomBladeletContext context)<br>Parameters:<br>context—CustomBladeletContext<br>Specified by setContext in interface CustomBladelet.<br>Sets the context for the custom bladelet. |
| onCreate | public void onCreate(IAONSMessage msg)<br>Specified by onCreate in interface CustomBladelet.<br>Custom bladelets implement this method to perform creation activities. |

| Methods | Description |
|---|---|
| onDestroy | public void onDestroy()<br>Specified by onDestroy in interface CustomBladelet.<br>Custom bladelets implement this method to reverse onCreate actions. |
| onLoad | public void onLoad()<br>Specified by onLoad in interface CustomBladelet.<br>Custom bladelets implement this method to perform loading activities. |
| onException | public void onException(Exception exception)<br>Specified by onException in interface CustomBladelet.<br>exception --- Exception to be processed.<br>Custom bladelets implement this method to handle exceptions. |
| onUnload | public void onUnload()<br>Specified by onUnload in interface CustomBladelet.<br>Custom bladelets implement this method to perform unloading activities. |
| getContext | public CustomBladeletContext getContext()<br>Specified by getContext in interface CustomBladelet.<br>Returns CustomBladeletContext, getting the context from the custom bladelet. |
| setContext | public void setContext(CustomBladeletContext context)<br>Specified by setContext in interface CustomBladelet.<br>Parameters:<br>context—CustomBladeletContext<br>Sets the custom bladelet context. |
| getDomainReader | public DomainReader getDomainReader()<br>Specified by getDomainReader in interface CustomBladelet.<br>Returns the attribute domainReader from the custom bladelet. |
| setDomainReader | public void setDomainReader(DomainReader domainReader)<br>Specified by setDomainReader in interface CustomBladelet.<br>Parameters:<br>domainReader—DomainReader<br>Sets the attribute domainReader to the custom bladelet. |

| Methods | Description |
|---|---|
| getLogger | public Log getLogger()<br>Specified by getLogger in interface CustomBladelet.<br>Returns the log object to record information and error messages. |
| setLogger | public void setLogger(Log logger)<br>Specified by setLogger in CustomBladelet.<br>Custom bladelets implement this method to set a logger. |

# CustomBladelet

The CustomBladelet interface must be included in all new custom bladelets. Generally, it is used to access PEP variables and get and set context data. The following sections focus on the interface code and methods:


- Code, page 2-36
- Methods, page 2-37


## Code

```
package com.cisco.AON.sdk.custombladelet;

import com.cisco.AON.message.IAONSMessage;
import com.cisco.AON.log.Log;
import com.cisco.AON.exception.AONSException;
import com.cisco.AON.util.DomainReader;

/**
 * CustomBladelet interface
*/
public interface CustomBladelet
{
    /**
     * Custom Bladelet classes can implement this method to do the loading time activities
     */
    public void onLoad();
/**
* Custom Bladelet classes can implement this method to do the creation time activities
     */
    public void onCreate(IAONSMessage ctx);

    /**
* Custom Bladelet classes can implement this method to do the actual business logic
     * @throws AONSException
     */
    public void execute() throws AONSException;

    /**
     * Custom Bladelet classes can implement this method to do the inverse activities of
the onCreate method
     */
    public void onDestroy();

    /**
* Custom Bladelet classes can implement this method to do the unload time activities
     */
```

```
    public void onUnload();

    /**
     * Custom Bladelet classes can implement this method to do the activities when it
encounters exception
     */
    public void onException(Exception exception);

    /**
* Setting the context the Custom Bladelet.
     *@param context CustomBladeletContext
     */
    public void setContext(CustomBladeletContext context);
/**


* Getting the context from the Custom Bladelet
     * @return CustomBladeletContext
     */
    public CustomBladeletContext getContext();

    /**
     * Setting the (attribute) domain reader to the Custom Bladelet
     * @param domainReader DomainReader
     */
    public void setDomainReader(DomainReader domainReader);

    /**
     * Getting the (attribute) domain reader from the Custom Bladelet
     * @return DomainReader
     */
    public DomainReader getDomainReader();
    /**
     * This method returns the log object to record  info, error messages
*/
public Log getLogger();
    /**
     * This method sets the log object
     */
    public void setLogger(Log logger);
}
```

## Methods

The CustomBladelet interface includes the methods summarized below:

| Methods | Description |
|---|---|
| execute | public void execute()<br>throws AONSException<br>Custom bladelet classes implement this method to perform business logic operations. |
| getContext | public CustomBladeletContext getContext()<br>Returns the CustomBaldeletContext, getting context from the custom bladelet. |

| Methods | Description |
|---|---|
| setContext | public void setContext(CustomBladeletContext context)<br>Parameters:<br>context—CustomBladeletContext<br>Sets the custom bladelet context. |
| getDomainReader | public DomainReader getDomainReader()<br>Returns the attribute domain reader from the custom bladelet. |
| setDomainReader | public void setDomainReader (DomainReader domainReader)<br>Parameters:<br>domainReader—DomainReader<br>Sets the (attribute) domain reader to the custom bladelet. |
| getLogger | public Log getLogger()<br>Returns the log object to record information and error messages. |
| setLogger | public void setLogger(Log logger)<br>Sets the log object. |
| onCreate | public void onCreate(IAONSMessage ctx)<br>Custom bladelet classes implement this method to perform creation time activities. |
| onDestroy | public void onDestroy()<br>Custom bladelet classes implement this method to reverse the actions of onCreate. |
| onException | public void onException(Exception exception)<br>Custom Bladelet classes can implement this method to perform operations when they encounters an exception. |
| onLoad | public void onLoad()<br>Custom bladelet classes implement this method to perform loading operations. |
| onUnload | public void onUnload()<br>Custom bladelet classes implement this method to perform unloading operations. |

# CustomBladeletContext

The CustomBladeletContext interface provides the context to the custom bladelet. It is mainly used to pass PEP variables, get PEP details, set output paths, and log messages. For example, you can get the variables later to remove them from the PEP. The following sections focus on the interface code and methods:

- Code, page 2-38
- Methods, page 2-40

## Code

```
package com.cisco.AON.sdk.custombladelet;
```

```
import java.util.Map;
import com.cisco.AON.exception.AONSException;
import com.cisco.AON.exception.ExtServiceException;
import com.cisco.AON.pep.PEPData;

/**

* This interface provides the context to the custom bladelet
 * Mainly used to pass around variables, get PEP details,
 * setting output path and logging messages etc
*/
public interface CustomBladeletContext
{
    /**
     * This method is used to set output parameters of custom bladelet at the global
context
* @param name output parameter as String
     * @param value output parameter Object
     * @throws AONSException
     */
    public void setGlobalContextData(String name,Object value) throws AONSException;

    /**
     * This method is used to retrieve data associated with an input parameter (global
scope)
* of the custom bladelet
     * @param name input parameter as String
     * @return Object output object
     */
    public Object getGlobalContextData(String name);

    /**

* This method is used to set output parameters of custom bladelet
     * @param name output parameter as String
     * @param Object output parameter object
     * @throws AONSException
     */
    public void setContextData(String name,Object value) throws AONSException;

    /**
     * This method is used to retreive data associated with an input
     * parameter of custom bladelet
     * @param name input parameter as String
     * @return Object output object
     */
    public Object getContextData(String name);
/**
     * This method returns the PEPData object from the context
     * @return PEPData
     */
    public PEPData getPEPData();
    /**
     * This method sets the output path (to determine the next step) for the bladelet
     * @param pOutputLabel lable to be set
     */
    public void setOutputPath(String pOutputLabel);
/**
     * Returns the bladelet name
     * @return bladelet name
     */
    public String getBladeletName();
    /**
     *
```

```
     * @param bSync true ==> Log records will be written to data source synchronously,
     *              false ==> Log records will be written to data source asynchronously
     * @param messageLogPolicy Name of message log policy to use for logging.
     *                         Message Log policy is configured for each node on AMC.
     * @param logLevel Takes one of the following values
      *
       *    {@link MessageLog.LOG_BASIC} - Only message attributes, no message contents
      *    {@link MessageLog.LOG_HEADER} - Message Headers only. For example, SOAP header.
      *    {@link MessageLog.LOG_BODY} - Message Body only. For example, SOAP body only.
      *    {@link MessageLog.LOG_WHOLE_MESSAGE}
       *   {@link MessageLog.LOG_BY_EXPRESSION} - Extract message contents as specified by
the XPaths in expressions.
       *
       *     <b>Note</b> For non SOAP messages, Message Header is often empty
     *      and Message Body will be the Whole message.
 * @param customId  Arbitary string to identify this log entry. e.g. key to other
database records.
     *                  This is optional, can be null.
     * @param expressions  A map of (String,String) pairs. The key is the name of the
expression.
     *                     The value is the XPath expression. For log level != 5, this
param will be ignored.


     *                     This is optional, can be null.
     * @param variables    A map of (String, Object) pairs. The key is the name of the
variable mapping.
     *                     The value is the PEP variable object. The object must either
have a meaningful
     *                     toString() implementation or implements the Loggable interface.
     *                     This is optional, can be null.
     * @throws ExtServiceException
     */
public void logMessage(boolean bSync,String messageLogPolicy,short logLevel,String
customId, Map expressions, Map variables)
       throws ExtServiceException;
}
```

## Methods

CustomBladeletContext includes the methods summarized below.

| Methods | Description |
|---|---|
| getBladeletName | public java.lang.String getBladeletName()<br>Returns the bladelet name. |
| getContextData | public java.lang.Object getContextData(java.lang.String name)<br>Parameters:<br>name—input parameter as String<br>Returns the output object containing data associated with an input parameter of custom bladelet. |

| Methods | Description |
|---|---|
| setContextData | public void setContextData(java.lang.String name,<br>java.lang.Object value)<br>throws AONSException<br>Parameters:<br>name—output parameter as String<br>Sets output parameters of custom bladelet. |
| getPEPData | public PEPData getPEPData()<br>Returns the PEPData object from the context.<br>For more information, see the PEPData description in Appendix A. AONSCommon Specification. |
| getGlobalContextData | public java.lang.Object getGlobalContextData(java.lang.String name)<br>Parameters:<br>name—input parameter String<br>Returns the output object containing data associated with an input parameter (global scope) of the custom bladelet. |
| setGlobalContextData | setGlobalContextData<br>public void setGlobalContextData(java.lang.String name,<br>java.lang.Object value)<br>throws AONSException<br>Parameters:<br>name—output parameter as String<br>value—output parameter Object<br>Sets output parameters of custom bladelet at the global context |

| Methods | Description |
|---|---|
| logMessage | logMessage<br>public void logMessage(boolean bSync,<br>java.lang.String messageLogPolicy,<br>short logLevel,<br>java.lang.String customId,<br>java.util.Map expressions,<br>java.util.Map variables)<br>throws ExtServiceException<br>Parameters:<br>bSync (true)—Log records will be written to data source synchronously.<br>(false)—Log records will be written to data source asynchronously.<br>messageLogPolicy—Name of message log policy to use for logging. Message Log plogLevel<br>logLevel—Takes one of the following values MessageLog.LOG_BASIC - Only message attributes, no message contents MessageLog.LOG_HEADER—Message Headers only. For example, SOAP header. MessageLog.LOG_BODY—Message Body only. For example, SOAP body only. MessageLog.LOG_WHOLE_MESSAGE MessageLog.LOG_BY_EXPRESSION—Extract message contents as specified by the XPaths in expressions. Note For non SOAP messages, Message Header is often empty and Message Body will be the Whole message.<br>customId—Arbitrary string to identify this log entry. e.g. key to other<br>expressions—A map of (String,String) pairs. The key is the name of the expression. The value is the XPath expression. For log level != 5, this param will be ignored. This is optional, can be null.<br>variables—A map of (String, Object) pairs. The key is the name of the variable mapping. The value is the PEP variable object. The object must either have a meaningful toString() implementation or implements the Loggable interface. This is optional, can be null.<br>Writes log records to a data source. |
| setOutputPath | setOutputPath<br>public void setOutputPath(java.lang.String pOutputLabel)<br>Parameters:<br>pOutputLabel—label to be set<br>Sets the output path (to determine the next step) for the bladelet. |

# Optimized Custom Bladelets—Fastpath

AON provides both a built-in infrastructure and customization tools for optimized message processing services. You can incorporate the built-in optimized bladelets (Send, Validate, and SetDestination) into optimized PEPs. And, you can use the optimized custom bladelet development SDK to write your own optimized custom bladelets for incorporation into optimized PEPs. When an entire PEP can be optimized, it is directed to the Fastpath service for processing.

When an entire PEP can be optimized, the Execution Control module calls a dispatcher (that uses bladelet attributes) to invoke the Fastpath service and pass it the message and context. An incoming message is examined for possible Fastpath processing.

- If the entire PEP can be optimized, it is passed on for Fastpath processing.
- If not, the massage is passed on for conventional processing.

The following sections describe the Fastpath bladelets SDK:

- Custom Bladelet Development SDK for Fastpath Processing, page 2-43
- Developing a Custom Bladelet for Fastpath Processing, page 4-1

For more information, see Chapter 4, “AON Optimization - Fastpath”.

## Custom Bladelet Development SDK for Fastpath Processing

The AON software development kit (SDK) provides all files and support software necessary to create new Fastpath custom bladelets. When the compressed SDK file (.tar) is unzipped, it creates all necessary folders, header files, and so on. These include a folder (include/bdksample) that contains a sample Fastpath Log bladelets. For more information, see:

- Custom Bladelet for Fastpath Processing, page 4-1
- Fastpath Custom Bladelet SDK Files, page 4-3
- Sample Fastpath Custom Bladelet, page 4-4

For additional background information, see the *AON Development Studio Guide*.

# Policy Execution Plans

Supplied and custom bladelets are incorporated into Policy Execution Plans (PEPs). Using the ADS, you create new PEPs in the PEP Developer area. This tool enables PEP authors to drag and drop icons representing bladelets into the PEP Developer canvas and appropriately connected to create a PEP.

During this process, the PEP author configures bladelets using dialogue boxes that reflect the syntax and property types exposed (by the bladelet developer) for the bladelet. Thus, bladelet designers decide which properties are to be exposed and how the GUI interface will display those properties in the ADS. For additional information, see the AON Bladelet Reference in the *AON Development Studio Guide*.

You may want to develop a Fastpath custom bladelet and PEP. The are designed to move messages through AON at a faster rate than ordinary messages by removing unnecessary processing overhead. For more information see:

- Creating and Deploying a PEP, page 2-44
- PEP Descriptive Language, page 2-44

## Creating and Deploying a PEP

You use ADS windows to create policy execution plan (PEP) and one or more *Message Type*s that determine how messages are processed by the new PEP. In this process, you:

1. Select the AON node where the PEP will execute on the ADS main screen.
2. Select bladelets from the ADS Task Pane and connect them in the ADS Developer for a particular task.
3. Configure each bladelet through its Properties screen.
4. Create and associate one or more message types for this PEP.
5. Package the PEP.
6. Deploy the PEP.

   You synchronize to AMC and then, use AMC windows to deploy the PEP to AON.

For detailed instructions, see the *AON Development Studio Guide*.

## PEP Descriptive Language

When ADS users create PEPs on ADS windows, they are actually using a transparent procedural programming language, PEP Descriptive Language (PDL). PDL is a business logic that can be executed on an AON node.

PDL supports various language constructs including looping and block instruction execution (if/else, while, and so on) for loops in Java. PDL uses the Loop construct for repetitive processes on a List, Map, or Iterator. PDL uses the Scope construct (a no-op block) to restrict the visibility of variables similar to the {} block in Java. PDL uses the another construct, Break (similar to break statements in Java and C) to exit a loop prematurely based on specified conditions.

For more information, see the descriptions of the Loop and Scope in the AONS Bladelet Reference of the *AON Development Studio Guide*.

# Samples

The first set of samples in this section indicate how the bladelet-info.xml file is used, the second shows the code for two sample custom bladelets.

- Sample Bladelet Information XML File, page 2-44
- Custom Bladelet Samples, page 2-47

For more information, refer back to Designing Custom Bladelets, page 2-2.

## Sample Bladelet Information XML File

The samples shown below is a typical bladelet-info.xml file.

- Verify Bladelet Info XML, page 2-45
- Sample CustomLog Bladelet-Info File, page 2-46

For more information, see For detailed steps, see Developing a Conventional Custom Bladelet, page 2-9., page 2-5.

## Verify Bladelet Info XML

The following bladelet-info.xml file defines the Verify bladelet.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
-->
- <bladelet-info>
- <bladelet name="XMLVerifier:1" displayNameKey="verify.bladelet.name" versionId="1"
categoryKey="security.category.key" bundle="com.cisco.AON.bladelet.v1.verifyBladelet"
bladeletClass="com.cisco.AON.bladelet.v1.DSigVerifierBladelet"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=""
validatorClass="com.cisco.AON.bladelet.v1.VerificationBladeletValidator"
validatorRules="com.cisco.AON.bladelet.v1.VerifyValidatorRules">
- <icon-ref>
<palette-icon href="com/cisco/aons/bladelet/v1/26i_verifysignature.png" />
<document-icon href="com/cisco/aons/bladelet/v1/i_verifysignature.png" />
</icon-ref>
- <exceptions>
<exception id="NoDataFoundToVerify" key="exception.signaturenotfoundexception.label"
desc="No Signature Found in Message" descKey="exception.signaturenotfoundexception.desc"
/>
</exceptions>
- <bladelet-design>
- <bladelet-parameters>
- <!--
 Message verifying configuration group
-->
- <configuration-group name="ContentToVerify" key="cg.contenttoverify">
- <configuration-subgroup>
- <parameter-group name="SourceContent" key="cg.contenttoverify.pg.sourcecontent">
- <!--
 Content that carries the signature
-->
<parameter name="Input" designName="ContentToVerify.SourceContent.Input" type="Content"
key="cg.contenttoverify.pg.sourcecontent.p.input" allowVarBinding="true"
allowUserInput="false" />
</parameter-group>
</configuration-subgroup>
</configuration-group>
- <!--
 The following are miscellaneous configurations to be used mostly as defaults.
-->
- <configuration-group name="Advanced" key="cg.advanced">
- <configuration-subgroup>
- <parameter-group name="Trust Verification"
key="cg.contenttoverify.pg.trustverification">
- <!--
 XKMS Trust Verification Mode - Disabled, Pilot mode or Production mode
-->
- <parameter name="XKMSTrustVerification"
designName="Advanced.TrustVerification.XKMSTrustVerification"
key="cg.advanced.pg.trustverification.p.xkms" type="string" allowVarBinding="false"
allowUserInput="false" editor="combo-box">
- <allowed-values>
<allowed-value type="string" value="Disable"
valueKey="cg.advanced.pg.trustverification.p.xkmstrust.disable" default="true" />
<allowed-value type="string" value="Pilot"
valueKey="cg.advanced.pg.trustverification.p.xkmstrust.pilot" />
<allowed-value type="string" value="Production"
valueKey="cg.advanced.pg.trustverification.p.xkmstrust.production" />
</allowed-values>
</parameter>
```

```
- <!--
 Local Certificate Trust Verification - Disabled or Enabled
-->
- <!--
 Whether the certificate that came with signature is explicitly trusted in the local trust
store
-->
- <parameter name="LocalCertTrustVerification"
designName="Advanced.TrustVerification.LocalCertTrustVerification"
key="cg.advanced.pg.trustverification.p.localcerttrust" type="string"
allowVarBinding="false" allowUserInput="false" editor="combo-box">
- <allowed-values>
<allowed-value type="string" value="Disable"
valueKey="cg.advanced.pg.trustverification.p.localcerttrust.disable" default="true" />
<allowed-value type="string" value="Enable"
valueKey="cg.advanced.pg.trustverification.p.localcerttrust.enable" />
</allowed-values>
</parameter>
- <!--
 Certificate Chain Trust Verification - Disabled or Enabled
-->
- <!--
 Whether the root of the certificate chain that came with signature is explicitly trusted
in the local CA trust store
-->
- <parameter name="CertChainTrustVerification"
designName="Advanced.TrustVerification.CertChainTrustVerification"
key="cg.advanced.pg.trustverification.p.certchain" type="string" allowVarBinding="false"
allowUserInput="false" editor="combo-box">
- <allowed-values>
<allowed-value type="string" value="Disable"
valueKey="cg.advanced.pg.trustverification.p.certchaintrust.disable" default="true" />
<allowed-value type="string" value="Enable"
valueKey="cg.advanced.pg.trustverification.p.certchaintrust.enable" />
</allowed-values>
</parameter>
</parameter-group>
</configuration-subgroup>
</configuration-group>
</bladelet-parameters>
</bladelet-design>
- <bladelet-deployment>
<system-params />
</bladelet-deployment>
- <bladelet-runtime>
- <exported-params>
<param name="Output" type="Content" designName="Output" key="verify.verifiedContent" />
</exported-params>
</bladelet-runtime>
- <output-paths number="static">
<output-path label="success" key="success.label" />
<output-path label="fail" key="fail.label" />
</output-paths>
</bladelet>
</bladelet-info>
```

For more information, see Developing a Conventional Custom Bladelet, page 2-9.

## Sample CustomLog Bladelet-Info File

The supplied sample CustomLog bladelet-info.xml file, shown below, has fastpathEnabled set to true.

```
<bladelet-info>
    <bladelet name="CustomLog:1"
              displayNameKey="CustomLog.name"
              versionId="1"
              bladeletClass="com.cisco.aons.custom.CustomLogBladelet"
              categoryKey="general.category.key"
              bundle="customlog"
              fastpathEnabled="true"
              nativeId="customlog"
              nativeLib="customlog"
              validatorClass=""
              validatorRules="">
    <bladelet-design>
           <bladelet-parameters>
                <configuration-group name="CustomTextGroup" key="cg.customtext">
                     <configuration-subgroup>
                              <parameter name="CustomText"
designName="CustomText-DesignName" cdataType="true"
                              key="param.customtext"
                              type="string"  optional="false" editor="textarea"
                              tooltipKey="param.tooltip.customtext"/>
                    </configuration-subgroup>
                </configuration-group>
            </bladelet-parameters>
        </bladelet-design>
        <bladelet-deployment>
            <system-params/>
        </bladelet-deployment>
        <bladelet-runtime>
           <exported-params>
          </exported-params>
        </bladelet-runtime>
    </bladelet>
</bladelet-info>
```

# Custom Bladelet Samples

Custom bladelets samples are presented in the following sections:

- RnetBladelet1, page 2-47
- EmailBladelet, page 2-49

Each sample conforms to the standard described in the Using the AON Bladelet Schema, page 2-17. For more about custom bladelet usage, see the *AON Development Studio Guide*.

## RnetBladelet1

Extending AbstractCustomBladelet, RnetBladelet1 (shown below) is used to process a purchase order request.

```
package com.cisco.purchase;

import java.util.Properties;

import org.w3c.dom.*;
import org.apache.xpath.XPathAPI;
import javax.xml.transform.TransformerException;

import com.cisco.AON.message.IAONSMessage;
import com.cisco.AON.message.IContent;
```

```
import com.cisco.AON.message.IXMLContent;
import com.cisco.AON.message.IMessageContext;
import com.cisco.AON.message.MessageParseException;
import com.cisco.AON.exception.AONSException;
import com.cisco.AON.sdk.custombladelet.AbstractCustomBladelet;
import com.cisco.AON.sdk.custombladelet.CustomBladeletContext;

public class RnetBladelet1 extends AbstractCustomBladelet
{
    public RnetBladelet1() {
    }

    public void execute() throws AONSException {
      IContent content = msg.getContent();
        IMessageContext ctx = msg.getMessageContext();
      CustomBladeletContext context = null;
      Document doc = null;
      String EMAIL = "email";
      String SUBJECT = "subject";
      String TEXTMESSAGE = "textmessage";
        String SENDEMAIL = "sendemail";
      String
EMAILPATH="Pip3A4PurchaseOrderRequest/fromRole/PartnerRoleDescription/ContactInformation/E
mailAddress/text()";
      String TOTALAMOUNTPATH =
"Pip3A4PurchaseOrderRequest/PurchaseOrder/totalAmount/FinancialAmount/MonetaryAmount/text(
)";
        getLogger().debug("RnetBladelet1 Bladelet");
      if(content instanceof IXMLContent) {
        IXMLContent xmlContent = (IXMLContent) content;
        getLogger().debug("xml content recognized");
        try {
            doc = xmlContent.getAsDocument();
        }
        catch(MessageParseException exception) {
            throw new AONSException(exception);
        }
        try {
            Node amountNode = XPathAPI.selectSingleNode(doc,TOTALAMOUNTPATH);
            if(amountNode != null) {
                String amount = amountNode.getNodeValue();
                getLogger().debug("Amount is " + amount);
                String email = null;
                double damount = Double.valueOf(amount).doubleValue();
                Node emailNode = XPathAPI.selectSingleNode(doc,EMAILPATH);
                email = emailNode.getNodeValue();
                getLogger().debug("Email is " + email);
                context = this.getContext();
                context.setGlobalContextData(EMAIL,email);
                String subject = "RosettaNet Purchase Order";
                context.setGlobalContextData(SUBJECT,subject);
                String message = "You entered a purchase order. Order amount is " + amount
+". ";
                String lamount = (String)context.getContextData("limitamount");
                message = message + "Entered limit amount to send email is " + lamount;
                double ldamount = Double.valueOf(lamount).doubleValue();
                if(damount >= ldamount) {
                    context.setGlobalContextData(SENDEMAIL,"true");
                }
                else {
                    context.setGlobalContextData(SENDEMAIL,"false");
                }
                context.setGlobalContextData(TEXTMESSAGE,message);
                  context.setContextData("passamount",lamount);
```

```
            } // amount not null
          } //try
          catch(TransformerException exception) {
              throw new AONSException(exception);
          }
          catch(NumberFormatException exception) {
              throw new AONSException(exception);
          }
        } // IXMLContent instance check
      }
}
```

## EmailBladelet

Extending AbstractCustomBladelet, EmailBladelet (shown below) is used to handle incoming email.

```
package com.cisco.purchase;

import java.util.Properties;
import java.util.List;
import java.util.Iterator;

import org.w3c.dom.*;

import javax.mail.Session;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.InternetAddress;
import javax.mail.Address;
import javax.mail.Transport;
import javax.mail.Message;
import javax.mail.internet.AddressException;
import javax.mail.MessagingException;

import com.cisco.AON.message.IAONSMessage;
import com.cisco.AON.message.IContent;
import com.cisco.AON.message.IXMLContent;
import com.cisco.AON.message.IMessageContext;
import com.cisco.AON.message.MessageParseException;
import com.cisco.AON.exception.AONSException;
import com.cisco.AON.util.DomainReader;
import com.cisco.AON.util.DomainException;
import com.cisco.AON.sdk.custombladelet.AbstractCustomBladelet;
import com.cisco.AON.sdk.custombladelet.CustomBladeletContext;

public class EmailBladelet extends AbstractCustomBladelet
{
    public EmailBladelet() {
    }

    public void execute() throws AONSException {
      IContent content = msg.getContent();
        IMessageContext ctx = msg.getMessageContext();
      Document doc = null;
      CustomBladeletContext context = null;
      String EMAIL = "email";
      String SUBJECT = "subject";
      String TEXTMESSAGE = "textmessage";
      String SENDEMAIL = "sendemail";
        getLogger().debug("EmailBladelet Begin");
      context = this.getContext();
      String toemail = (String)context.getGlobalContextData(EMAIL);
      String subject = (String)context.getGlobalContextData(SUBJECT);
      String textmessage = (String)context.getGlobalContextData(TEXTMESSAGE);
```

```
String passamount = (String)context.getContextData("passamount");
getLogger().debug("Testing context data passing thru variable " + passamount);
String sendemail = (String)context.getGlobalContextData(SENDEMAIL);
if(sendemail.equals("true")) {
  String host = null;
  String fromemail = null;
  DomainReader domainReader = super.getDomainReader();
  try {
      domainReader.setDomain("email");
      List valueList = domainReader.getValues("default","host");
      /**
   for(Iterator valueIter=valueList.iterator();valueIter.hasNext();) {
          String value = (String)valueIter.next();
          getLogger().debug("Host value from PM is "+value);
      }
      getLogger().debug("Host value from PM is " + valueList.get(0));
      */
      host = (String)valueList.get(0);
      valueList = domainReader.getValues("default","fromemail");
      fromemail = (String)valueList.get(0);
  }
  catch(DomainException domainException) {
      getLogger().error("Error in Setting the domain/propertyset");
      throw new AONSException(domainException);
  }
  Properties props = System.getProperties();
  props.put("mail.smtp.host",host);
  try {
      Session session = Session.getDefaultInstance(props,null);
      //session.setDebug(true);
      MimeMessage message = new MimeMessage(session);
      message.setFrom(new InternetAddress(fromemail));
      message.addRecipient(Message.RecipientType.TO,new InternetAddress(toemail));
      message.setSubject(subject);
      message.setText(textmessage);
      Transport.send(message);
  }
  catch(AddressException exception) {
      throw new AONSException(exception);
```

CHAPTER 3

# Custom Adapters

AON can process network traffic that includes a variety of industry standard and custom protocols. While the built-in AON adapters may address all of your operating needs, customization is also available with the AON Custom Adapter Software Development Kit (SDK).

This chapter provides an overview of the adpters available for customization overview and explains how to use the Custom Adpater SDK.



See Chapter 4, “AON Optimization - Fastpath” for additional information on extending custom adapters. See the “AON Sandbox” section in Chapter 2, “Custom Bladelets” for a general description of the sandbox service.

## Custom Adapter Overview

Using AON, you can meet the requirements of the IBM Websphere MQ service by developing an MQ adapter that will run on top of the AON runtime as a standalone adapter.

Message Delivery Semantics (MDS) support enhancements to the Custom Adapter API enable AON to guarantee reliable and/or ordered message delivery based on user defined message types.

Adapters, like bladelets, are configured to execute on a specific AON node.

When you design an adpater, you can specify the configuration parameters that will be available to network administrators, typically through the AMC. The configuration parameters are read in by the adapter at runtime. It is this read in process that enforces the policy determining how the adapter connects to an external message queue.

An adapter serves as the entry point and exit point for application messages. Adapters are logically implemented in pairs, as inbound and outbound adapters.

The inbound adapter performs the following tasks:

- protocol termination.
- encapsulation of the application message into an AON-specific protocol header for processing by the AON system.

The outgoing adapter performs the following tasks:

- removal of the AON-specific protocol header from the message.
- re-encapsulation of the message into the outbound protocol.

This section provides background information about custom adapters and related AON processes focusing on:



# AON Adapter Requirements and options

AON imposes the requirements listed below on adapters.

Mandatory—AON adapters must:

- Support the following interaction models
  - One-way
  - Request-response
- Expose content management APIs.
- Provide a mechanism to read/write Attribute Domains.
- Handle exceptions:
  - Must follow AON Policy Execution Plans (PEP) exception management specification
  - Must provide error login interfaces
- Support sandbox class loading of custom adapter
- Provide utility classes for read/write operations on AON buffers

Optional—an adapter may also:

- Provide extensions to customize message classification in an adapter specific way
- Support custom content types
- May provide extensions to customize message classification in an *Adapter* specific way

- May support custom content types

# Custom Adapter Interactions

The AON Custom Adapter SDK supports two interaction models:

- One-way message

  In this case, the message sender expects no response. The Custom Adapter Framework immediately sends an acknowledge (ACK) message to the message sender before forwarding the processed message.

- Request-response message

  In this case, the message sender expects a response message from the recipient of the request.

# Developing an Embedded or Standalone Custom Adapter

Depending on your requirements, you may develop an embedded or standalone adapter. These models are described in the following sections:

- Embedded Adapter, page 3-3
- Standalone Adapter, page 3-3

## Embedded Adapter

When you want the adapter to use all AON resources, you should consider developing an embedded adapter. This type of adapter is tightly integrated with the AON framework, which controls the message input and output through network connections, data buffers, and thread management. An embedded adapter is well-suited to handle stream-oriented protocols (HTTP, SMTP, FTP, etc), because it functions by opening a socket and reading the stream of data.

The Custom Adapter SDK provides a unique set of APIs for writing the embedded adapter. The embedded adapter must maintain its internal state. The framework drives the adapter's state transitions through a series of callbacks to the adapter's receive and send handlers.

You use a callback programming model to develop an embedded adapter. Here, callbacks are used to drive the adapter's state transitions . Because the embedded adapter is tightly interwoven within the AON framework, callbacks must be implemented (more than in stand-alone adapter development) for managing network connections and AON container resources, e.g., data buffers and thread pools.

For more information, see Developing an Embedded Adapter, page 3-6.

## Standalone Adapter

When you want to use the adapter decoupled from the rest of AON, you should develop a standalone adapter. A standalone adapter uses few or none of the resources provided by the adapter framework. The framework makes no assumptions about reading the message in or writing it out. When the standalone adapter has received enough information to transform the incoming message into an AON message, it dispatches the AON message. A standalone adapter is well-suited to handle integration with data sources such as databases, message-oriented middleware, and native libraries.

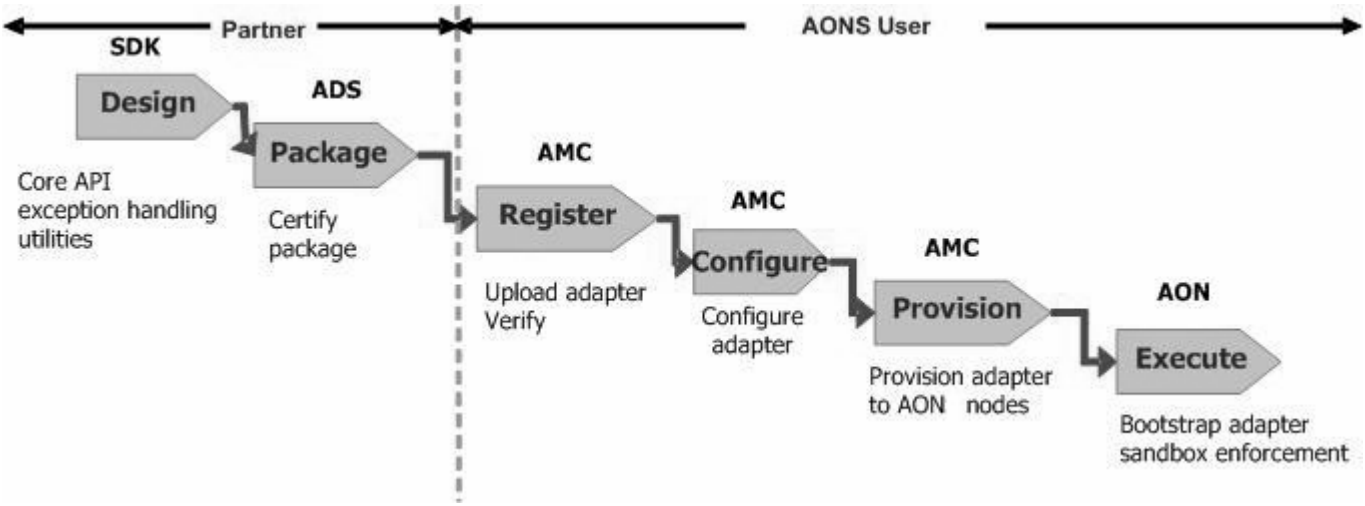
The model for developing a standalone adapter reflects the two types of message dispatch currently permitted: direct, and callback. Direct message dispatch requires the standalone adapter to send the message to the AON Adapter framework. Callback message dispatch requires the standalone adapter to notify the adapter framework of a message arrival and implement the callback that enables the framework to get the message.

For more information, see Developing a Standalone Adapter, page 3-20.

# Custom Adapter Life Cycle

Figure 3-1 shows the life cycle of the custom adapter from design to execution.

***Figure 3-1 Custom Adapter Life Cycle***




As this figure indicates, the adpater developer and the adapter user perfomkr different tasks realted to custom adpters, as follows:

#### Custom Adapter Developer

- Develop—The developer writes the adapter using the Custom Adapter SDK supplied with AON.

  Package—The developer uses the AON Development Studio (ADS) to package the adapter. The developer selects the files to be packaged such as adapter classes, descriptor, configuration templates, and third party libraries. For a list of the components included in an adapter package, see Adapter Package Content, page 3-37.

  ADS validates the package contents and creates Java archive files (.jar). Then, it attaches a vendor certificate. Next, the developer makes it available for downloading.

#### Adapter User

- Register—A user uploads the adapter package to the AMC. The AMC verifies the package contents.
- Configure—Using the AMC, the user configures the adapter. In the case of an embedded adapter, additional configuration may be necessary. Standalone adapters are configured via custom policies.
- Provision—Using the AMC, the user provisions the new adapter to AON nodes.
- Execute—AON bootstraps (loads and initializes) the adapter, reading in its configuration and providing a sandbox environment for its execution.

**Note** To protect the system, a sandbox environment denies the included applications (in this case, the new adapter) access to operating system calls or other resources.

## Registering and Activating the Custom Adapter

After you develop a custom embedded or standalone adapter, you use the AON Management Console (AMC) to register and activate it. For directions, see the *AON Installation and Administration Guid*e. For background information, see Custom Adapter Life Cycle, page 3-4.

## Using the AdapterListenerDomain

After registering your custom embedded or standalone adapter, use the AMC to specify the communication parameters (a global policy) for the new adapter. For details, see the *AON Installation and Administration Guid*e. For background information, see Custom Adapter Life Cycle, page 3-4.

# Custom Adapter SDK

The Custom Adapter Software Development Kit (SDK) currently supports adapter development in Java. Briefly, the ADK includes the following components:

- Adaptersdk.jar—Java archive file containing:
  - Class Files—Contains all SDK class files including AbstractOutboxHandler, Adapter, AdapterEvent, IAdatperConstants, IAdapterContext, IGroup, Source, and so on.
  - Schemas—XSD file defining the ExtenstionClass, ExtensionName, ExtensionPolicy, AdapterName, AttributreDomans, AdapterClass, AdapterClassInfo, and so on
  - Manifest—Simple manifest file.
- JavaDocs—Java docs describing all adapter SDK classes and interfaces.

For file descriptions, see Custom Adapter API Specification, page 3-76.

## Setting Up the Custom Adapter SDK

The Adapter SDK does not require a complex installation process. However, you must include the Java archive file in the development environment. Follow the steps listed below.

**Step 1** Install the SDK package in a separate directory.

**Step 2** Add the SDK Java archive files (.jar) into the classpath.

- adapter.jar
- aonscommon.jar

# Developing an Embedded Adapter

This section gives the general requirements for an embedded adapter, identifies the minimum code components, and explains how to create a new embedded adapter.



## Custom Adapter Names and Versions

The custom embedded adapter must be uniquely identified by the following two types of names:

- Registered name—The custom adapter must have a fully qualified class name. The registered name is used to uniquely identify the adapter within AON.
- Display name—The custom adapter should have a shorter, user-friendly name for display.

The custom adapter must have a version number following the format "[major].[patch number]." The custom adapter developer assigns the major number. The ADS and AMC assign the patch number during packaging and deployment.

## Minimum Required Adapter Components

You write the custom embedded adapter using a Java editor. As a minimum, your code must implement the following base classes:

- EmbeddedAdapter—Extension of the main adapter class from EmbeddedAdapter (package com.cisco.aons.adapter).
- MessageReceiveHandler—Extension of the message reader class from MessageReceiveHandler (package com.cisco.aons.adapter). The class receives the message, creates a message object, and dispatches the messages. See MessageReceiveHandler, page 3-6.
- MessageSendHandler—Extension of the message send class from MessageSendHandler (package com.cisco.aons.adapter). This class converts a message object back to the output message format, and sends the message to the other endpoint. See MessageSendHandler, page 3-10.

In addition, your custom adapter code may implement these classes:

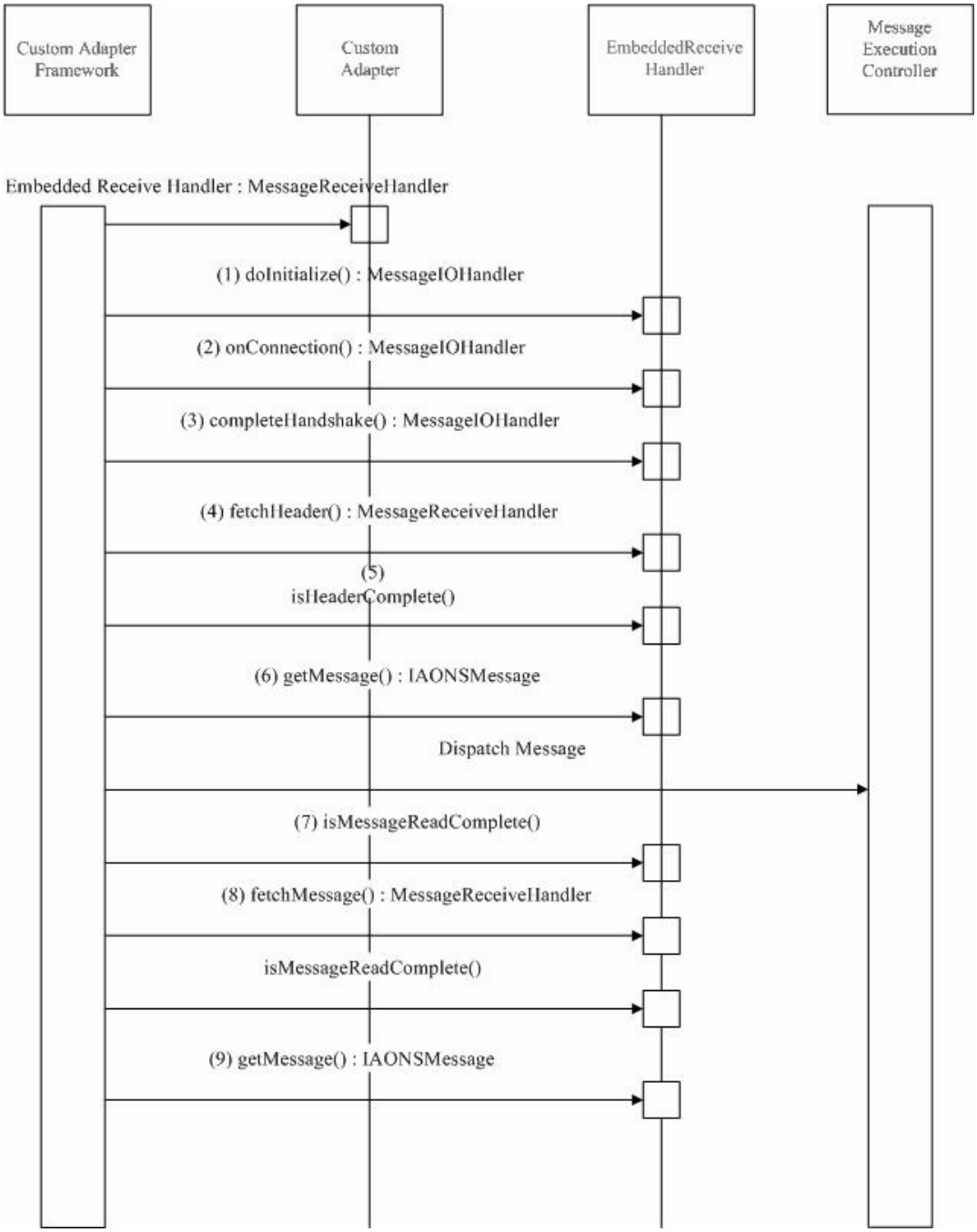
- IContentDecoder—Required for message decoding.
- IContentEncoder—Required for message encoding.

For detailed lists of the interface packages that are included in the custom adapter, see Custom Adapter API Specification, page 3-76.

### MessageReceiveHandler

The MessageReceiveHandler class must be executed the order indicated in Figure 3-2 below.

*Figure 3-2 Embedded Adapter Receive Handler*




This indicates that the Embedded Receive Handler code executes in the following sequence:

1. doInitialize()

   The adapter code must include the doInitialize method for initialization. This method is used for initializations such as creating a data reader.

**Note** An instance of the receive handler is created each time a new message arrives, not once per TCP connection.

Optionally, registerForAttachment()

The adapter code may have to preserve message information for the life of the message—for example, from dispatch to AON to response completion.

If this is the case, at this point in the adapter code, you must include the registerForAttachment method of the IAdapterManager class. The adapter code registers for the type of attachment that it may have to set at runtime.

An adapter registers this attachment during initialization. The adapter caches the index returned by this call and uses it at a later time, during message processing, to set and retrieve objects.

The registered index is constant for that AON runtime instance and is not valid across restarts.

2. onConnection()

   The adapter code calls the onConnection method when a new connection is established. It is called once during each TCP connection

   The following adapter code sample makes this callback.

```
    protected int onConnection(IConnectionContext pContext)
    {
/** Adapter may perform validation or some processing on the established connection,
Once the processing is done, the method must return STATUS_OK **/

/** If no validation or processing is required, the adapter must return STATUS_OK **/

/** Adapter may return appropriate event, such as EVENT_CLOSE to stop from proceeding
to the next state **/

      return IAdapterConstants.STATUS_OK;
    }:
```

3. completeHandshake()

   When the connection is established, if the protocol requires it, the adapter must complete the handshake using the callback completeHandshake. This method is called when a new connection is established. It is called once for each TCP connection. The adapter may perform some processing to complete the handshake with the endpoint.

   The following adapter code sample makes this callback:

```
public int completeHandshake(int pEvent, IConnectionContext
            pContext)
    {
    /** If the adapter requires any handshaking with the client after a
       connection establishment, this is done using the callback
       completeHandshake. If no handshaking is required, the adapter
       returns a STATUS_OK, to transition to the next state. **/

/** Once the handshaking is complete, Adapter returns STATUS_OK to transition to the
next state. **/

/** If errors are found during handshaking, the adapter may return appropriate event,
such as EVENT_CLOSE, etc. to stop the processing.**/

    return IAdapterConstants.STATUS_OK;
    }
```

4. fetchHeader()

Each time a message arrives on a connection, the adapter tries to fetch the headers, using the callback fetchHeaders. The adapter makes this callback repeatedly until all the message headers are completely read in.

The minimum buffer size for a read operation is 512 bytes. If the header is less than 512 bytes, the entire header is read in one callback. If the header exceeds 512 bytes, multiple callbacks to this method are needed until all the headers have been completely read in.

The following sample shows this callback:

```
    public int fetchHeader(int pEvent, IConnectionContext pContext)
    {
/** In this callback, the adapter parses all the headers. The callback to this method
is repeatedly made, until all the headers are read-in.
**/
/** If all headers are successfully read-in, the adapter returns the status EVENT_READ
**/
/** If there is an error in reading the headers, appropriate event such as
EVENT_OUT_OF_BUFFER, will be returned **/
      pEvent = IAdapterConstants.EVENT_READ;
    return pEvent;
     }
```

5. isHeaderComplete()

   The adapter must make the isHeaderComplete callback to determine if the headers are fully read in.

   The following sample shows this callback:

```
public boolean isHeaderComplete()
    {
/** The callback to isHeaderComplete is required to verify whether all the headers
have been completely read in **/
        return mIsHeaderComplete;
    }
```

6. getMessage()

   After the message headers are completely read in, the framework calls for a getMessage state. At this point, the adapter constructs a message object. The getMessage() only examines the headers and other context data.

   The following sample shows this callback:

```
public IAONSMessage getMessage() throws AdapterException
    {
    /** Get Message Builder **/
                IMessageBuilder msgBuilder =
                    ( (EmbeddedAdapter) getAdapter()).getAdapterManager().
                    getMessageBuilder();

    /** Create Message Context **/
                IMessageContext msgCtx
                 = msgBuilder.createMessageContext();

    /** Create AON Message **/
                IMessageHeaders msgHdrs =
               msgBuilder.createMessageHeaders();
               mAONMessage =
               msgBuilder.createAONMessage(mMsgType, msgCtx,
               msgHdrs);
               return mAONMessage;
        }
```

7. isMessageReadComplete()

   The adapter code checks isMessageReadComplete to determine if the message is fully read in.

   The following sample code shows this callback:

```
public boolean isMessageReadComplete()
    {
        return mIsMessageComplete;
    }
```

8. fetchMessage()

   Each time a message arrives on the connection, the adapter code calls fetchMessage repeatedly until the message body is fully read in. Return values can be read or write events.

   The following sample shows this callback:

```
public int fetchMessage(int pEvent, IConnectionContext pContext) throws
        AdapterException
    {
/** In this callback, the adapter parses the message body. The callback to this method
is repeatedly made, until the message body is read-in. **/
/** If the body is successfully read-in, the adapter returns the status EVENT_READ **/
/** If there is an error in reading the body, appropriate event such as
EVENT_OUT_OF_BUFFER, will be returned **/
        pEvent = IAdapterConstants.EVENT_READ;
                return pEvent;
      }
```

9. getMessage()

   When the message has been completely fetched, the adapter code calls getMessage again. At this point, the custom adapter expects that the message object has content.

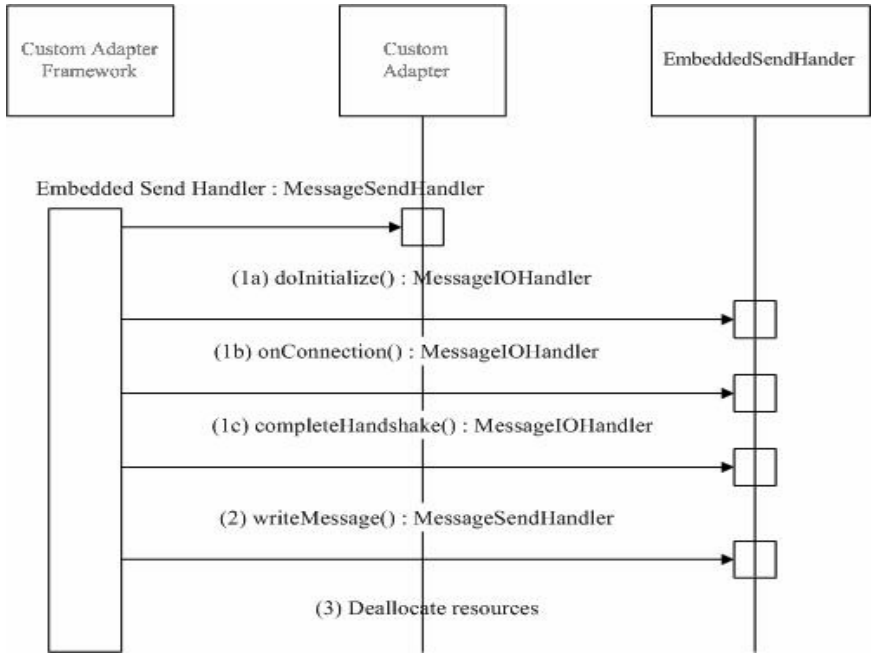
   If the dispatched message is a Request message, the framework classifies it according to the user-defined message type. The message type may have an associated Policy Execution Plans (PEP). AON executes the Request part of the PEP, if one is associated.

10. When all the PEP processing is done, the adapter deposits the message into the AON output queue.

## MessageSendHandler

The MessageSendHandler class must execute in the order shown in Figure 3-3 below.

*Figure 3-3 Embedded Adapter Send Handler*




As the above figure indicates, the Embedded Send Handler code, which extends MessageSendHandler, executes in the following sequence:

1. The adapter code uses the MessageSendHandler class to receive the AON message.
2. Starting up, the code executes the following:
   - doInitialize()
   - onConnection()
   - completedHandshake()
3. The code uses the callback writeMessage to write the message to the endpoint.

   When the write is complete, the adapter code updates the state to ST_COMPLETE.
4. The code deallocates (closes) obtained resources according to conditions as follows:
   - If the message is a one-way request, the adapter code closes (deallocates) any obtained resources.
   - If the message is a two-way request/response, the adapter code uses the Read handler class to receive the response. After the response is sent, the adapter code closes (deallocates) any obtained resources.

# Embedded Adapter Samples

This section provides samples of the three types of Java files that must be included in an embedded adapter.

- TLVReceiverHandler.java, page 3-12
- TLVSendHandler.java, page 3-18

## TLVReceiverHandler.java

This Java coded sample contains the Receive Handler.

```
package com.cisco.aons.adapter.stream.tlv;

import com.cisco.aons.adapter.*;
import com.cisco.aons.message.IAONSMessage;
import com.cisco.aons.message.IMessageHandler;
import com.cisco.aons.io.IDataReader;
import com.cisco.aons.io.IAdapterReader;
import java.io.InputStream;
import java.io.IOException;
import com.cisco.aons.message.IMessageBuilder;
import com.cisco.aons.adapter.EmbeddedAdapter;
import com.cisco.aons.message.IMessageConstants;
import com.cisco.aons.message.IMessageContext;
import com.cisco.aons.message.IMessageHeaders;
import com.cisco.aons.message.IContentDecoder;
import com.cisco.aons.message.DefaultContentDecoder;
import com.cisco.aons.message.IContent;
import com.cisco.aons.message.IAdapterMessageBuilder;
import com.cisco.aons.message.INullContent;

import com.cisco.aons.message.IContentVisitor;
import com.cisco.aons.exception.AONSException;
import com.cisco.aons.message.IContentEncoder;
import com.cisco.aons.net.*;

public class TLVRecvHandler extends MessageReceiveHandler
{
    private static int gInitBufferSize = 512; /** default size **/
    private static final int gDefaultCntBufSize = 4092;
    private static final int ST_READ_TYPE = 0x01;
    private static final int ST_READ_URL_LEN = 0x02;
    private static final int ST_READ_URL = 0x03;
    private static final int ST_READ_CNT_LEN = 0x04;

    private int mMode;
    private IDataReader mDataReader;
    private IAdapterReader mReader;
    private int mMsgType;
    private int mMsgLen = 0;
    private boolean mIsHeaderComplete = false;
    private boolean mIsMessageComplete = false;
    private boolean mIsProcessingDone = false;
    private IAONSMessage mAONMessage;
    private int mMsgFetchSize;
    public String IPaddr = null;
    private int mState = ST_READ_TYPE;
    private int mLen;

    /**
     * doInitialize
     *
     * @param pMode int
     * @param pMessageType int
     * @param pContext IConnectionContext
     */
    public void doInitialize(int pMode, int pMessageType,
                             IConnectionContext pContext)
    {
        mReader = pContext.getReader();
```

```
    ((IAdapterReader) mReader).reset();
    mDataReader = mReader.getAONBuffer().createDataReader();
    mMode = pMode;
    if (pMode == IAdapterConstants.ADAPTER_REQUEST)
    {
    mMsgType = IMessageConstants.APP_REQUEST_MESSAGE;
    }
    else if (pMode == IAdapterConstants.ADAPTER_RESPONSE)
    {
    mMsgType = IMessageConstants.APP_REPLY_MESSAGE;
    }
}

/**
 * onConnection
 *
 * @param pContext IConnectionContext
 * @return int
 */
protected int onConnection(IConnectionContext pContext)
{
    return IAdapterConstants.STATUS_OK;
}

/**
 * completeHandshake
 *
 * @param pEvent int
 * @param pContext IConnectionContext
 * @return int
 */
public int completeHandshake(int pEvent, IConnectionContext pContext)
{
    return IAdapterConstants.STATUS_OK;
}

/**
 * fetchHeader
 *
 * @param pEvent int
 * @param pContext IConnectionContext
 * @return int
 */
public int fetchHeader(int pEvent, IConnectionContext pContext) throws
    AdapterException
{
    try
    {
        pEvent = IAdapterConstants.STATUS_OK;
        int status;
        outerloop:for (; ; )
        {
            if ( (status = mReader.readData(gInitBufferSize)) ==
                IAdapterReader.BUFFERS_UNAVAILABLE)
            {
                /** no available buffers **/
                pEvent = IAdapterConstants.EVENT_OUT_OF_BUFFER;
                break;
            }
            else
            {
                int bytesRead = mReader.getBytesRead();
                if (bytesRead > 0)
                {
```

```
    mDataReader.update();
    switch(mState)
   {
    case ST_READ_TYPE:
            /** Type **/
            int msgType = mDataReader.readByte();
            mState = ST_READ_URL_LEN;
    case ST_READ_URL_LEN:
    if (mDataReader.available() >= 2)
    {
                /** URL Length **/
                byte[] URLLen = new byte[2];
                mDataReader.read(URLLen);
                mLen = (URLLen[0] << 8);
                mLen |= (URLLen[1] << 0);
                mState = ST_READ_URL;
    }
    else
    {
    pEvent = IAdapterConstants.EVENT_READ;
    break outerloop;
    }
    case ST_READ_URL:
    if (mDataReader.available() >= mLen)
    {
            /** URL Buffer **/
            byte[] urlBuffer = new byte[mLen];
            mDataReader.read(urlBuffer);
            IPaddr = new String(urlBuffer);
    }
    else
    {
    pEvent = IAdapterConstants.EVENT_READ;
    break outerloop;
    }
    case ST_READ_CNT_LEN:
    if (mDataReader.available() >= 4)
    {
            /** Content Length **/
            byte[] cntent = new byte[4];
            mDataReader.read(cntent);

            mMsgLen = (cntent[0] << 24) & 0xff000000;
            mMsgLen |= ((cntent[1] << 16) & 0x00ff0000);
            mMsgLen |= ((cntent[2] << 8) & 0x0000ff00);
            mMsgLen |= ((cntent[3] << 0) & 0x000000ff);
            mMsgFetchSize = mMsgLen;
            mIsHeaderComplete = true;
            if (mDataReader.available() >=  mMsgLen)
            {
            mIsMessageComplete = true;
            }
            mState = 0;
    }
     }
      break outerloop;
}
else if (mDataReader.available() == 0
         && status == IAdapterReader.EOF)
{
    pEvent = IAdapterConstants.EVENT_CLOSE;
    break;
}
else
```

```
                    {
                        pEvent = IAdapterConstants.EVENT_READ;
                        break;
                    }
                }
            }
            return pEvent;
        }
        catch (IOException ioEx)
        {
            throw new AdapterException(ioEx);
        }
    }

    /**
     * getMessage
     *
     * @return IAONSMessage
     */
    public IAONSMessage getMessage() throws AdapterException
    {

        try
        {
            if (mAONMessage == null)
            {
                IMessageBuilder msgBuilder =
                    ( (EmbeddedAdapter) getAdapter()).getAdapterManager().
                    getMessageBuilder();

                IMessageContext msgCtx = msgBuilder.createMessageContext();

                String location=IPaddr;
                msgCtx.setDestination(new URI(location));
                int protocol = ((EmbeddedAdapter)
getAdapter()).getAdapterManager().getProtocol();
                msgCtx.setDestinationProtocol(protocol);
                msgCtx.setSourceProtocol(protocol);

                IMessageHeaders msgHdrs = msgBuilder.createMessageHeaders();

                IContent content = null;
                if (mIsMessageComplete)
                {
                    IContentDecoder cd = new DefaultContentDecoder(mDataReader,
                        0, null);
                    content =
                        ( (IAdapterMessageBuilder) msgBuilder).
                        createStreamContent(cd);
                    this.mIsProcessingDone = true;
                }
                else
                {
                    content = new INullContent()
                    {
                        public void close() throws IOException
                        {
                            mDataReader.close();
                        }

                        public InputStream getInputStream()
                        {
                            return null;
                        }
```

```
                    public boolean isInputStreamAvailable()
                    {
                        return false;
                    }

                    public void acceptVisitor(IContentVisitor pVisitor) throws
                        AONSException
                    {
                        pVisitor.visitNullContent(this);
                    }

                    public void encode(IContentEncoder pEncoder)
                    {
                        pEncoder.setContentSize(0);
                    }

                    public int getContentType()
                    {
                        return IContent.NULL_CONTENT;
                    }
                };

            }
            mAONMessage = msgBuilder.createAONMessage(mMsgType, msgCtx,
                msgHdrs);
        }
        return mAONMessage;

    }
    catch (Exception ex)
    {
        throw new AdapterException(ex);
    }
}

/**
 * fetchMessage
 *
 * @param pEvent int
 * @param pContext IConnectionContext
 * @return int
 */
public int fetchMessage(int pEvent, IConnectionContext pContext) throws
    AdapterException
{

    try
    {
        pEvent = IAdapterConstants.STATUS_OK;
        int status = 0;

        if (!mIsMessageComplete)
        {
            for (; ; )
            {
                if ( (status = mReader.readData(mMsgFetchSize)) ==
                    IAdapterReader.BUFFERS_UNAVAILABLE)
                {
                    /** no available buffers **/
                    pEvent |= IAdapterConstants.EVENT_OUT_OF_BUFFER;
                    break;
                }
                else
```

```
                    {
                        long bytesRead = mReader.getBytesRead();
                        if (bytesRead > 0)
                        {
                            mMsgFetchSize -= bytesRead;
                            mDataReader.update();
                            if (mDataReader.available() >= mMsgLen)
                            {
                                this.mIsMessageComplete = true;
                                    IMessageBuilder msgBuilder =
mAONMessage.getMessageBuilder();
                                    IContentDecoder cd = new
DefaultContentDecoder(mDataReader, 0, null);
                                    IContent content = ((IAdapterMessageBuilder)
msgBuilder).createStreamContent(cd);

mAONMessage.getMessageBuilder().replaceContent(mAONMessage, content);
                                    this.mIsProcessingDone = true;
                                break;
                            }
                        }
                        else if (status == IAdapterReader.OK && bytesRead == 0)
                        {
                            pEvent |= IAdapterConstants.EVENT_READ;
                            break;
                        }
                        else if (status < 0)
                        {
                            pEvent |= IAdapterConstants.EVENT_CLOSE;
                            break;
                        }
                    }
                }
            }

            return pEvent;
        }
        catch (Exception ex)
        {
            throw new AdapterException(ex);
        }
    }

    /**
     * isHeaderComplete
     *
     * @return boolean
     */
    public boolean isHeaderComplete()
    {
        return mIsHeaderComplete;
    }

    /**
     * isMessageReadComplete
     *
     * @return boolean
     */
    public boolean isMessageReadComplete()
    {
        return mIsMessageComplete;
    }

    /**
```

```
     * doneMessageProcessing
     *
     * @return boolean
     */
    public boolean doneMessageProcessing()
    {
        return mIsProcessingDone;
    }
}
```

## TLVSendHandler.java

This Java-coded sample class contains the send handler.

```
package com.cisco.aons.adapter.stream.tlv;

import com.cisco.aons.adapter.*;
import com.cisco.aons.message.IAONSMessage;
import com.cisco.aons.adapter.IAdapterConstants;
import com.cisco.aons.message.IContentEncoder;
import com.cisco.aons.message.IContentCanonicalizer;
import com.cisco.aons.message.DefaultContentEncoder;
import com.cisco.aons.adapter.EmbeddedAdapter;
import com.cisco.aons.io.IDataReader;


public class TLVSendHandler extends MessageSendHandler
{
    private static final int ST_INIT = 0x01;
    private static final int ST_WRITE = 0x02;
    private static final int ST_COMPLETE = 0x03;

    private int mMode;
    private int mMsgType;
    private int mContentLength;
    private IAONSMessage mAONMessage;
    private int mState;
    private IContentEncoder mSCE;
    private IContentCanonicalizer mCS;
    private IDataReader mDataReader;

    public TLVSendHandler()
    {
        super();

    }

    /**
     * doInitialize
     *
     * @param pMode int
     * @param pMessageType int
     * @param pContext IConnectionContext
     */

    public void doInitialize(int pMode, int pMessageType,
                             IConnectionContext pContext)
    {
        this.mMode = pMode;
        this.mMsgType = pMessageType;
        this.mAONMessage = getMessage();
        this.mState = ST_INIT;
    }
```

```
    /**
     * onConnection
     *
     * @param pContext IConnectionContext
     * @return int
     */
    protected int onConnection(IConnectionContext pContext)
    {
        return IAdapterConstants.STATUS_OK;
    }

    /**
     * completeHandshake
     *
     * @param pEvent int
     * @param pContext IConnectionContext
     * @return int
     */
    public int completeHandshake(int pEvent, IConnectionContext pContext)
    {
        return IAdapterConstants.STATUS_OK;
    }

    /**
     * fetchMessage
     *
     * @param pEvent int
     * @param pContext IConnectionContext
     * @return int
     */
    public int fetchMessage(int pEvent, IConnectionContext pContext)
    {
        return IAdapterConstants.STATUS_OK;
    }


    /**
     * writeMessage
     *
     * @param pEvent int
     * @param pContext IConnectionContext
     * @return int
     */
    protected int writeMessage(int pEvent, IConnectionContext pContext) throws
AdapterException
    {
        pEvent = IAdapterConstants.STATUS_OK;

        try
        {
            switch (mState)
            {
                case ST_INIT:
                    mCS = (IContentCanonicalizer) getMessage().getContent();
                    mSCE = new DefaultContentEncoder(
                        ( (EmbeddedAdapter) getAdapter()).getAdapterManager().
                        getBufferManager());
                    mCS.encode(mSCE);
                    mState= ST_WRITE;
                    mDataReader = mSCE.createDataReader();
                case ST_WRITE:
                int status = mDataReader.writeTo(getWriter());
                    if (mDataReader.available() == 0)
```

```
                    {
                        mState = ST_COMPLETE;
                        mDataReader.close();
                        break;
                    }
                    else
                    {
                    if (status < 0)
                    {
                    pEvent |= IAdapterConstants.EVENT_CLOSE;
                    break;
                    }
                    else
                    {
                    pEvent = IAdapterConstants.EVENT_WRITE;

                    break;
                    }
                    }
                case ST_COMPLETE:
            }
        }
        catch(Exception ex)
        {
            throw new AdapterException(ex);
        }
        return pEvent;
    }
}
```

# Developing a Standalone Adapter

For standalone adapters, the framework permits two types of message dispatch: direct and callback. For direct message dispatch, the adapter sends a message directly to the custom adapter framework using its thread of control. For callback message dispatch, the adapter notifies the framework of a message arrival by dispatching a callback.

This section explains how to develop standalone adapters, focusing on requirements and essential code components.



## Adapter Names and Versions

The adapter must contain both of the following unique identifiers:

- Registered name—The adapter must have a fully qualified class name. The registered name is used to uniquely identify the adapter within AON.
- Display name—The adapter should have a user friendly, abbreviated name.

The adapter must have a version number following the format “[major].[patch number].” The custom adapter developer assigns the major number. The ADS and AMC assign the patch number during packaging and deployment.

# Adapter Code Components

Using a Java-editor, write the adapter code. Your code must include the following base classes:

- Main Standalone Adapter Class

  Extend the main adapter class from StandaloneAdapter (package com.cisco.aons.adapter).

  This class creates sockets, listens, and accepts connections from a transmitter (usually, the "client").

- Standalone Receive Handler Class

  This class receives data from an already accepted connection, creates a message object, and dispatches the message.

- Standalone Send Handler Class

  Extend the message send class from AbstractOutboxHandler (package com.cisco.aons.adapter). This class receives the message, converts it to the output message format, and sends it to the receiver (usually, the "Sender").

Your adapter code may also include the following classes:

- Decoder Class

  Provide a decoder class that can decode an application message.

- Encoder Class

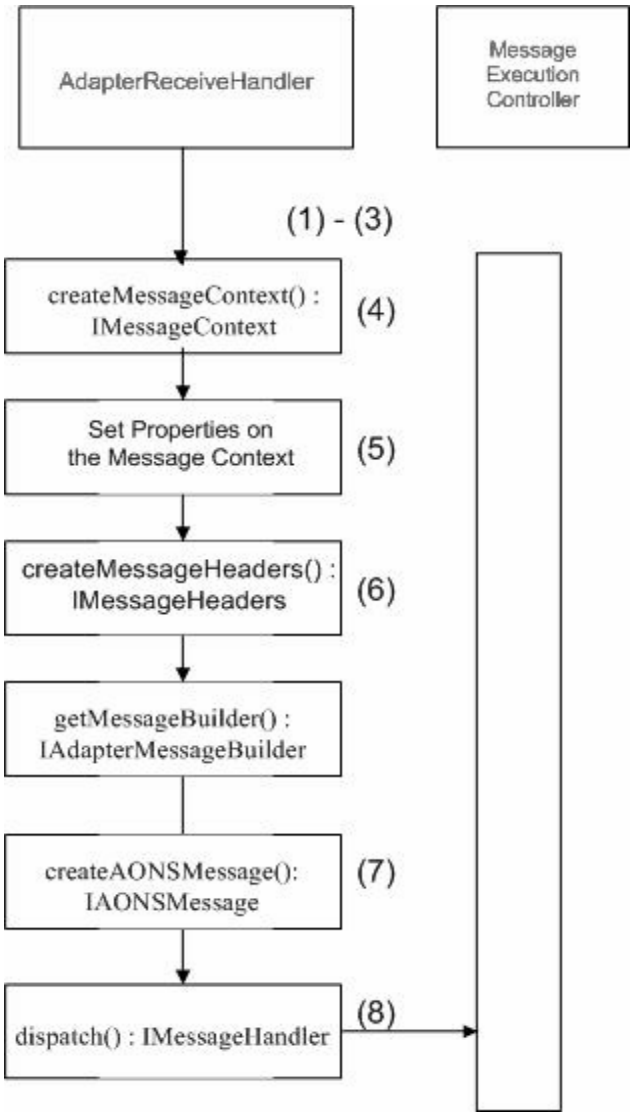
  Provide an encoder class that can encode an application message.

For an example, see the "Custom Adapter API Specification" section on page 3-76.

## Receive Handler

The standalone receive handler executes in the sequence shown in Figure 3-4 below.

*Figure 3-4 Standalone Adapter Receive Handler*




This figure indicates that the Standalone Receive Handler code executes as follows:

1. Receives messages.

   The adapter uses the reader class to receive all messages. It can provide its own mechanism for receiving messages. For example, it can have its own socket for listening and accepting connections. Alternatively, it can extend the StandaloneMessageReader (Package: com.cisco.aons.adapter) to read the data.

2. Decodes the customer application.

   The adapter code may include the decoder class to decode customer application messages.

3. Parses headers.

The adapter parses header fields according the corresponding protocol specification.

4. createMessageContext()

   The adapter uses the createMessageContext method which implements the IMessageContext interface. The message context contains network information about source and destination, such as protocol type, source protocol, destination protocol, destination URI, source host, destination host, destination port, and so on.

   The following sample code creates message content context:

```
/** Create Message Context **/
IMessageContext msgCtx = mCtx.getMessageBuilder().createMessageContext();
```

5. Sets properties.

   The adapter code must set properties correctly in order to send the message successfully. The URI must contain the appropriate protocol name.

   The following sample code sets network properties on the message context:

```
/** Set the properties of Message Context **/
int Proto = mCtx.getProtocol();

/** Set Source and Destinal Protocol **/
msgCtx.setSourceProtocol(Proto);
msgCtx.setDestinationProtocol(Proto);

/** Set Destination URI **/
/** The URI must contain the appropriate protocol name **/
msgCtx.setDestination(URI);
```

6. createSteamContent()

   The adapter code creates the content, depending on the content type. The code includes the createStreamContent method, which implements the IAdapterMessageBuilder interface. Typical contents are stream, XML, Simple Object Access Protocol (SOAP) content, and so on.

   The following sample creates stream content:

```
/** Message Builder **/
            IAdapterMessageBuildermMsgBuilder
      =(IAdapterMessageBuilder) mCtx.getMessageBuilder();

            /** Create Stream Content **/
            IContent content=null;
            content = mMsgBuilder.createStreamContent(cdecode);
```

7. createAONMessage()

   The adapter code includes the createAONMessage method, which implements the IAONSMessage interface to create a message object.

   The following sample code creates a message object:

```
    /** Create AON Message **/
            msg = this.mAdapter.getAdapterManager().getMessageBuilder().
       createAONMessage (IMessageConstants.APP_REQUEST_MESSAGE,
      msgCtx, msgHdrs, content);
```

dispatch()

The Custom Adapter SDK enables you to write the code for dispatching two ways:

- The code can use its own thread to read in the message. In this case, the adapter reads in the message and dispatches it to AON for PEP processing. The code uses the dispatch method of the IMessageDispatch class. The following sample code dispatches the AON message:

```
/** Dispatch callback **/
                mCtx.dispatch(this);
```

- The code can use an AON thread to read the message. In this case, the adapter extends the StandaloneMessageReader class and implements the readMessage method. The readMessage method is called in the context of the AON thread. This approach has an advantage—the adapter does not have to maintain its own thread pool to manage reader threads. Instead, it maintains a single thread, which is used to dispatch the callbacks.

If the adapter dispatches a one-way request message, it classifies the message according to the user defined message type. The message type may have an associated PEP. In this case, AON executes the request as part of the PEP.

After all the PEP processing is done, the adapter deposits the message into the AON output queue.

If the adapter dispatches a two-way, request/response message, the adapter stores/saves the context ID/index information in a hash map or a similar data structure to correlate the response, when it arrives.

The following sample code saves the context ID of a message:

```
/** Save the message context id **/
String msgCtxId = msg.getMessageContextId();
mAdapter.put(msgCtxId, mSocket);

private HashMap mConnMap = new HashMap();
protected void put(String pMsgCtxId, Socket pSocket)
{
mConnMap.put(pMsgCtxId, pSocket);
}
```

When a response arrives, the context ID/index information can be correlated with the information saved earlier. This enables the adapter to dispatch the response to the correct endpoint and on the correct context.

The following sample code retrieves the context ID of a message:

```
            protected Socket get(String pMsgCtxId)
{
        return (Socket) mConnMap.remove(pMsgCtxId);
}
```

The adapter code can include the setAttachment method which implements the IMessageHandlerAttachment interface.

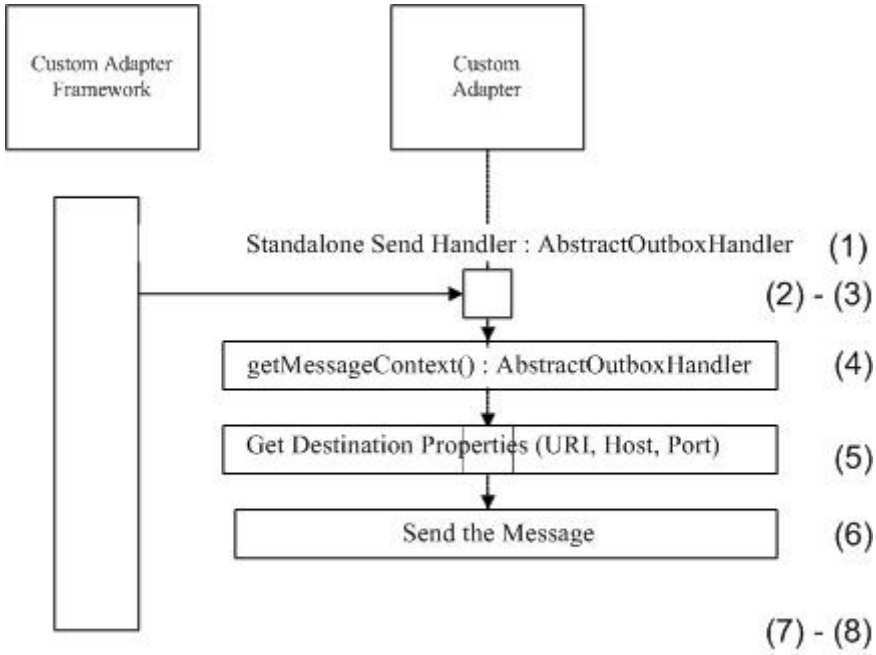
The following sample code sets attachments:

```
pMsgHandler.setAttachment(mAdapter.JMS_RUNNEABLE_IDX,
   new IJMSAdapterRunneable.MsgHandlerWrapper(mRunneable));
```

## Send Handler

The send handler transmits a message object. This class extends the AbstractOutboxHandler, package com.cisco.aons.adapter. The code executes the processes represented in Figure 3-5.

*Figure 3-5 Standalone Adapter Send Handler*




The figure indicates that the standalone adapter send handler executes in these steps:

1. AbstractOutboxHandler

   The adapter includes the sender class to transmit the message object. This class extends the AbstractOutboxHandler (package com.cisco.aons.adapter).

2. getAttachment()

   The adapter code can include the getAttachment method of the IMessageHandler class (implementing the IMessageHandlerAttachment interface) to retrieve message attachments.

   The following sample code retrieves attachments.

   ```
   mMessageHandler.getAttachment(((JMSAdapter) mAdapter).JMS_RUNNEABLE_IDX);
   ```

3. Encode class.

   The adapter code can include the encode class to encode messages.

4. getMessageContext()

   The adapter code includes the getMessageContext method (implementing the IMessageContext interface) to get the message context.

   The following sample code gets the message context:

   ```
   /** Get the Message Context **/
   IMessageContext msgCtx = mMessage.getMessageContext();
   ```

5. Get properties.

   The adapter code gets properties on the message context. These can include destination URI, host, port, and so on.

   The following sample code gets network properties on the message context:

   ```
   ** Get the Destination URI **/
   destURI = mMessage.getMessageContext().getDestination();
   ```

```
** Get the Destination Host **/
ost = destURI.getHost();

** Get the Destination Port **/
ort = destURI.getPort();
```

6. Writes the message to the endpoint.

   The adapter code writes the message to the endpoint.

7. Updates state information to indicate that the message has been written.

   The following sample code updates the state:

```
/** Update the state **/
          this.mMessageHandler.updateState();
```

8. Closes and deallocates obtained resources.
   - If the message is a one-way request message, the code closes and deallocates any obtained resources such as sockets.
   - If the message is a two-way request/response, the code waits to receive the response using the read handler class. After the response is received, the code closes and deallocates any obtained resources.

# Standalone Adapter Samples

This section presents samples of the Java files that must be included in a standalone adapter.


- ReceiveRunnable.java, page 3-26
- TLVSendHandler.java, page 3-30


## ReceiveRunnable.java

This Java-coded sample contains the standalone receive handler.

```
package com.tlv.standalone.mec;

import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.io.OutputStream;
import java.io.IOException;

import com.cisco.aons.adapter.IStandaloneAdapterManager;
import com.cisco.aons.adapter.StandaloneMessageReader;
import com.cisco.aons.message.IAONSMessage;
import com.cisco.aons.message.IMessageHandler;
import com.cisco.aons.message.IMessageHeaders;
import com.cisco.aons.message.MessageParseException;
import com.cisco.aons.adapter.*;
import com.cisco.aons.exception.AONSException;
import com.cisco.aons.message.IMessageConstants;
import com.cisco.aons.message.IAdapterMessageBuilder;
import com.cisco.aons.message.IMessageContext;
import com.cisco.aons.net.*;
```

```
import com.cisco.aons.message.IContent;
import com.cisco.aons.message.IContentDecoder;
import com.cisco.aons.log.Log;
import com.cisco.aons.message.IMessageHandlerCallback;

/**
 * <p>Title: Application Oriented Networking Systems (AON)</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: Cisco Systems</p>
* @version 1.0
 */

public class ReceiveRunnable extends StandaloneMessageReader
    implements Runnable, IMessageHandlerCallback
{
    // private instance variables
    private Socket mSocket;
    private IStandaloneAdapterManager mCtx;
    private TLVStandaloneAdapter mAdapter;
    private int mMsgType;
    private String mMsgCtxId;
   // public static int Msgtype;
    private IAONSMessage mMsg;
    private Log mLogger;
    private TLVStandaloneAdapter.ClientInfo mInfo = new TLVStandaloneAdapter.ClientInfo();

    public ReceiveRunnable(TLVStandaloneAdapter pAdapter, Socket pSocket, int pMsgType)
    {
        this(pAdapter, pSocket, pMsgType, null);
    }

    public ReceiveRunnable(TLVStandaloneAdapter pAdapter, Socket pSocket, int pMsgType,
String pMsgCtxId)
    {
        super(pAdapter);
        mAdapter = pAdapter;
        mSocket = pSocket;
        mMsgType = pMsgType;
        mMsgCtxId = pMsgCtxId;
        mCtx = mAdapter.getStandaloneAdapterManager();
        mLogger = mCtx.getLogger();
    }

    public void run()
    {
        try
        {
            mLogger.debug("mSocket = " + mSocket);
            InputStream in = null;

            in = mSocket.getInputStream();
            DataInputStream readin = new DataInputStream(in);

            /** Type **/
            int msgType = readin.readByte();
            mLogger.debug("Type is : " + msgType);

            /** URL Length **/
            byte[] urlLen = new byte[2];
            readin.readFully(urlLen);
            int len = (urlLen[0] << 8);
            len |= (urlLen[1] << 0);
            mLogger.debug("URL length is : " + len);
```

```
                /** URL Buffer **/
                byte[] urlBuffer = new byte[len];
                readin.readFully(urlBuffer);
                mLogger.debug("URL is : " + new String(urlBuffer));

                /** Content Length **/
                int cntLen = readin.readInt();
                mLogger.debug("content length is : " + cntLen);

                byte[] cntnt = new byte[cntLen];
                readin.readFully(cntnt);

                mLogger.debug("content is : " + new String(cntnt));

                /** Content Decoder **/
                IContentDecoder cdecode = new ContentDecoderImpl(new
ByteArrayInputStream(cntnt),
                    len, null);

                /*** Create Message Context ***/
                IMessageContext msgCtx = mCtx.getMessageBuilder().createMessageContext();

                /** Set the properties of Message Context **/
                int Proto = mCtx.getProtocol();

                /** Set Source and Destinal Protocol **/
                //msgCtx.setSourceProtocol(Proto);
                //msgCtx.setDestinationProtocol(Proto);

                /** Set Destination **/
                String Dest = new String(urlBuffer);
                mLogger.debug("Dest URI = " + Dest);

                /** Set Dest URI **/
                msgCtx.setDestination(new URI(Dest));

                /** Create Message Headers **/
                IMessageHeaders msgHdrs = mCtx.getMessageBuilder().createMessageHeaders();

                /** Message Builder **/
                IAdapterMessageBuilder mMsgBuilder = (IAdapterMessageBuilder)
mCtx.getMessageBuilder();

                /** Create Stream Content **/
                IContent content = null;
                content = mMsgBuilder.createStreamContent(cdecode);

                /** Request Message **/
                if (mMsgType == IMessageConstants.APP_REQUEST_MESSAGE)
                {
                    /*** Create AON Message ***/
                    mMsg = this.mAdapter.getAdapterManager().getMessageBuilder().
                        createAONMessage
                        (IMessageConstants.APP_REQUEST_MESSAGE, msgCtx, msgHdrs, content);
                    mInfo.mMsgType = msgType;
                    mInfo.mSocket = mSocket;
                    /** Dispatch callback **/
                    mCtx.dispatch(this);

                }
                /** Response Message **/
                else if (mMsgType == IMessageConstants.APP_REPLY_MESSAGE)
                {
```

```
                    mLogger.debug("mMsgCtxId = " + mMsgCtxId);

                    /** Create AON Message with context id **/
                    mMsg = this.mAdapter.getAdapterManager().getMessageBuilder().
                        createAONMessage(IMessageConstants.APP_REPLY_MESSAGE,
                                         msgCtx, msgHdrs, content, mMsgCtxId);

                    /** Dispatch callback **/
                    mCtx.dispatch(this);
                }
                /** Unhandled Message Type **/
                else
                {
                    mLogger.debug("ERROR: Unhandled Message Type");
                }
            }

            catch (AdapterException e)
            {
                mLogger.fatal(e.getMessage(), e);
            }
            catch (IOException e)
            {
                mLogger.fatal(e.getMessage(), e);
            }
            catch (MessageParseException e)
            {
                mLogger.fatal(e.getMessage(), e);
            }
            catch (AONSException e)
            {
                mLogger.fatal(e.getMessage(), e);
            }
        }

        /**
         * readMessage
         *
         * @return IAONSMessage
         */
        public void readMessage() throws AdapterException
        {
            mAdapter.getStandaloneAdapterManager().dispatch(mMsg, this);
        }

        /**
         * updateMessageContext
         *
         * @param pMsgHandler IMessageHandler
         */
        public void updateMessageContext(IMessageHandler pMsgHandler)
        {
            if(mMsgType == IMessageConstants.APP_REQUEST_MESSAGE)
            {
                // save the message context id
                String msgCtxId = mMsg.getMessageContextId();
                mLogger.debug("msgCtxId = " + msgCtxId);
                mAdapter.put(msgCtxId, mInfo);
            }
        }
    }
```

## TLVSendHandler.java

This Java-coded sample contains the standalone send handler.

```
package com.cisco.aons.adapter.stream.tlvstand;

import com.cisco.aons.adapter.*;
import com.cisco.aons.adapter.stream.tlvstand.SocketUtil;
import com.cisco.aons.message.IContentCanonicalizer;
import com.cisco.aons.message.IContentEncoder;
import com.cisco.aons.message.IMessageConstants;
import com.cisco.aons.message.IMessageContext;
import com.cisco.aons.message.MessageWriteException;

import java.net.Socket;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintStrea

import com.cisco.aons.net.URI;
import java.io.DataOutputStream;

public class TLVSendHandler extends AbstractOutboxHandler
{
    public TLVSendHandler()
    {
    }

    public void sendMessage(int pMsgType) throws com.cisco.aons.adapter.
        AdapterException
    {
        URI destURI = null;
        String host = null;
        int port = 0;

        try
        {
            TLVStandaloneAdapter adapter = (TLVStandaloneAdapter)
             mAdapter;
            /** Content Encoder **/
            IContentEncoder cencode = new ContentEncoderImpl();

            /** Content Canonicalizer **/
            IContentCanonicalizer canon = (IContentCanonicalizer)
            mMessage.getContent();


            canon.encode(cencode);

            /** Get the Message Context **/
            IMessageContext msgCtx = mMessage.getMessageContext();

            /** Get the Destination URI **/
            destURI = mMessage.getMessageContext().getDestination();

            /** Get the Destination Host **/
            host = destURI.getHost();

            /** Get the Destination Port **/
            port = destURI.getPort();
```

```
/** Output Stream **/
ByteArrayOutputStream outstream = null;
outstream = (ByteArrayOutputStream)
cencode.getOutputStream();
Socket socket = null;
if ( (pMsgType & IMessageConstants.APP_REQUEST_MESSAGE) ==
    IMessageConstants.APP_REQUEST_MESSAGE)
{
    /** Socket connection **/
    socket = new Socket(host, port);

}
else if ( (pMsgType & IMessageConstants.APP_REPLY_MESSAGE)
     ==             IMessageConstants.APP_REPLY_MESSAGE)
{
    socket = adapter.get(mMessage.getMessageContextId());
}

if (socket != null)
{
    /** Write to the socket **/
    DataOutputStream dos = new DataOutputStream(socket.
        getOutputStream());
    byte[] buffer = new byte[1];
    buffer[0] = (byte) ReceiveRunnable.Msgtype;
    dos.write(buffer);

    String uri = destURI.toString();
    int len = uri.length();
    buffer = new byte[2];
    buffer[0] = (byte) (len >> 8);
    buffer[1] = (byte) (len >> 0);
    dos.write(buffer);
    dos.writeBytes(uri);

    len = outstream.size();
    dos.writeInt(len);
    dos.write(outstream.toByteArray());

    /** Update the state **/
    this.mMessageHandler.updateState();

    if ( (pMsgType & IMessageConstants.APP_REQUEST_MESSAGE)
    == IMessageConstants.APP_REQUEST_MESSAGE)
    {
        if (ReceiveRunnable.Msgtype == 0)
        {
            /** Close the endpoint socket **/
            socket.close();
        }
        else
        {

            /** Create new thread to read the Response
            Message **/
            new Thread(
                new ReceiveRunnable(adapter, socket,
                                    IMessageConstants.
                                    APP_REPLY_MESSAGE,

                 mMessage.getMessageContextId())).start();
        }
    }
```

```
                else
                {
                    /** close client socket **/
                    socket.close();
                }
            }
        }
        catch (IOException ioEx)
        {
            throw new AdapterException(ioEx);
        }
        catch (MessageWriteException e1)
        {
            e1.printStackTrace();
        }
    }

}
```

# Packaging the Custom Adapter and Incorporating it into AON

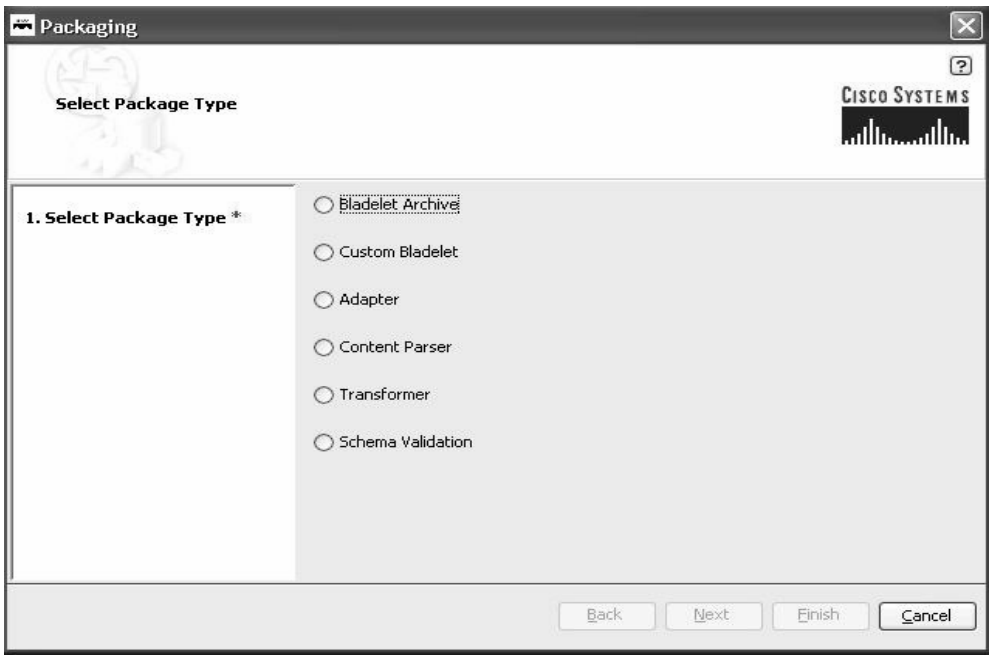
The same processes are used to package all embedded and standalone adapters. You use the AON Development Studio (ADS) to package the adapter file including all necessary components. Then, you use the AON Management Console (AMC) to upload and incorporate the new packaging into AON.

This process is used for all custom adapters including the MQ Adapter.

**Step 1** Using the ADS, select **Tools > Packaging**.

The AON Packaging window appears. For example:

***Figure 3-6 AON Packaging window***

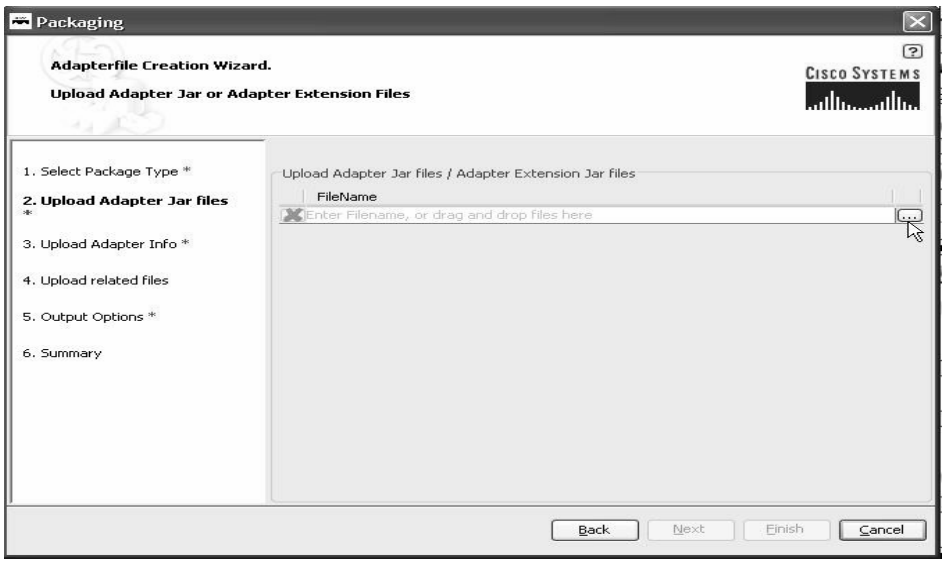



As the figure shows, this window is used for all AON packaging activities.

**Step 2** Select **Adapter** and click **Next**.

The Upload Adapter Jar or Adapter Extension Files screen appears.

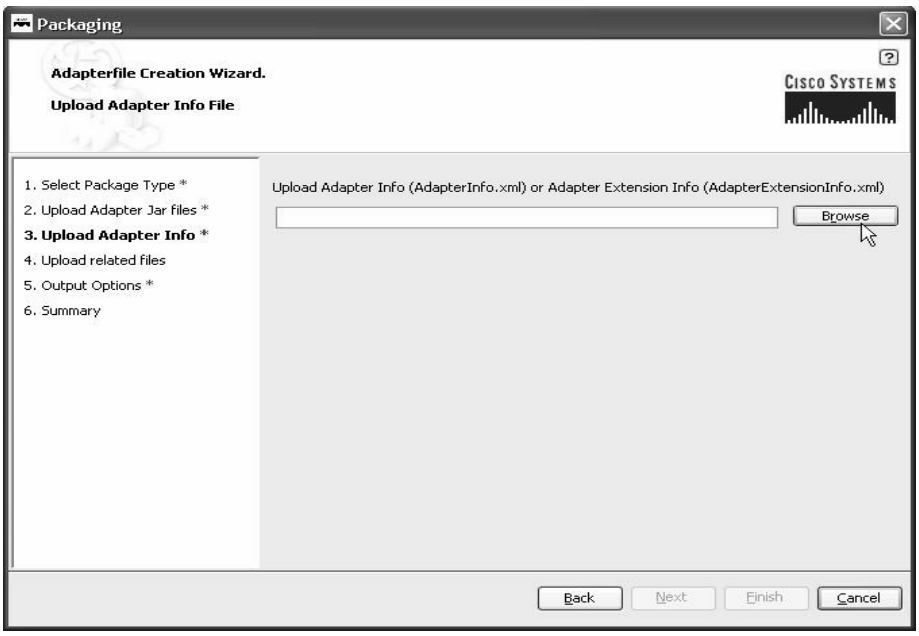
***Figure 3-7 Upload Adapter Jar or Adapter Extension Files window***




**Step 3** Using the selection button, locate adapter Java archive (.jar) files and click **Next**.

The Upload Adapter Info File screen appears.

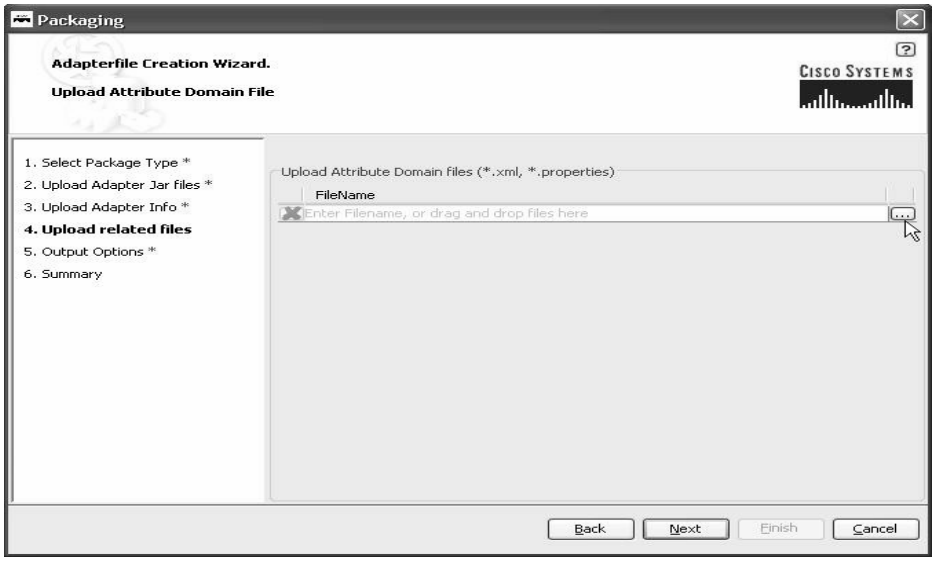
***Figure 3-8 Upload Adapter Info File window***




**Step 4** Click **Browse,** locate the AdapterInfo.xml or AdapterExentionInfo.xml file and click **Next**.

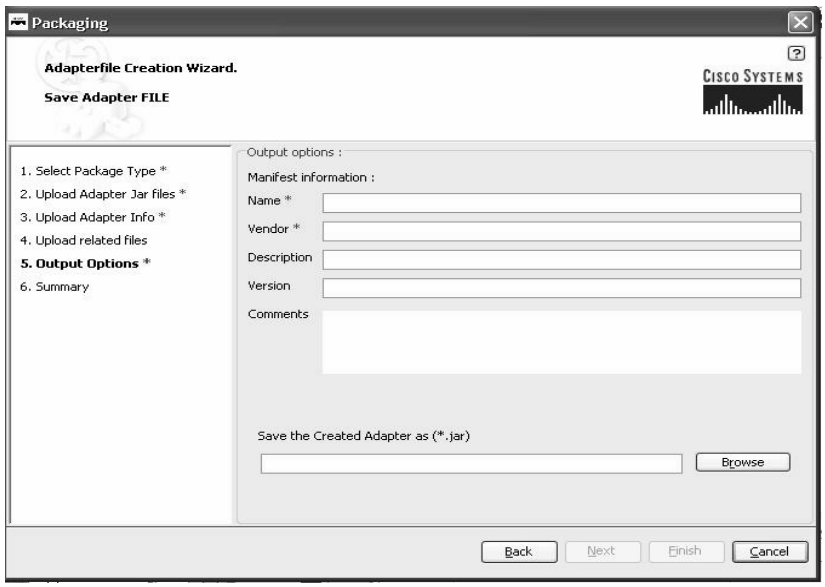
The Upload Attribute Domain File screen appears.

*Figure 3-9 Upload Attribute Domain File browser field*




**Step 5** Using the selection button, locate the attribute domain files (.xml and .properties) and click **Next**. The Save Adapter File window appears.

*Figure 3-10 Save Adapter File window*




**Step 6** Fill in the manifest fields (red star = obligatory) and click **Next**.

*Table 3-1 Adapter File Creation Manifest Information*

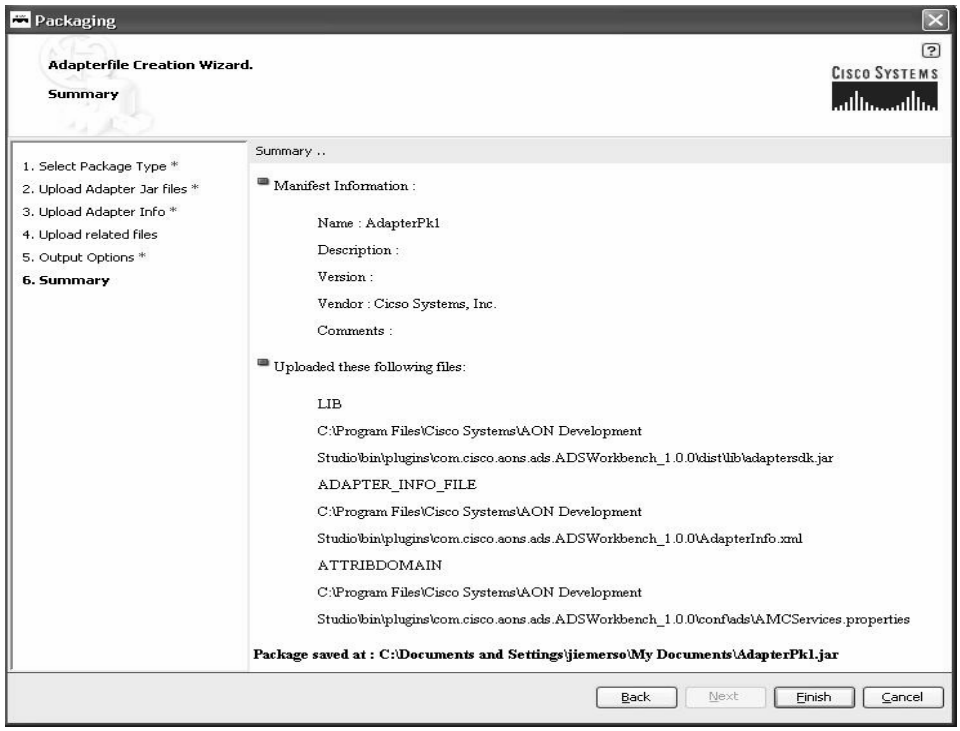
| Field | Description |
|---|---|
| Name | Filename; for example, an adapter file. |
| Vendor | Vendor associated with the file. For example, Cisco Systems, Inc. |

*Table 3-1 Adapter File Creation Manifest Information (continued)*

| Field | Description |
|---|---|
| Description | (Optional) Description of the file. |
| Version | (Optional) File version. |
| Comments | (Optional) Comments about the file. |
| Save the Created Adapter as | Automatically filled in the Name field name and current default location. You may use the Browse button to find a different location. |

A Packaging Summary window appears. For example:

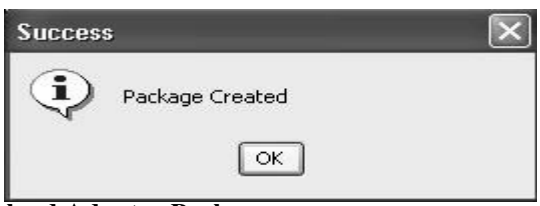
*Figure 3-11 Adapterfile creation summary window*




**Step 7** Review the package contents and click **Finish**.

For a list of the components in an adapter package, see the "Adapter Package Content" section on page 3-37. A package completion message appears. For example:

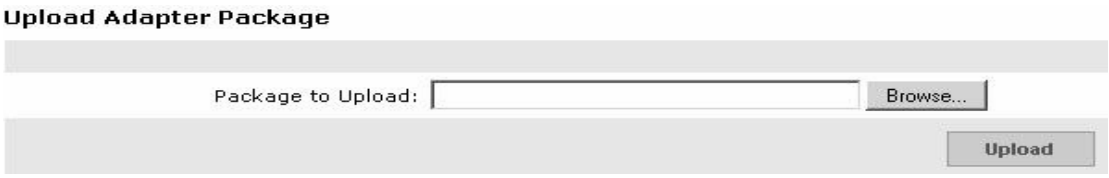
*Figure 3-12 Success Message dialog*




**Step 8** In AMC, select **Admin > Upload Adapter Package**.

An upload dialog (Figure 3-15) appears.

*Figure 3-13 Selecting Package to Upload*




**Step 9** Using the Browse button, locate the recently prepared package and click **Upload**.

The package is uploaded. ADS lists the newly uploaded package.

*Figure 3-14 Uploading and Registering Package*


**Step 10** Click **Register** to register the package with AMC.

The newly registered package is listed in AMC and ready for use.

# Adapter Use Cases

The following examples illustrate the use of custom adapters.


- HTTP Embedded Adapter Use Case, page 3-36
- Stock Trading Company Embedded Adapter Use Case, page 3-37


## HTTP Embedded Adapter Use Case

A customer needs a new adapter that meets the functional requirements listed below.

- Message format—The adapter must be able to handle messages with the following format:
  - Request/responses line
  - HTTP headers
  - Message body
- Protocol semantics—The adapter must be able to perform the following protocol-related actions:
  - Fetch headers:

    (a) Read data

    (b) Parse request/response line

    (c) Parse headers
  - Fetch content:

    (a) Read data

(b) Parse content

(c) Create an AON message

– Keep the connection alive

### Stock Trading Company Embedded Adapter Use Case

A customer (a major equity trading company) needs an adapter that meets the requirements listed below.

- Message format—The adapter must be able to handle messages with the following format:
  - 8 bytes in message length
  - SOAP message body
- Protocol semantics—The adapter must be able to perform the following protocol-related actions:
  - Maintain a constant TCP connection
  - Request-reply over the same connection

    Responses may arrive out of sequence. Response ordering is not required for AON.
  - AON Policy Execution Plans model must be able to modify the Request to add a Request-ID for correlation purposes, while the following requirements are met:

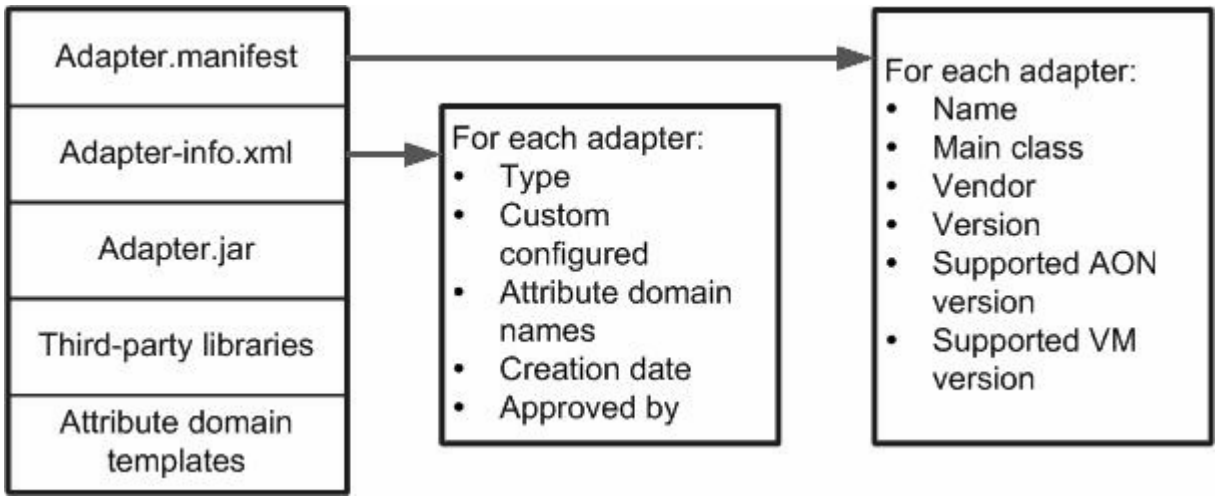
    A globally unique identifier (GUID) bladelet to create the Request-ID.

    AON must have a mechanism (such as a bladelet) to modify the SOAP content.

## Adapter Package Content

When your adapter package is complete, the embedded or standalone package must contain the components listed in Figure 3-15.

*Figure 3-15 Embedded Adapter Package*



# Compiling the Custom Adapter

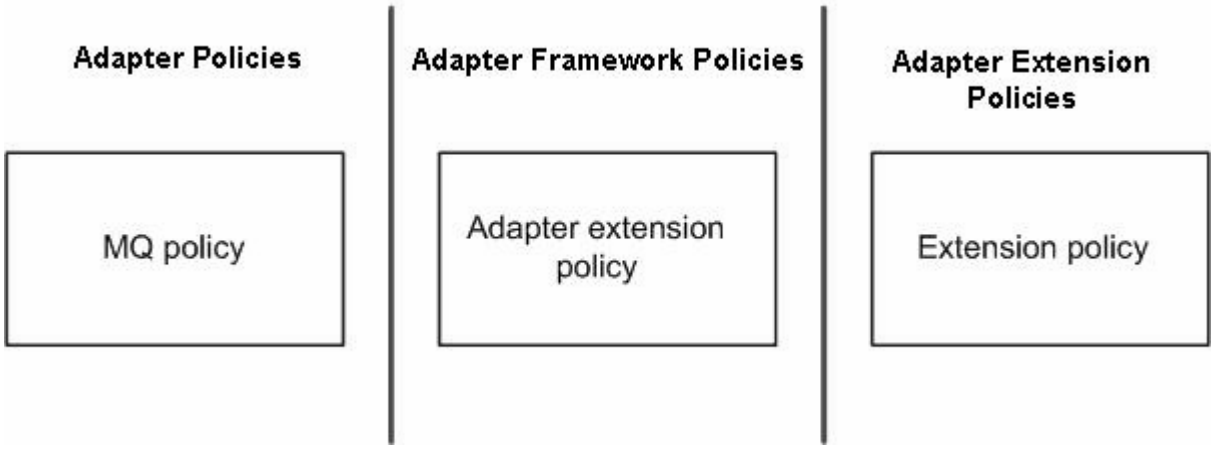
After creating an embedded or standalone adapter, you can use the ant scripting tool to compile the code.

More more information, see Extending the Custom Adapter, page 3-38.

# Extending the Custom Adapter

You can extend a custom adapter by setting certain AON adapter policies. Figure 3-11 shows the relationships of adapter policies, adapter framework policies, and adapter extension policies.

***Figure 3-16 Adapter Extension - Policy Relationships***




This figure shows the relationships of the following policies:

- Adapter Policies

  An adapter has one or more policies for its configuration and assumes that one of them contains configuration information for the supported extensions. The definition of this policy should have a reference to the adapter extension policy which is defined by the AON team that has extension policy name.

- Adapter Framework Policies

  The Adapter framework includes the adapter extension policy which contains:

  - Extension name: The name of the extension
  - Extension Type: The extension type
  - Extension Class: The main class of the extension
  - Extension policy name: The configuration policy of the extension.

- Extension Policy

  The extension policy is defined by the adapter extension developer. However it should contain a well-defined attribute that will reference the adapter policy (MQ policy in Figure 3-18).

For more information, see the class descriptions in the Custom Adapter API Specification, page 3-76.

# Graceful Handling and Adapters

An AON node may run low on memory resources during periods of high message traffic. To get around these impacts, AON includes Graceful Handling mechanisms that reject additional messages when the available resources (Java heap space or native byte buffers) drop below specified levels and restart message handling when higher levels have returned. Graceful Handling operates for embedded (HTTP) and standalone adapters.

To learn more about Graceful Handling, see:

- Graceful Handling and Embedded Adapters, page 3-39
- Graceful Handling and Standalone Adapters, page 3-42

For more about custom adapters, see Developing an Embedded or Standalone Custom Adapter, page 3-3.

## Graceful Handling and Embedded Adapters

When available AON message processing resources are at low points, Graceful Handling mechanisms automatically take effect for embedded adapters. The following sections summarize Graceful Handling for embedded adapters

- Resource Thresholds Triggering Graceful Handling, page 3-39
- Available Memory Level Trigger Points for Graceful Handling, page 3-40
- Registering and Migrating Sources, page 3-41

To learn about Graceful Handling and its application to standalone adapters, see Graceful Handling and Standalone Adapters, page 3-42.

### Resource Thresholds Triggering Graceful Handling

Graceful Handling is triggered by demands on resources that exceed certain thresholds. Generally, available memory (Java heap space) may be limited under the following conditions:

- High message load conditions—An AON node receives an unusually large number of incoming messages.
- Executing bladelets—A few bladelets (Encrypt, Decrypt, VerifySignature, Sign) are resource intensive at runtime. For example, 50MB of Java heap space could be required to encrypt/decrypt a 5MB document.
- Distribute bladelet—The Distribute message bladelet is a high resource user.
- Slow endpoint and high stress—These conditions may lengthen the time for network connections resulting in multiple socket use.

The next table identifies typical low buffer and memory scenarios.

*Table 3-2 Graceful Handling Scenarios: Low Buffer and Memory Conditions*

| Condition | HTTP Client Behavior |
|---|---|
| **Low Buffer** | |
| Because the buffer is too low, an AON node cannot read the headers of a Request message. | The HTTP client connections is closed. The client does not see the error code. However, it does see the connection being reset. |

*Table 3-2 Graceful Handling Scenarios: Low Buffer and Memory Conditions (continued)*

| Condition | HTTP Client Behavior |
|---|---|
| Because the buffer is too low, an AON node can read the headers but not the message of a of a Request message. | Response Message: Message size exceeded system limit. The client does not see the error code. However, it does see the connection being reset. |
| An AON node is in an Out of Buffer condition when it sends a Response message. It cannot read the headers of the Response message. | The HTTP client connections is closed with an error code. |
| An AON node is in an Out of Buffer condition when it sends a Response message. It can read the headers but not the message itself. | Response Code: 503<br>Response Message: Service Unavailable. |
| **Low Memory** | |
| An AON node does not have enough memory to process the request of an embedded adapter. | Response Code 500<br>Response Message: Internal Server Error |
| An AON node does not have enough memory to process the response of embedded adapter. | Response Code 500<br>Response Message: Internal Server Error |
| In a two node situation, the client proxy does not have enough memory to handle a request. | The HTTP client connection is closed without any error code. |
| In a two node situation, the client proxy does not have enough memory to handle a response. | Response Code 500<br>Response Message: Internal Server Error |
| In a two node situation, the server proxy does not have enough memory to handle a request. | The HTTP client connection is closed without any error code. |
| In a two node situation, the sending node (server proxy) does not have memory to handle a response. | Response Code 500<br>Response Message: Internal Server Error |

For more information, see Available Memory Level Trigger Points for Graceful Handling, page 3-40, Registering and Migrating Sources, page 3-41 and Implementing Graceful Handling in Standalone Custom Adapters, page 3-42.

## Available Memory Level Trigger Points for Graceful Handling

Graceful Handling mechanisms generally give precedence to messages that are already in the system. New incoming messages are rejected at the node when resources such as available memory are limited. Graceful Handling also gives precedence to response messages over requests.

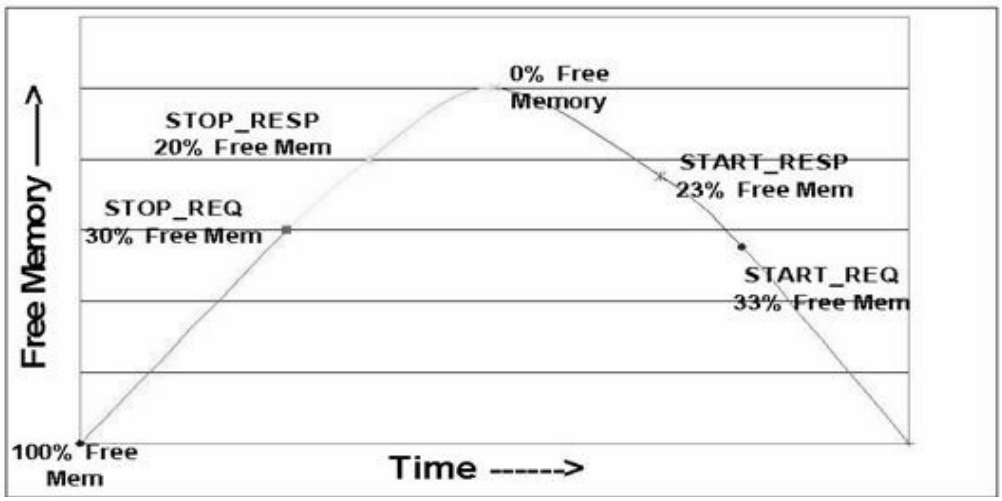
Based on these considerations, AON has four available memory thresholds that act as triggers turning Graceful Handling processes on and off:

- Stop_Req (30% free memory)—Stops the AON node from generating request messages when the free memory space is down to 30%.
- Stop_Resp (20% free memory)—Stops the node from accepting response messages when free memory space is down to 20%.
- Start_Resp (23% free memory)—Enables the node to accept response messages when free memory space is up to 23%.

- Start_Req (33% free memory)—Enables the node to accept request messages when free memory space is up to 33%.

These thresholds (defined in aonsFactoryDefaults.properties) are represented in the next figure.

*Figure 3-17 Graceful Handling Request and Response Thresholds*




For more information, see Registering and Migrating Sources, page 3-41 and Implementing Graceful Handling in Standalone Custom Adapters, page 3-42.

## Registering and Migrating Sources

At startup, adapters register their sources (including unordered or unreliable sources) with AON. In return, AON sends a dispatcher (IGroup) to the adapter. From that point on, the adapter uses the dispatcher to send messages to AON.

If the adapter is running on a node that is a virtual cluster of multiple blades, the message sources are migrated to other blades when the current blade reaches a memory Stop level. The decision to migrate or not depends on the source's MessageType attribute. There are four groups of sources:

- Ordered and Pinned Source—An ordered source that is pinned to a blade after it is assigned to the blade. It can migrate to another blade only if the adapter permits it.
- Ordered and Unpinned Source—An ordered source that can migrate to another blade after the adapter has yielded the source.
- Unordered and Pinned—An unordered source that is pinned to a blade after it is assigned to the blade. It can migrate to another blade only if the adapter permits it.
- Unordered and Unpinned—A source that is shared by all blades.

For more information, see Available Memory Level Trigger Points for Graceful Handling, page 3-40, and Implementing Graceful Handling in Standalone Custom Adapters, page 3-42. Also, see Message Delivery Semantics, page 3-74.

## Implementing Graceful Handling in Embedded Custom Adapters

To make Graceful Handling available, embedded custom adapters must:

- register all sources (Ordered and Unordered)

  For more information, see Registering and Migrating Sources, page 3-41.

- implement the following AttributeValue and AbstractSourceCallback methods:
  - public AttributeValue getMessageType()

    See AttributeValue, page 3-84.
  - public AttributeValue getAffinity()

    See AttributeValue, page 3-84.
  - public abstract void mustYield() throws AdapterException

    See AbstractSourceCallback, page 3-79.

For more about custom adapters, see Developing an Embedded or Standalone Custom Adapter, page 3-3.

# Graceful Handling and Standalone Adapters

Using the software provided in the AON adapter SDK, you can make Graceful Handling mechanisms operate for standalone adapters. For directions, see Implementing Graceful Handling in Standalone Custom Adapters, page 3-42.

The same resource limiting conditions that trigger Graceful Handling for embedded adapters also apply to standalone adapters. For details, see Available Memory Level Trigger Points for Graceful Handling, page 3-40.

To learn about Graceful Handling and its application to embedded adapters, see Graceful Handling and Embedded Adapters, page 3-39.

## Implementing Graceful Handling in Standalone Custom Adapters

To make Graceful Handling available, standalone custom adapters must:

- register all sources (Ordered and Unordered).

  For more information, see Registering and Migrating Sources, page 3-41.
- implement the following AttributeValue and AbstractSourceCallback methods:
  - public AttributeValue getMessageType()

    See AttributeValue, page 3-84.
  - public AttributeValue getAffinity()

    See AttributeValue, page 3-84.
  - public abstract void mustYield() throws AdapterException

    See AbstractSourceCallback, page 3-79.

For general directions on creating custom adapters, see Developing an Embedded or Standalone Custom Adapter, page 3-3.

# Developing MQ Adapters

The AON MQ Adapter enables AON to process Message Queue (MQ) messages. An IBM standard, Websphere MQ establishes a common interface for program-to-program communications across networks that include dissimilar components and communications protocols.

This document introduces the adapter. It explains how to use the AON Development Studio (ADS) and AON Management Console (AMC) to package, upload to AON, configure various types of MQ adapters.



For additional information about AON adapter development and configuration, see the "AON Programming Guide."

# Overview

The AON MQ adapter for the IBM Websphere MQ runs on top of the AON runtime as a standalone adapter. You can develop MQ Adapters for a single node, two nodes that use the same queue manager, two nodes that use different queue managers, or a multiblade virtual cluster. Generally, an MQ Adapter provides inbound and outbound message processing.

Inbound, the adapter receives messages from queues and queue managers. In response, the MQ adapter generates AON messages corresponding to the received MQ messages. Outbound, the MQ adapter sends message received from AON to an MQ queue.

In summary, the MQ adapter has the following features:

- Receives messages from multiple queues and multiple queue managers

  The inbound adapter (feature) can receive messages from local queues. Multiple queues can be configured for each adapter. Each queue can belong to a different queue manager. The adapter gets the message under synchronization point control and sends messages to AON. If AON responds (indicating successful message delivery), the adapter commits the message.

  In addition to content, messages produced by the inbound adapter also contain the following meta-information:

  - Message context—Source URI of the message source queue in the form:

    mq:///<queue manager name>/<queue name>

  - Message header—Contains all message headers of the MQ message.

- Sends messages to multiple queues and to multiple queue managers.

  The outbound adapter (feature) puts messages (received from AON) into a specified destination based on destination information provided in the AON message destination URI.

- Handles datagram and request-reply messages. Request-reply messages are put into the sender's reply-to queue.

  Depending on the message type, the outbound adapter feature configures the message header to indicate the destination reply queue.

- Preserves message order within a queue.

  The adapter sends messages (from a single queue) to AON in the order that they are received into the queue.

- Batches messages from a single queue.

The inbound adapter (feature) can batch messages using a configurable parameter. The adapter batching processes enables multiple messages (from a single queue) to be received as part of a single synchronization point, sent to AON, and committed as a single synchronization point.

- Supports dynamic queues.

  The MQ adapter can handle reply message deliveries to dynamic reply-to queues. In addition to AON reply queues, MQ adapters can also send messages to dynamically-created client reply-to queues. The message transfer process (via AON) is summarized below:

  - MQ clients puts an MQ message in the "Request1" queue and specifies that the response is to arrive at the "Response1" queue.
  - MQ adapter picks up the message from the "Request1" queue.
  - MQ adapter changes the Reply-To queue name to "Response 2" and deposits the message in the "Request2" queue.
  - MQ server/endpoint picks up the message from "Request2" queue and deposits the reply in "Response2" queue.
  - MQ adapter picks up the message from the "Response2" queue and deposits the message in the "Response1" queue.

  **Note** The queue names in the bullets above ("Request1," "Response1," "Request2," and "Response2") are only used for this summary. You may assign the more convenient names that similarly distinguish between reply and response queues.

  In this process, the MQ Adapter checks to determine if the queue referred to is present in the configuration, if not the adapter treats the queue as a one time dynamic queue.

  The MQ adapter opens a new MQ connection to the queue for each response message. The connection is closed after the adapter deposits the message in the dynamic reply-to queue.

  Since the connection is opened-closed for each message, only a batch size of "1" is supported for exactly-once transaction semantics.

  Limitations:

  - The MQ adapter expects that the client reply-to queues already exist. The clients should have created them before sending messages to the MQ adapter.
  - The MQ adapter does not create the queues, if they are not present. Accordingly, they are not defined in MQ adapter policies.
  - The model queue created in MQ should have the "Shareability" property set to "Shareable" so that access to this queue can be shared.

- Issues notifications of message delivery failures.

  The adapter sends out failure notifications as failure queues (DLQ or user-defined) entries and reports.

- Provides exactly-once semantics for message delivery.

  The inbound adapter (feature) enables messages to be delivered at least once. It gets messages under synchronization point control. After it receives success notification from AON, the adapter commits messages, removing them from a queue. The adapter assumes that AON provides reliable delivery of outbound messages.

- Authenticates to the queue manager using custom authentication mechanisms.

Both the inbound and outbound adapter features support authentication to the MQ server using custom authentication mechanisms based on MQ security exits. To turn on this feature, the user provides the name of the Java class that implements the interface MQSecurityExit.

Although the adapter can be configured to support any authentication scheme (for example, Kerberos), it does not provide any built-in implementation of security exits. These must be provided based on server-side authentication implementation.

# Setting Up MQ Adapter Monitoring Tools

You can use the MQ Visual Editor to monitor MQ Adapter activity. The following sections summarize the setup up steps for these tools.

- Downloading and Configuring MQ Visual Edit, page 3-45

## Downloading and Configuring MQ Visual Edit

You should also configure MQ Visual Edit as in the following procedure:

**Step 1** Download MQ Visual Edit.

Download this tool from http://www.caplitalware.biz/mqve_overview.html.

**Step 2** Configure the tool.

See the sample windows at: http://www.caplitalware.biz/mqve_screenshots.html.

# Developing the MQ Adapter for a Single Node

Developing the MQ Adapter for one node, you use the AMC to upload, register, configure, deploy, and validate the adapter. See the following sections:

- Uploading, Registering, and Turning On the MQ Adapter for One Node, page 3-45
- Configuring the MQ Adapter for One Node, page 3-48
- Deploying the MQ Adapter for One Node, page 3-54
- Validating the MQ Adapter for One Node, page 3-55

## Uploading, Registering, and Turning On the MQ Adapter for One Node

After the MQ adapter and adapter extension are packaged, you upload, register and turn on the adapter. Follow the steps listed below.

**Step 1** Upload the adapter package to the AMC.

**a.** Using the AMC, select **Admin > Extensions > Adapter Packages > Upload**.

An upload dialog appears.

*Figure 3-18 Selecting an Adapter Package to Upload*


b. Using the **Browse** button, locate the recently prepared package and click **Upload**.

AMC lists the newly uploaded package.

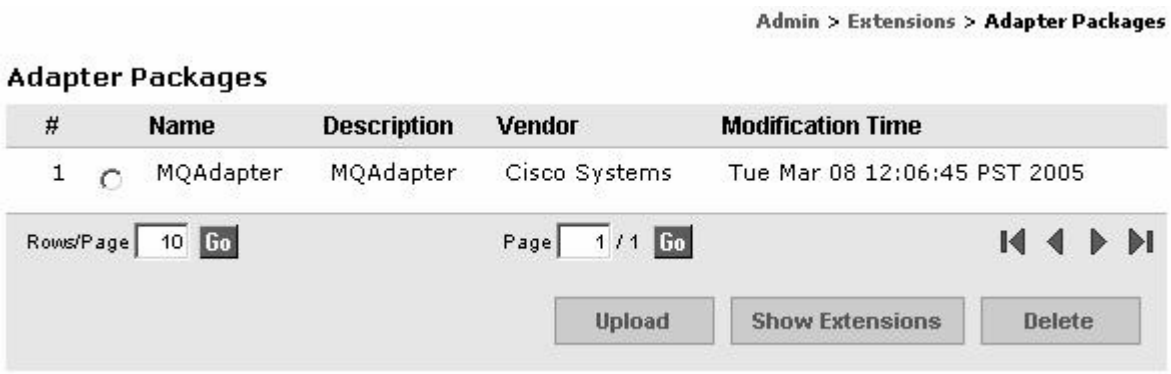
*Figure 3-19 Uploading and Registering the Package*


**Step 2** Register the package with AMC.

a. Click **Register**.

The newly registered package is listed in AMC and ready for use.

*Figure 3-20 Registering the Package*




**Step 3** Upload the MQ Adapter Extension package to the AMC.

a. Using the AMC, select **Admin > Extensions > Adapter Extension Packages > Upload**.

An upload dialog appears.

*Figure 3-21 Selecting an Adapter Extension Package to Upload*


b. Using the **Browse** button, locate the recently prepared adapter extension package and click **Upload**.

AMC lists the newly uploaded package.

*Figure 3-22 Uploading and Registering a Package*


**Step 4** Register the package with AMC.

a. Click **Register**.

The newly registered adapter extension package is listed in AMC and ready for use.

**Step 5** Turn on the adapter.

a. Select **Properties > Adapter > Global.**

b. Select the MQ Adapter row and click **Edit**.

The Adapter Registry: Edit Policy screen appears.

c. Set Is Active to **True**.

d. Click **Submit**.

This action turns on the adapter. The MQAdapter: Global Properties screen appears. Now, see Configuring the MQ Adapter for One Node, page 3-48.

## Configuring the MQ Adapter for One Node

Use the AMC to configure the adapter. As the steps below indicate, you should configure MQ adapter components in the following sequence:

1. Outbound Configuration
   - MQ Outbound Queues
   - MQ Outbound Queue Managers
2. Inbound Configuration
   - MQ Inbound Queues
   - MQ Inbound Queue Managers
3. MQ Adapter
4. MQ Adapter Extension

**Note** This same sequence should be followed to configure MQ adapters in other environments (for example, two nodes with different queue managers).

Follow the steps listed below.

**Step 1** Set the outbound configuration.

a. Setup the MQ outbound queues (DEFAULT_Q).

- Select **Properties > Adapter > Global**.

  The MQAdapter Global Properties screen appears.
- Select **MQ OutboundQueues**.and click **Import.**

  The MQOutboundQueues: Add New Property Set screen appears.
- Fill in the appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the MQOutboundQueue. For example, DEFAULT_Q. |
| OutboundQueueName | MQ queue manager name that exists on the MQ server.<br>Set the name to DEFAULT_Q. |
| OutboundIsDefaultQueue | Possible values = True or false (default). Set to True. In this case, when an incoming AON message does not have a valid destination, it will be sent to this queue. |

- Click **Submit**.

b. Setup the MQ outbound queues (Final_Q).

- Select **Properties > Adapter > Global**.
- Select **MQOutboundQueueManager** and click **Import.**

  The MQOutboundQueues: Edit Property Set screen appears.

- Change the appropriate field values.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the MQOutboundQueue Manager.<br>Set to FINAL_Q. |
| OutboundQueueName | Name of the outbound queue.<br>Set to FINAL_Q. |
| OutboundIsDefaultQueue | True or false. Default = false.<br>Set to True. |

- Click **Submit**.

**c.** Verify all outbound queues.

- Select **Properties > Adapter > Global**.

  The MQAdapter: Global Properties screen appears.

- Verify the Default_Q and Final_Q rows.

  If changes are necessary, click Edit. To add another row, click New. To remove a row, click Delete.

**d.** Setup the MQ outbound queue manager.

- Select **Properties > Adapter > Global**.

  The MQAdapter: Global Properties screen appears.

- Select **MQOutboundQueueManager** and click **Import.**

  The MQOutboundQueueManagers: Add New Property Set screen appears.

- Fill in the appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the outbound queue manager. For example, QM1. |
| OutboundQueueManagerName | MQ queue manager name that exists on the MQ server. For example, QM1. |
| OutboundHostName | Host name or IP address of the MQ server. For example, 172.22.51.112. |
| OutboundPort | Port number on which the MQ server channel has been started. For example, 1414. |
| OutboundChannelName | MQ channel to which the adapter will connect. For example, QM1.Channel. |

| Field | Definition |
|---|---|
| OutboundTransactionQueueName | Name of the transaction queue that exists on this outbound queue manager. In addition to request-reply processing, this queue will be used for persistence to achieve exactly-once semantics. For example, Transaction_Q. |
| OutboundQueuesName | List of all outbound queues on this queue manager. The outbound adapter will send messages to the queue name that is selected. You click Edit List to view and change the queues. See the next substep. |

- Click **Submit**.

**e.** Select all outbound queues.

- Select **Properties > Adapter > Global > Edit Properties > New > Edit List**.

  The Select List Items for: OutboundQueuesName screen appears.

- Check the two queues (Final_Q and Default_Q) and click **Save**.

**f.** Verify the outbound configuration.

- Select **Properties > Adapter > Global > Properties > Edit Properties**.

  The MQOutboundQueueManagers: Add New Property Set appears.

- Verify the information (for example, OutboundHostName) and click **Submit**.

  The outbound configuration setup is complete.

**Step 2** Setup the inbound configuration.

**a.** Setup the inbound queues (Local_Q).

- Select **Properties > Adapter > Global > MQInboundQueues** and click **Import**.

  The MQInboundQueues: Add New Property Set screen appears.

- Fill in the appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue. For example, Local_Q. |
| InboundQueueName | Name of the inbound queue for the connection. For example, Local_Q. |
| InboundBatchSize | Batch size to work and commit on the inbound side. If not specified, defaults to 1. |
| InboundIsReplyQueue | Boolean value indicating whether or not this inbound queue is a reply queue. Possible values True or false (default). Set to False. |
| InboundModelQueueName | Model queue name that will inherit properties for creating dynamic reply queues. Optional. |
| InboundWaitLimit | Time to wait before polling the inbound queue when the message queue contains no messages. Default = 10 (seconds). |

| Field | Definition |
|---|---|
| InboundStopQueueOnError | Boolean value indicates whether or not to stop processing this queue if errors occur. If the value is true, the queue is stopped when a delivery error is received. Possible values: true (default) or false. |
| InboundStopWaitLimit | If the InboundStopQueueOnError is configured, this parameter indicates how long to wait before reprocessing the inbound queue again. Default = 60 (seconds). |
| InboundDeliveryErrorQueueName | In case of errors, the message is delivered to this queue. For example, Deadletter_Q. |
| MessageContent | Identifies the type of message content in the inbound queue. Possible values: Stream Content and XML Content. |
| MessageDelivery | Indicates that the message ordered and reliable or unordered and unreliable. Possible values: Ordered/Reliable (default) and Unordered/Unreliable. |
| DestinationQueueManager | If configured, this queue manager is part of the destination URI. This field indicates that AON should try to send messages from the inbound queue to this Destination Queue Manager. You can click Edit List to access the list. See the next substep below. |

- Click **Submit**.

**b.** Select the destination queue manager.

- Select **Properties > Adapter > Global > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: DestinationQueueManager screen appears.

- Check the destination queue manager and click **Save**.

  The MQInboundQueues: Add New Property Set appears.

**c.** Select the destination queue (Default_Q).

- Select **Properties > Adapter > Global > Properties > Edit Properties > New > Edit List**.

  The Select List items for: DestinationQueue screen appears.

- Check the Default_Q and click **Save**.

  The MQInboundQueues: Add New Property Set appears.

**d.** Verify the inbound queue configuration.

- Review the field information and click **Submit**.

  The MQInboundQueues: Add New Property Set appears.

**e.** Setup the inbound queues (Reply_Q).

- On the MQInboundQueues: Add New Property Set screen change appropriate field values.

The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue. Change to Reply_Q. |
| InboundQueueName | Name of the inbound queue. Change to Reply_Q. |
| InboundBatchSize | Size of the inbound queue batch. Default = 1. |
| InboundIsReplyQueue | Indicates that this is the reply queue. Possible values True or false (default). Set to True. |
| InboundModelQueueName | Name of the inbound model queue. Optional. |
| InboundWaitLimit | Wait time to get messages. Default = 10 (seconds). |
| InboundStopQueueOnError | Indicates that the queue will stop on an error. Possible values True (default) or false. |
| InboundStopWaitLimit | Time to wait before restarting the queue. Default = 60 (seconds). |
| InboundDeliveryErrorQueueName | Name of the delivery error queue. For example, Deadletter_Q. |
| MessageContent | Identifies the type of message content in the inbound queue. Possible values: Stream Content and XML Content. |
| MessageDelivery | Is the message ordered and reliable or unordered and unreliable. Possible values: Ordered/Reliable (default) and Unordered/Unreliable. |
| DestinationQueueManager | Name of the destination queue manager. You can click Edit List to access the list. See the next substep below. |

- Click **Submit**.

**f.** Select the destination queue manager.

- Select **Properties > Adapter > Global > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: DestinationQueueManager screen appears.

- Check the destination queue manager and click **Save**.

  The MQInboundQueues: Add New Property Set appears.

**g.** Select the destination queue (Final_Q).

- Select **Properties > Adapter > Global > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: DestinationQueue screen appears.

- Check the Final_Q box and click **Save**.

  The MQInboundQueues: Add New Property Set appears.

**h.** Verify the inbound queue.

- Review the information in various fields and click **Submit**.
- Select **Properties > Adapter > Global > Properties > Edit Properties > New**.

  The MQInboundQueueManagers: Add New Property Set screen appears.

- Set the appropriate field values.

The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue manager. For example, QM1. |
| InboundQueueManagerName | Name of the inbound queue manager. For example, QM1. |
| InboundHostName | Host name of the queue manager or IP address of the MQ server. For example, 172.22.51.112. Defaults to localhost if not specified. |
| InboundPort | Number of the port to be used to connect to the queue manager. For example, 1414. If not specified, defaults to MQ port. |
| InboundChannelName | Channel name for the queue manager. The adapter connects to this M channel. For example, QM1_Channel. |
| InboundQueuesName | List of currently existing inbound queues. Click Edit List to review. See the next substep below. |

- Click **Submit**.

i. Select all inbound queues.

- Select **Properties > Adapter > Global > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: InboundQueuesName screen appears.

- Check all boxes (for example, Reply_Q and Local_Q) and click **Save**.

  The MQInboundQueueManagers: Add New Property Set screen appears.

j. Verify the information (for example, InboundHostName) and click **Submit.**

k. Setup the MQ adapter configuration.

- Select **Properties > Adapter > Global > Properties > Edit Properties > Edit**.

  The MQAdapter: Edit Property Set screen appears.

- Fill in appropriate field values.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the MQ adapter manager. For example, default. |
| InboundPollingInterval | Interval between polling calls. Default = 10 (seconds). |
| boundedMapSize | Upper bound (number) for in-memory persistence cache. This is the maximum number of cache entries that can be stored in memory. The in-memory cache is searched first for lookup values. When a value is not found or when the in-memory caches reaches this bound size, the system conducts an external lookup. Default = 10000. |

| Field | Definition |
|---|---|
| Language | Language associated with the resource bundle. For example, en = English. Optional. |
| Country | Country associated with the resource bundle. For example, us = United States. Optional. |
| Variant | Variant associated with the resource bundle. Optional. |
| InboundQueueManagers | Names of inbound queue managers that will receive messages. You can click Edit List to access the list. See the substep below. |
| OutboundQueueManagers | Names of the outbound queue managers that will send out messages. You can click Edit List to access the list. See the substep below. |

- Click **Submit**.

l. Select the inbound queue manager.

- Select **Properties > Adapter > Global > Properties > Edit Properties > Edit** > **Edit List**.

  The Select List items for: InboundQueueManagers screen appears.

- Check the box for the queue manager (for example, QM1) and click **Save**.

m. Select the outbound queue manager.

- Select **Properties > Adapter > Global > Properties > Edit Properties > Edit > Edit List**.

  The Select List Items for: OutboundQueueManagers screen appears.

- Check the box for the queue manager (for example, QM1) and click **Save**.

You have completed the inbound configuration. Now, see Deploying the MQ Adapter for One Node, page 3-54.

## Deploying the MQ Adapter for One Node

After the MQ adapter is uploaded and registered, you can deploy it. Follow the steps listed below.

**Step 1** Stage the MQ adapter.

a. Using AMC, select **Deploy > Deployment Requests > Manage Staging**.

The resulting screen lists the current deployment requests.

*Figure 3-23 Selecting a Deployment Request*


**b.** Select a deployment request and click **Stage**.

**c.** Follow the on-screen instructions to stage the adapter.

For more information, see the section "Developing Standalone Adapters" in the "AON Programming Guide."

**Step 2** Deploy the MQ adapter.

**a.** Select **Deploy > Deployment Requests > Manage Deployment.**

The resulting screen shows the staged deployment requests.

*Figure 3-24 Selecting a Staged Deployment Request*


**b.** Select a staged deployment request and click **Deploy**.

For more information, see the section "Developing Standalone Adapters" in the "AON Programming Guide." In addition, see Validating the MQ Adapter for One Node, page 3-55.

## Validating the MQ Adapter for One Node

After you have packaged, uploaded, registered, staged, and deployed the MQ adapter you can validate it. In this process, you check it to determine whether or not it is properly configured and able to transfer messages successfully. Follow the steps listed below.

**Step 1** Start the MQ adapter.

**Step 2** Check log messages to determine whether the adapter is connecting to the queues and polling them on the inbound side for messages. A sample log file could look like the following:

QueueConfig #of messages on queue QM1.LOCAL_Q1 = 0

**Step 3** Go to the MQ server and check for connections on a given inbound queue.

**a.** Find the MQ queues in the MQ Explorer, right-click and select **Status**.

The display should show that a live connection from AON (in the form of a Java client) is being made to the queue.

**Step 4** Put a message into the inbound queue.

**a.** Using the MQ Explorer, right-click on the queue and press **Put message**.

**b.** Enter the message data and press **Return**.

This action generates a datagram message. The MQ adapter should be able to pick up this message and transfer it to the outbound side through AON.

**Step 5** Check the AON log to determine how the message is being transferred.

**Step 6** Finally, look at the outbound queue to determine if the message is being deposited in the outbound queue or not.

# Developing the MQ Adapter for Two Nodes Using the Same Queue Manager

After you use the ADS to package the MQ adapter for two nodes using the same queue manager, you switch to the AMC to complete the setup. You must upload, register, and configure the adapter. You should also set up a next hop domain. These tasks are described in the following sections:

- Uploading, Registering, and Turning On the MQ Adapter for Two Nodes Using the Same Queue Manager, page 3-56
- Configuring the MQ Adapter for Two Nodes Using the Same Queue Manager, page 3-57
- Setting Up a Next Hop Domain, page 3-66

## Uploading, Registering, and Turning On the MQ Adapter for Two Nodes Using the Same Queue Manager

You use the AMC to upload, register, and turn on the new MQ adapter. Follow the steps listed below.

**Step 1** Upload the adapter package.

**a.** Using the AMC, select **Admin > Adapter Packages**.

The Upload Adapter Package screen appears.

**b.** Click the Browse button and locate the package to be uploaded.

**c.** Click **Upload**.

**Step 2** Register the adapter package with AMC.

The screen has changed to Upload and Register Adapter Package.

**a.** Select **Admin > Adapter Packages > Register**.

**Step 3** Upload the MQ Adapter Extension package to the AMC.

**a.** Using the AMC, select **Admin > Extensions > Adapter Extension Packages > Upload**.

An upload dialog appears.

**b.** Using the **Browse** button, locate the recently prepared adapter extension package and click **Upload**.

AMC lists the newly uploaded package.

**Step 4** Turn On the MQ adapter.

a. Select **Properties > Adapter > Node**.

The All AON Nodes screen appears.

b. Select one node and click **Edit Properties**.

The Adapter Registry: Node Properties for *nodename* screen appears.

c. Select the MQ Adapter row and click **Edit**.

The Adapter Registry: Edit Property Set screen appears.

d. Set Is Active to **True**.

e. On the All AON Nodes screen, select the other node and click **Edit Properties**.

The Adapter Registry: Node Properties for *nodename* screen appears.

f. Select the MQ Adapter row and click **Edit**.

The Adapter Registry: Edit Property Set screen appears.

g. Set Is Active to **True**.

The adapter is turned on for both nodes. Now, see Configuring the MQ Adapter for Two Nodes Using the Same Queue Manager, page 3-57.

## Configuring the MQ Adapter for Two Nodes Using the Same Queue Manager

Using the AMC, follow the steps listed below to configure the MQ adapter for two nodes that use the same queue manager. Follow the steps listed below.

**Step 1** Configure the MQ adapter for one node.

a. Setup MQ Outbound Queues for one node.

- On the All AON Nodes screen, select one node and click **Edit Properties**.

  The Adapter Registry: Node Properties for *nodename* screen appears.

- Select the MQAdapter row and click **Properties**.

  The MQAdapter: Node Properties for *nodename* screen appears.

- Select the MQOutboundQueues category and click **New**.

  The MQOutboundQueues: Add New Property Set screen appears. The Name and OutboundQueueName fields are already set to Default_Q.

- Set OutboundIsDefaultQueue to **True** and click **Submit**.

- On the MQAdapter: Node Properties for *nodename* screen, select MQOutboundQueues and click **New**.

  The MQOutboundQueues: Add New Property Set screen appears. The Name and OutboundQueueName fields are already set to Final_Q.

- Set OutboundIsDefaultQueue to **False** and click **Submit**.

b. Verify the outbound queues.

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties**.

The MQAdapter: Node Properties for *nodename* screen appears.

- Verify the Default_Q and Final_Q rows.

  If changes are necessary, click Edit. To add another row, click New. To remove a row, click Delete.

**c.** Setup MQ Outbound Queue Managers for one node.

- On the MQAdapter: Node Properties for *nodename* screen select MQOutboundQueueManagers and click **New**.

  The MQOutboundQueueManagers: Add New Property Set screen appears.

- Fill in appropriate fields and click **Submit**.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the outbound queue manager. For example, QM1. |
| OutboundQueueManagerName | MQ queue manager name that exists on the MQ server. For example, QM1. |
| OutboundHostName | Host name or IP address of the MQ server. For example, 172.22.51.112. |
| OutboundPort | Port number on which the MQ server channel has been started. For example, 1414. |
| OutboundChannelName | MQ channel to which the adapter will connect. For example, QM1.Channel. |
| OutboundTransactionQueueName | Name of the transaction queue that exists on this outbound queue manager. In addition to request-reply processing, this queue will be used for persistence to achieve exactly-once semantics. For example, Transaction_Q. |
| OutboundQueuesName | List of all outbound queues on this queue manager. The outbound adapter will send messages to the queue name that is selected. You click Edit List to view and change the queues. See the next substep. |

- Click **Submit**.
- Select **Properties > Adapter > Global > Edit Properties > New > Edit List**.

  The Select List Items for: OutboundQueuesName screen appears.

- Check the two queues (Final_Q and Default_Q) and click **Save**.

**d.** Verify and submit the settings.

- Select **Properties > Adapter > Node > Properties > Edit Properties**.

  The MQOutboundQueueManagers: Add New Property Set appears.

- Verify the information (for example, OutboundHostName) and click **Submit**.

**e.** Setup MQ Inbound Queues for one node.

- On the MQAdapter: Node Properties for *nodename* screen select MQInboundQueues and click **New**.

The MQInboundQueues: Add New Property Set screen appears.

- Fill in the appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue. For example, Local_Q. |
| InboundQueueName | Name of the inbound queue for the connection. For example, Local_Q. |
| InboundBatchSize | Batch size to work and commit on the inbound side. If not specified, defaults to 1. |
| InboundIsReplyQueue | Boolean value indicating whether or not this inbound queue is a reply queue. Possible values True or false (default). Set to False. |
| InboundModelQueueName | Model queue name that will inherit properties for creating dynamic reply queues. Optional. |
| InboundWaitLimit | Time to wait before polling the inbound queue when the message queue contains no messages. Default = 10 (seconds). |
| InboundStopQueueOnError | Boolean value indicates whether or not to stop processing this queue if errors occur. If the value is true, the queue is stopped when a delivery error is received. Possible values: true (default) or false. |
| InboundStopWaitLimit | If the InboundStopQueueOnError is configured, this parameter indicates how long to wait before reprocessing the inbound queue again. Default = 60 (seconds). |
| InboundDeliveryErrorQueueName | In case of errors, the message is delivered to this queue. For example, Deadletter_Q. |
| MessageContent | Identifies the type of message content in the inbound queue. Possible values: Stream Content and XML Content. |
| MessageDelivery | Is the message ordered and reliable or unordered and unreliable. Possible values: Ordered/Reliable (default) and Unordered/Unreliable. |
| DestinationQueueManager | If configured, this queue manager is part of the destination URI. This field indicates that AON should try to send messages from the inbound queue to this Destination Queue Manager. You can click Edit List to access the list. See the next substep below. |

- Click **Submit**.

**f.** Select the destination queue manager.

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: DestinationQueueManager screen appears.

- Check the destination queue manager (for example, QM1) and click **Save**.

g. Select the destination queue (Default_Q).

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > New > Edit List**.

  The Select List items for: DestinationQueue screen appears.

- Check the Default_Q and click **Save**.

h. Verify the inbound queue configuration.

- On the MQInboundQueues: Add New Property Set screen, verify the information and click **Submit**.

  The MQInboundQueues: Add New Property Set appears.

i. Setup the inbound queue manager (QM1).

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties**.

  The MQAdapter: Node Properties for *nodename* screen appears.

- Select MQInboundQueueManagers and click **New**.

  The MQInboundQueueManagers: Add New Property Set appears.

- Fill in appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue manager. For example, QM1. |
| InboundQueueManagerName | Name of the inbound queue manager. For example, QM1. |
| InboundHostName | Host name of the queue manager or IP address of the MQ server. For example, 172.22.51.112. Defaults to localhost if not specified. |
| InboundPort | Number of the port to be used to connect to the queue manager. For example, 1414. If not specified, defaults to MQ port. |
| InboundChannelName | Channel name for the queue manager. The adapter connects to this M channel. For example, QM1_Channel. |
| InboundQueuesName | List of currently existing inbound queues. Click Edit List to review. See the next substep below. |

- Click **Submit.**

j. Select all inbound queues.

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: InboundQueuesName screen appears.

- Check the box (for example, Local_Q) and click **Save**.

  The MQInboundQueueManagers: Add New Property Set screen appears.

k. Verify the information (for example, InboundHostName) and click **Submit.**

l. Setup MQ Adapter for one node.

- On the MQAdapter: Node Properties for *nodename* screen select MQAdapter and click **New**.
- When a warning appears the selected property set has global scope click **OK**.

  The MQAdapter: Edit Property Set screen appears.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the MQ adapter manager. For example, default. |
| InboundPollingInterval | Interval between polling calls. Default = 10 (seconds). |
| boundedMapSize | Upper bound (number) for in-memory persistence cache. This is the maximum number of cache entries that can be stored in memory. The in-memory cache is searched first for lookup values. When a value is not found or when the in-memory caches reaches this bound size, the system conducts an external lookup. Default = 10000. |
| Language | Language associated with the resource bundle. For example, en = English. Optional. |
| Country | Country associated with the resource bundle. For example, us = United States. Optional. |
| Variant | Variant associated with the resource bundle. Optional. |
| InboundQueueManagers | Names of inbound queue managers that will receive messages. You can click Edit List to access the list. See the substep below. |
| OutboundQueueManagers | Names of the outbound queue managers that will send out messages. You can click Edit List to access the list. See the substep below. |

- For InboundQueueManagers, click **Edit List**.

  The Select List Items for: InboundQueueManagers appears.
- Check the listed manager (for example, QM1) and click **Save**.

  The MQAdapter: Edit Property Set screen appears.
- For OutboundQueueManager, click **Edit List**.
- Check the listed manager (for example, QM1) and click **Save**.

  The MQAdapter: Edit Property Set screen appears.
- Click **Submit**.

  The inbound configuration for one node is complete.

**Step 2** Configure the MQ adapter for the other node.

a. Setup MQ Outbound Queues for a second node.

- On the All AON Nodes screen, select the second node and click **Edit Properties**.

  The Adapter Registry: Node Properties for *nodename* screen appears.
- Select the MQAdapter row and click **Properties**.

The MQAdapter: Node Properties for *nodename* screen appears.

- Select MQOutboundQueues and click **Import**.

  The MQOutboundQueues: Add New Property Set screen appears. The Name and OutboundQueueName fields are already set to Default_Q.

- Set OutboundIsDefaultQueue to **True** and click **Submit**.

  The MQOutboundQueues: Add New Property Set screen appears. The Name and OutboundQueueName fields are already set to Final_Q.

- Set OutboundIsDefaultQueue to **False** and click **Submit**.

**b.** Verify the outbound queues.

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties**.

  The MQAdapter: Node Properties for *nodename* screen appears.

- Verify the Default_Q and Final_Q rows.

  If changes are necessary, click Edit. To add another row, click New. To remove a row, click Delete.

**c.** Setup MQ Outbound Queue Managers for the second node.

- On the MQAdapter: Node Properties for *nodename* screen select MQOutboundQueueManagers and click **New**.

  The MQOutboundQueueManagers: Add New Property Set screen appears.

- Fill in appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the outbound queue manager. For example, QM1. |
| OutboundQueueManagerName | MQ queue manager name that exists on the MQ server. For example, QM1. |
| OutboundHostName | Host name or IP address of the MQ server. For example, 172.22.51.112. |
| OutboundPort | Port number on which the MQ server channel has been started. For example, 1414. |
| OutboundChannelName | MQ channel to which the adapter will connect. For example, QM1.Channel. |
| OutboundTransactionQueueName | Name of the transaction queue that exists on this outbound queue manager. In addition to request-reply processing, this queue will be used for persistence to achieve exactly-once semantics. For example, Transaction_Q. |
| OutboundQueuesName | List of all outbound queues on this queue manager. The outbound adapter will send messages to the queue name that is selected. You click Edit List to view and change the queues. See the next substep. |

- Click **Submit**.

- Select **Properties > Adapter > Node > Edit Properties > New > Edit List**.

  The Select List Items for: OutboundQueuesName screen appears.

- Check the two queues (Final_Q and Default_Q) and click **Save**.

**d.** Verify and submit the settings.

- On the MQOutboundQueueManagers: Add New Property Set screen, verify the information (for example, OutboundHostName) and click **Submit**.

**e.** Setup MQ Inbound Queues (Local_Q2) for one node.

- On the MQAdapter: Node Properties for *nodename* screen select MQInboundQueues and click **New**.

  The MQInboundQueues: Add New Property Set screen appears.

- Fill in the appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue. For example, Local_Q2. |
| InboundQueueName | Name of the inbound queue for the connection. For example, Local_Q2. |
| InboundBatchSize | Batch size to work and commit on the inbound side. If not specified, defaults to 1. |
| InboundIsReplyQueue | Boolean value indicating whether or not this inbound queue is a reply queue. Possible values True or false (default). Set to False. |
| InboundModelQueueName | Model queue name that will inherit properties for creating dynamic reply queues. Optional. |
| InboundWaitLimit | Time to wait before polling the inbound queue when the message queue contains no messages. Default = 10 (seconds). |
| InboundStopQueueOnError | Boolean value indicates whether or not to stop processing this queue if errors occur. If the value is true, the queue is stopped when a delivery error is received. Possible values: true (default) or false. |
| InboundStopWaitLimit | If the InboundStopQueueOnError is configured, this parameter indicates how long to wait before reprocessing the inbound queue again. Default = 60 (seconds). |
| InboundDeliveryErrorQueueName | In case of errors, the message is delivered to this queue. For example, Deadletter_Q. |
| MessageContent | Identifies the type of message content in the inbound queue. Possible values: Stream Content and XML Content. |

| Field | Definition |
|---|---|
| MessageDelivery | Is the message ordered and reliable or unordered and unreliable. Possible values: Ordered/Reliable (default) and Unordered/Unreliable. |
| DestinationQueueManager | If configured, this queue manager is part of the destination URI. This field indicates that AON should try to send messages from the inbound queue to this Destination Queue Manager. You can click Edit List to access the list. See the next substep below. |

f. Select the destination queue manager.

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: DestinationQueueManager screen appears.

- Check the destination queue manager (for example, QM1) and click **Save**.

g. Select the destination queue (Default_Q).

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: DestinationQueue screen appears.

- Check the Default_Q and click **Save**.

h. Verify and submit the inbound queue configuration (Reply_Q).

- On the MQInboundQueues: Add New Property Set screen, set InboundIsReplyQueue to **True**.

  The MQInboundQueues: Add New Property Set appears.

i. Select the destination queue manager (QM1)

- On the Select List Items for: DestinationQueueManager check the listed manager (QM1) and click **Save**.

j. Select the destination queue manager (Final_Q)

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > Edit List**.

  The Select List Items for: DestinationQueue screen appears.

- Check the Final_Q and click **Save**.

k. Verify and submit the inbound queue configuration (Reply_Q)

- On the MQInboundQueues: Add New Property Set screen, verify the information and click **Submit**.

l. Setup the inbound queue manager (QM1).

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties**.

  The MQAdapter: Node Properties for *nodename* screen appears.

- Select MQInboundQueueManagers and click **New**.

  The MQInboundQueueManagers: Add New Property Set appears.

- Fill in appropriate fields.

The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the inbound queue manager. For example, QM1. |
| InboundQueueManagerName | Name of the inbound queue manager. For example, QM1. |
| InboundHostName | Host name of the queue manager or IP address of the MQ server. For example, 172.22.51.112. Defaults to localhost if not specified. |
| InboundPort | Number of the port to be used to connect to the queue manager. For example, 1414. If not specified, defaults to MQ port. |
| InboundChannelName | Channel name for the queue manager. The adapter connects to this M channel. For example, QM1_Channel. |
| InboundQueuesName | List of currently existing inbound queues. Click Edit List to review. See the next substep below. |

**m.** Select all inbound queues.

- Select **Properties > Adapter > Node > Edit Properties > Properties > Edit Properties > New > Edit List**.

  The Select List Items for: InboundQueuesName screen appears.

- Check both boxes (for example, Reply_Q and Local_Q2) and click **Save**.

  The MQInboundQueueManagers: Add New Property Set screen appears.

**n.** Verify and submit the information.

- Review the information (for example, InboundHostName) and click **Submit.**

**o.** Setup MQ Adapter for the second node.

- On the MQAdapter: Node Properties for *nodename* screen select MQAdapter and click **New**.
- When a warning appears the selected property set has global scope click **OK**.

  The MQAdapter: Edit Property Set screen appears.

**p.** Select the inbound queue manager (QM1).

- On the MQAdapter: Edit Property set screen fill in appropriate fields.

  The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the MQ adapter manager. For example, default. |
| InboundPollingInterval | Interval between polling calls. Default = 10 (seconds). |

| Field | Definition |
|---|---|
| boundedMapSize | Upper bound (number) for in-memory persistence cache. This is the maximum number of cache entries that can be stored in memory. The in-memory cache is searched first for lookup values. When a value is not found or when the in-memory caches reaches this bound size, the system conducts an external lookup. Default = 10000. |
| Language | Language associated with the resource bundle. For example, en = English. Optional. |
| Country | Country associated with the resource bundle. For example, us = United States. Optional. |
| Variant | Variant associated with the resource bundle. Optional. |
| InboundQueueManagers | Names of inbound queue managers that will receive messages. You can click Edit List to access the list. See the substep below. |
| OutboundQueueManagers | Names of the outbound queue managers that will send out messages. You can click Edit List to access the list. See the substep below. |

- For InboundQueueManagers, click **Edit List**.

  The Select List Items for: InboundQueueManagers appears.
- Check the listed manager (for example, QM1) and click **Save**.

  The MQAdapter: Edit Property Set screen appears.
- For OutboundQueueManager, click **Edit List**.
- Check the listed manager (for example, QM1) and click **Save**.

  The MQAdapter: Edit Property Set screen appears.

**q.** Verify and submit the information.

- Review the information and click **Submit.**

  The second configuration is complete.

## Setting Up a Next Hop Domain

When your MQ adapter setup is planned for two nodes that use the same queue manager, you should use AMC to setup a next hop domain depending on the URI. Follow the steps listed below.

**Step 1** Select Properties > Application > Node.

The All AON Nodes screen appears.

**Step 2** Select one node and click **Edit Properties**.

The Application: Node Properties for **nodename** screen appears.

**Step 3** Select Next Hop Domain and click **New**.

The Next Hop Domain: Add New Property Set screen appears.

**Step 4** Fill in the appropriate fields.

The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Name of the next hop domain. For example, localhost:1414. |
| Address | Address of the next hop. For example, localhost. |
| Port | Next hop port. For example, 5555. |
| Protocol | Protocol. Possible values: HTTP (default) or TCP. |
| Mode | Mode associated with the next hop domain. Possible values: clear (default) or secure. |

**Step 5** Click **Submit**.

The next hop domain is setup.

# MQ Adapter Exceptions, Error Messages, and Solutions

This section provides a list of exceptions, associated error messages, and suggested solutions for problems that may occur as an MQ Adapter is being uploaded, registered, configured, or deployed.

*Table 3-3 MQ Adapter Exceptions, Errors, and Solutions*

| Exception or Error Code | Exception or Error Message | Possible Solutions |
|---|---|---|
| Code 2085 | exception = MQJE001: Completion Code 2, Reason 2085<br>com.ibm.mq.MQException: MQJE001: Completion Code 2, Reason 2085<br>at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1527)<br>at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1579)<br>at com.cisco.aons.adapter.mq.QueueConfig.setupQueues(QueueConfig.java:289) | • Check for a missing or incorrect TRANSACTION Queue Name.<br>• Verify that the Transaction Queue Name in both the AONS Configuration and the Websphere MQ Configuration.<br>• Verify that the case of the Transaction Queue Name (CAPS or small letter).<br>• Verify that the Transaction Queue Name is correct.<br>• Restart the Queue Manager, if the problem still exists. |
| Code 2009/ 6001 | MQJE001: An MQException occurred: Completion Code 2, Reason 2009<br>MQJE016: MQ queue manager closed channel immediately during connect<br>Closure reason = 2009<br>MQJE001: An MQException occurred: Completion Code 2, Reason 2009<br>MQJE016: MQ queue manager closed channel immediately during connect<br>Closure reason = 2009<br>ERROR [MEC-Q-4] mec.adapter.MQAdapter errorCode = 6001 | • Check for a missing or incorrect CHANNEL Name.<br>• Verify the Channel Name in both the AONS Configuration and Websphere MQ Configuration.<br>• Verify that the Channel Queue Name is correct.<br>• Restart the Queue Manager, if the problem still exists. |
| Code 6050 | MEC-Q-4 ERROR adapter.MQAdapter - errorCode = 6050<br>MEC-Q-4 ERROR adapter.MQAdapter - reason = Error getting destination queue | This error is thrown when MQ Adapter is unable to locate the Destination Queue that is set in the URI.<br>• Verify that the Destination Queue exists.<br>• Verify the MQ Adapter Destination Configuration.<br>• Verify the endpoint for setting the proper Destination URI. |

*Table 3-3 MQ Adapter Exceptions, Errors, and Solutions (continued)*

| Exception or Error Code | Exception or Error Message | Possible Solutions |
|---|---|---|
| throwMissingResourceException - MQMessagesBundle | [java] Caused by: java.util.MissingResourceException: Can't find bundle for base name MQMessagesBundle, locale en_US<br>[java] at java.util.ResourceBundle.throwMissingResourceException(ResourceBundle.java:804)<br>[java] at java.util.ResourceBundle.getBundleImpl(ResourceBundle.java:773)<br>[java] at java.util.ResourceBundle.getBundle(ResourceBundle.java:538) | • Verify that the Adapter jar contains the "MQMessagesBundle.properties" file.<br>• Check to be certain that the jar file is correctly built. |
| Code 6004 | ERROR [MEC-Q-4] mec.adapter.MQAdapter errorCode = 6004<br>ERROR [MEC-Q-4] mec.adapter.MQAdapter reason = Failed to find the reply queue info in persistent. key AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA | This error is thrown as reply messages are being processed from the reply queue. Key "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" is all blanks.<br>In this case, something may be wrong with the end-point that reads messages off of outbound queue and puts them in AONS reply queue.<br>The end-point is not putting messages in the reply queue with the correct CorrelationID. |
| Code 5003 | MQJE011: Socket connection attempt refused<br>12-22-2004 22:12:22 DEBUG [main] mq - MQOutboundExceptionHandler.handleAdapterException<br>12-22-2004 22:12:22 ERROR [main] mq - errorCode = 5003<br>12-22-2004 22:12:22 ERROR [main] mq - reason = Error loading persistent store during adapter initialization | • MQ Adapter is unable to connect to the Transaction Queue of the MQ Broker.<br>• Verify that the MQ Broker is running on the specified port.<br>• Verify that the MQ Adapter configuration specifies the port on which the MQ broker is running. |

*Table 3-3 MQ Adapter Exceptions, Errors, and Solutions (continued)*

| Exception or Error Code | Exception or Error Message | Possible Solutions |
|---|---|---|
| Null Domain | main ERROR mec.adapter - Error in initializing adapter MQAdapter com.cisco.aons.adapter.AdapterInitException: Null Domain Name trying to reach at com.cisco.aons.adapter.AdapterDescriptor.getDomainReader(AdapterDescriptor.java:618) at com.cisco.aons.adapter.mq.Configuration.setOutboundQueueManagers(Configuration.java:299) at com.cisco.aons.adapter.mq.Configuration.doInitialize(Configuration.java:75) at com.cisco.aons.adapter.mq.MQStandaloneAdapter.doInitialize(MQStandaloneAdapter.java:57) at com.cisco.aons.adapter.Adapter.initialize(Adapter.java:40) at com.cisco.aons.adapter.StandaloneAdapter.initialize(StandaloneAdapter.java:45) at com.cisco.aons.adapter.AdapterManager.initializeAdapter(AdapterManager.java:536) at com.cisco.aons.adapter.AdapterManager.initializeAdapters(AdapterManager.java:423) at com.cisco.aons.adapter.AdapterManager.doInitialize(AdapterManager.java:241) at com.cisco.aons.adapter.AdapterManager.<init>(AdapterManager.java:182) at com.cisco.aons.adapter.AdapterManager.initialize(AdapterManager.java:218) at com.cisco.aons.core.ExecutionController.initialize(ExecutionController.java:253) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25) at java.lang.reflect.Method.invoke(Method.java:324) at com.cisco.aons.bootstrap.AONSBootStrapper.bootstrapAll(AONSBootStrapper.java:125) at com.cisco.aons.bootstrap.AONSBootStrapper.bootstrap(AONSBootStrapper.java:92) at com.cisco.aons.AONSMain.main(AONSMain.java:39) | This occurs when the Destination Queue Manager/Destination Queue is not specified for each Inbound Queue. |

*Table 3-3 MQ Adapter Exceptions, Errors, and Solutions (continued)*

| Exception or Error Code | Exception or Error Message | Possible Solutions |
|---|---|---|
| ReplyQ Not Acquired – Virtual Cluster | MEC-Q-2 DEBUG adapter.MQAdapter - QueuePeeker: AON.QM:AON.REPLY.Q NOT yet acquired...no need to Peek.<br><br>MEC-Q-2 DEBUG adapter.MQAdapter - QueuePeeker: AON.QM:AON.LOCAL.Q NOT yet acquired...no need to Peek.<br><br>MEC-Q-2 DEBUG adapter.MQAdapter - QueuePeeker: AON.QM:AON.REPLY.Q NOT yet acquired...no need to Peek. | This AONS log entry can be a benign message or an error message, depending on the following determinations.<br><br>In a multiblade-virtual cluster situation, one reply is needed for each blade. If there are two blades, you must configure two reply queues.<br><br>• Verify that the WCCP is setup correctly.<br>• Check the log for each blade to determine if each blade has acquired one reply.<br><br>For example, if you have configured two reply queues with names AON.REPLY.Q and AON.REPLY.Q2, One blade should acquire AON.REPLY.Q and other blade should acquire AON.REPLY.Q2.<br><br>In this case, the above message displayed in the log is benign because blade2 is saying that it did not acquire AON.REPLY.Q (which is acquired by blade1)<br><br>In a single blade situation, the AONS log message is an error message indicating that the MQ Adapter was unable to acquire the Reply Queue. |

*Table 3-3 MQ Adapter Exceptions, Errors, and Solutions (continued)*

| Exception or Error Code | Exception or Error Message | Possible Solutions |
|---|---|---|
| Code 5002 | main DEBUG adapter.MQAdapter - MQInboundExceptionHandler handleAdapterInitException<br>2005-07-01 14:15:18,384 main ERROR adapter.MQAdapter - errorCode = 5002<br>2005-07-01 14:15:18,384 main ERROR adapter.MQAdapter - reason = Error invalid data for attribute OutboundQueueManagerName<br>2005-07-01 14:15:18,384 main ERROR adapter.MQAdapter - exception = InvalidAttributeName Passed<br>com.cisco.aons.util.DomainException:InvalidAttributeName Passed<br>at com.cisco.aons.pm.DomainReaderImpl.getValues(DomainReaderImpl.java:143)<br>at com.cisco.aons.adapter.mq.Configuration.getAttributeValue(Configuration.java:401)<br>at com.cisco.aons.adapter.mq.Configuration.setOutboundQueueManagers(Configuration.java:276)<br>at com.cisco.aons.adapter.mq.Configuration.doInitialize(Configuration.java:75)<br>at com.cisco.aons.core.ExecutionController.initialize(ExecutionController.java:254)<br>at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)<br>at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)<br>at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)<br>at java.lang.reflect.Method.invoke(Unknown Source) | This message appears when AMC creates the wrong property sets in MQAdapter.xml. |

*Table 3-3 MQ Adapter Exceptions, Errors, and Solutions (continued)*

| Exception or Error Code | Exception or Error Message | Possible Solutions |
|---|---|---|
| | at com.cisco.aons.bootstrap.AONSBootStrapper.bootstrapAll(AONSBootStrapper.java:125)<br>at com.cisco.aons.bootstrap.AONSBootStrapper.bootstrap(AONSBootStrapper.java:92)<br>at com.cisco.aons.AONSMain.main(AONSBootStrapper.java:39) | |
| Code AMQ6090 | AMQ6090: WebSphere MQ was unable to display an error message 20006220 | • Verify that you are logged in as the user "mqm"<br>• Export the following variable:<br>export LD_ASSUME_KERNEL=2.4.19<br>• Restart the queue manage |
| Socket Connection Refused - MQJE011 | MQJE011: Socket connection attempt refused | • Verify that the MQ Broker is running on the specified port.<br>• Verify that the MQ Adapter configuration specifies the port on which the MQ broker is running. |
| Code 2033 | MQJE001: Completion Code 2, Reason 2033 com.ibm.mq.MQException: MQJE001: Completion Code 2, Reason 2033at com.ibm.mq.MQQueue.get(MQQueue.java:863) at com.ibm.mq.MQQueue.get(MQQueue.java:1086) at com.cisco.aons.adapter.mq.TestMQ.putReplyMessage(TestMQ.java:225) at com.cisco.aons.adapter.mq.TestMQ.putReplyTest(TestMQ.java:142) at com.cisco.aons.adapter.mq.TestMQ.main(TestMQ.java:57) | This is the error code for NO_MSG_AVAILABLE.<br>If no messages are available in DEFAULT_Q, this error would appear. The server end-point should catch and handle this exception.<br>• MQ sometimes throws this exception if the user is trying to read from a queue and the queue contains no messages. In this case, the server program is trying to read from the default_q, when there are no messages there. The server should just ignore this and try again.<br>• Verify that the server program is not creating a new connection every time. It uses an old connection to get the next message. If a new connection is created every time eventually, the server will run out of connections. |

# Message Delivery Semantics

AON uses Message Delivery Semantics (MDS) to guarantee reliable and/or ordered delivery of messages based on user-defined message types. For more information, see the *AON Installation and Administration Guide.*



## MDS Inbound Processing

For inbound processing, adapters must implement the Adapter, ISourceCallback, and StandaloneMessageReader interfaces. These implementations will use the IAdapterManager, IResourceManager, and IGroup interfaces of AON.

During the startup process, adapters must get a resource manager by the IAdapterManager.getResourceManager() method. Then, adapters register all their sources by one of the IResourceManager.registerSource methods. Adapters must acquire the registered source only when the ISourceCallback.notifyAcquire() method is called. When the StandaloneMessageReader.execute() method is called, adapters must use the IGroup object returned from IResourceManager.registerSource to dispatch messages from the registered source to AON.

Adapters may be requested to yield a source with a call to ISourceCallback.notifyYield(). Adapters may also periodically yield sources without being requested. Adapters must first determine if it is necessary to yield the source by calling IResourceManager.shouldYield(Source). Adapters must then yield the source if necessary and indicate completion by calling IResourceManager.yieldSource(Source). If the adapter receives a shutdown request via Adapter.doShutdownInput(), the adapter must deregister all sources by IResourceManager.deregisterSource(Source).

### Custom Adapter Classes

MDS inbound processing is enabled through the following classes:

- Source

  Source is an abstract class that represents an adapter resource. A resource can be ordered or unordered.

- OrderedSource

  This is an abstract class that represents an adapter resource for messages that require ordered delivery.

- UnorderedSource

  This is an abstract class that represents an adapter resource for messages that do not require ordered delivery.

- IAdapterManager

  This has been modified to include a new method getResourceManager() that returns the resource manager.

- IResourceManager

  This interface controls adapter resources. Adapters can get an instance of this class using IAdapterManager.getResourceManager().

- IGroup

  This interface defines methods for dispatching messages in a group. A group is an inbound collection of messages that originate from the same source. Adapters can get an instance of this class using IResourceManager.registerSource. Adapters should then use the returned IGroup object to dispatch inbound messages from the registered source.

- ISourceCallback

  This interface defines callbacks for adapters to be notified to act on associated sources. Adapters should pass in an object which implements this interface as part of source registration byIResourceManager.registerSource.

# MDS Outbound Processing

For outbound message delivery, adapters must implement the Adapter and IDeliveryGroupDispatcher interfaces. These implementations will use the IDeliveryGroupCallback interface of AON.

Adapters must send messages to destinations when adapters receive a call to IDeliveryGroupDispatcher.sendMessage(IAONSMessage, IMessageHandler). Adapters may then either call IDeliveryGroupCallback.notifyCommit() or IDeliveryGroupCallback.notifyRollback() to commit or rollback the messages in the delivery group, respectively. After the adapter is ready to deliver the next message in the delivery group, the adapter must call IDeliveryGroupCallback.notifyReadyToReceive(IAONSMessage). If it is necessary to assign destinations to other network nodes in the virtual cluster, AON will call IDeliveryGroupDispatcher.releaseDestination() to request adapters to release destinations. Adapters must indicate completion by either calling IDeliveryGroupCallback.notifyReleaseCompleted() or IDeliveryGroupCallback.notifyReleaseFailed(AdapterException).

## Custom Adapter Interfaces

MDS outbound processing is enabled through the following interfaces:

- IDeliveryGroupCallback

  This is a new interface that defines callbacks on a delivery group. A delivery group is an outbound collection of messages that originate from the same source and are addressed to the same destination.

- IDeliveryGroupDispatcher

  This is a new interface that defines methods for dispatching messages in a delivery group. A delivery group is an outbound collection of messages that originate from the same source and are addressed to the same destination.

# Configuring a JMS Adapter to Use a File Naming Service

The Java Message Service (JMS) API is a messaging standard that enables application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables reliable asynchronous communications.

If you are developing a JMS Adapter that will use a file naming service, you must configure it to produce a JMS Resource file. The resource file (.binding file) is an automatically generated file containing binding information. To configure the adapter for JMS, you must:

- Download a SUN Java Naming and Directory Interface (JNDI) file naming service (utilities class).
- Write java code that binds JMS objects to a file system using the SUN file naming service. Your code must provide values for four types of administered objects:
  - TopicConnectionFactory
  - QueueConnectionFactory
  - Topic
  - Queue

  You may obtain a sample configuration file from the company that supplies JMD JNDI interface software such as Tibco Software, Inc.
- Run the JNDI file service against the script to produce the JMS Resources file.

Later, the AON Management Console (AMC) is used to upload the JMS Resource file.

# Custom Adapter API Specification

AON includes APIs that can be used to develop embedded or standalone custom adapters. Supplied in the AON Adapter SDK, the API includes the following packages:

- Adapter Package, page 3-76
- IO Package, page 3-117
- Message Package, page 3-127
- Net Package, page 3-131
- Utilities Package, page 3-131
- Utilities Pool Package, page 3-144
- Exception Package, page 3-145

**Note** A general set of interfaces and classes is used in conjunction with the Custom Adapter SDK (and other SDKs). For more information, see Appendix A, AONSCommon Specification.

## Adapter Package

The Adapter package (**com.cisco.aons.adapter**) defines the interfaces and classes listed below.

Adapter Interfaces

- AttributeValue, page 3-84
- IAdapterConstants, page 3-86
- IAdapterContext, page 3-86
- IAdapterDescriptor, page 3-87
- IAdapterManager, page 3-89
- IAdvisoryListener, page 3-90
- IConnectionContext, page 3-90
- IConnectionReceiver, page 3-91

**Adapter Classes**

For more information, see IO Package, page 3-117, Message Package, page 3-127. Net Package, page 3-131, Utilities Package, page 3-131, Utilities Pool Package, page 3-144, and Exception Package, page 3-145

## AbstractOutboxHandler

public abstract class AbstractOutboxHandler

extends PooledJob

This abstract class determines how a message is sent out of AON. It includes the fields, methods, and constructor summarized below.

| Fields | Description |
|---|---|
| mAdapter | protected StandaloneAdapter mAdapter<br>Standalone adapter. |
| mMessage | protected IAONSMessage mMesage<br>Message to be sent out of AON. |
| mMessageHandler | protected IMessageHandler mMessageHandler<br>MessageHandler associated with this message. |
| mMsgType | protected int mMsgType<br>Message type. |

| Methods | Description |
|---|---|
| execute | public final void execute()<br>Sends out the message. |
| getMessage | public IAONSMessage getMessage()<br>Returns the AON message of the outbox handler. |

| Methods | Description |
|---|---|
| initialize | public void initialize(StandaloneAdapter pAdapter,<br>IMessageHandler pHandler,<br>IAONSMessage pMessage,<br>int pMsgType)<br>Parameters:<br>pAdapter—adapter sending out the message<br>pHandler—message processing the handler associated with this message processing instance.<br>pMessage—message to be sent out<br>pMsgType—message type being sent out.<br>Initializes the Outboxhandler for the message to be sent out of AON. The AON runtime calls this method. It is not invoked by adapters. |
| sendMessage | public abstract void sendMessage(int pMsgType)<br>throws AdapterException<br>Deposits the message into the outbox from which it is sent out of AON. This method is overridden by derived classes that implement the functions that send out a message. |

| Constructor | Description |
|---|---|
| AbstractOutboxHandler | public AbstractOutboxHandler () |

## AbstractSourceCallback

public abstract class AbstractSourceCallback

Extending java.lang.Object, this abstract class implements ISourceCallback.The following tables summarize the class methods and constructor.

| Methods | Description |
|---|---|
| mustYield | public abstract void mustYield()<br>throws AdapterException<br>Callback to the adapter. Forces it to yield the associated source. The adapter may be forced to yield the source if the system has to throttle incoming messages. After the adapter yields, it indicates completion by calling the yieldSource(Source) method of IResourceManager.<br>Throws: AdapterException if an exception occurs while the adapter is yielding the source. |

| Constructor | Description |
|---|---|
| AbstractSourceCallback | public AbstractSourceCallback() |

# Adapter

public abstract class Adapter

This main class must be included in all adapters, embedded and standalone. Custom adapters should not extend this class, instead they must extend EmbeddedAdapter or StandaloneAdapter. This class includes the field, methods, and constructor summarized below.

| Field | Description |
|---|---|
| mDescriptor | protected IAdapterDescriptor mDescriptor |

| Methods | Description |
|---|---|
| doInitialize | protected abstract void doInitialize()<br>throws AdapterInitException<br>Initializes the adapter. Specific adapter implementations are required to override this method for initialization. This call is made to the adapter when it is initialized.<br>Throws:<br>AdapterException<br>AdapterInitException |
| doShutdown | public abstract void doShutdown()<br>throws AdapterException |
| doShutdownInput | public abstract void doShutdownInput()<br>throws AdapterException<br>Shuts down the adapter. Adapters close their input mechanism and stop processing new application messages. This call is made to the adapter when AONS is being deactivated or the adapter is reloaded. |
| doShutdownOutput | public abstract void doShutdownOutput()<br>throws AdapterException<br>Shuts down adapter output. Adapters stop sending out messages. |
| doStart | public abstract void doStart()<br>throws AdapterException<br>Starts the adapter. |
| getAdapterDescriptor | public IAdapterDescriptor getAdapterDescriptor()<br>Returns the adapter descriptor associated with this adapter. |
| getAdapterManager | public final IAdapterManager getAdapterManager()<br>Gets the AdapterManger associated with this adapter. |
| getAdapterName | public final java.lang.String getAdapterName()<br>Returns the adapter name. |

| Methods | Description |
|---|---|
| getDeliveryDispatcher | public abstract IDeliveryGroupDispatcher getDeliveryDispatcher(IDeliveryGroupCallback pCallback) throws AdapterException<br>Parameters:<br>pCallback—callback on which the adapter send notification.<br>Returns a delivery dispatcher associated with this adapter. The delivery dispatcher is called only when the message is delivered in a group. The group may be used for batching or ordering. |
| getInteractionStyle | public int getInteractionStyle()<br>Returns the default interaction style for messages that has no associated PEP. Possible values:IMessageHandler fields REQUEST_ONLY (default) or REQUEST_REPLY. An adapter can override this to return another interaction style.<br>Returns: int the interaction style. Possible values are IMessageHandler.REQUEST_ONLY, IMessageHandler.REQUEST_REPLY |
| getOutboxHandler | public abstract AbstractOutboxHandler getOutboxHandler(IMessageHandler pMsgHandler, IAONSMessage pMessage, int pMsgType) throws AdapterException<br>Uses the AbstractOutboxHandler class to generate an outbox handler for the adapter.<br>An OutboxHandler is used to handle the message in the outbox, sending it out of AON. The message can be sent out as REQUEST or RESPONSE. The adapter must handle the message appropriately. Adapters extend the AbstractOutboxHandler class and provide the information as a part of registration process. |
| getType | public final int getType()<br>Returns the adapter type (integer). |
| initialize | protected void initialize(IAdapterDescriptor pDescriptor, IAdapterManager pAdapterManager) throws AdapterInitException<br>Initializes the descriptor and adapter manager. |
| isLoopBack | public boolean isLoopBack(URI pDestURI) throws AdapterException<br>Checks to determine if the message is a loopback. A loopback can occur if the message arrived as a gateway without a preset message destination. |
| reload | public void reload() throws AdapterException<br>Reloads the adapter with a new configuration |

| Constructor | Description |
|---|---|
| Adapter | public Adapter() |

## AdapterEvent

Extending java.util.EvenObject, the AdapterEvent class represents adapter events. It inherits the following components:

- field source and methods getSource and toString from java.util.EventObject
- methods clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait (three expressions) from class.java.lang.Object.

AdapterEvent also has one constructor:

- public AdapterEvent (java.lang.Object.psource).

## AdapterException

Extending AONSException, the AdapterException class represents adapter exceptions. It inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait (three expressions). AdapterException also has the constructors listed below.

| Constructor | Description |
|---|---|
| AdapterException | public AdapterException(java.lang.Exception pException) |
| AdapterException | public AdapterException(java.lang.Exception pException) |
| AdapterException | public AdapterException(java.lang.String pMessage,<br>java.lang.Throwable pEx) |
| AdapterException | public AdapterException(int pErrorCode,<br>java.lang.Throwable pEx) |
| AdapterException | public AdapterException(int pErrorCode) |
| AdapterException | public AdapterException(java.lang.String pMessage,<br>int pError) |
| AdapterException | public AdapterException(java.lang.String pMessage) |

## AdapterExtension

This is the base abstract class for all adapter extensions. An adapter developer can further extend the AdapterExtension class to define its own interface methods that it expects the extension developer to implement. The adapter framework makes no assumptions about this class.

AdapterExtension inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). It also includes the methods and fields summarized below.

| Methods | Description |
|---|---|
| getExtPolicyKey | public final java.lang.String getExtPolicyKey()<br>Returns the policy key (string) of the extension. |
| getExtPolicyReader | public final DomainReader getExtPolicyReader()<br>Returns the extension policy reader (DomainReader). |
| initialize | public final void initialize(java.lang.String pAdapterPolicyKey,<br>java.lang.String pExtensionPolicyKey,<br>DomainReader)<br>throws AdapterException<br>Parameters:<br>pAdapterPolicyKey—String adapter policy key, the property set key.<br>pExtensionPolicyKey—String extension policy key, the property set key/<br>This is the last method called by the adapter code to initialize an extension. Throws AdapterException if unable to initialize properly. |

| Fields | Description |
|---|---|
| ADAPTER_EXTENSION_ DOMAIN | public static final java.lang.String ADAPTER_EXTENSION_DOMAIN |
| ADAPTER_EXTENSION_ POLICY_LINK | public static final java.lang.String ADAPTER_EXTENSION_POLICY_LINK<br>This field is an attribute of Extension policy XML file. |
| AMCGLOBALPOLICYCATEGORY | public static final java.lang.String AMCGLOBALPOLICYCATEGORY |
| EXTENSION_POLICY_ NAME | public static final java.lang.String EXTENSION_POLICY_NAME<br>This field is an attribute of AdapterExtRegistry.xml. |

## AdapterInitException

This class is extended from AdapterExtension, this class is used for exceptions associated with adapter initialization. It includes the following methods inherited from java.lang.Object:clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). AdapterInitException also includes the constructors listed below:

The methods summarized below.

| Constructor | Description |
|---|---|
| AdapterInitException | public AdapterInitException(java.lang.Exception pException) |
| AdapterInitException | public AdapterInitException(java.lang.String pMessage,<br>int pErrorCode,<br>java.lang.Throwable pException) |

| Constructor | Description |
|---|---|
| AdapterInitException | public AdapterInitException(java.lang.String pMessage, java.lang.Throwable pEx) |
| AdapterInitException | public AdapterInitException(int pErrorCode, java.lang.Throwable pEx) |
| AdapterInitException | public AdapterInitException(int pErrorCode) |
| AdapterInitException | public AdapterInitException(java.lang.String pMessage, int pError) |
| AdapterInitException | public AdapterInitException(java.lang.String pMessage) |

## AdapterTask

public abstract class AdapterTask

extends PooledJob

This is a tagging interface that marks implementations of this interface as an adapter task. The interface inherits fields from PooledJob. The methods and constructor are summarized below.

| Methods | Description |
|---|---|
| execute | public void execute()<br>This method is invoked when a pooled job is pulled from the queue for execution. |
| getAdapter | public final Adapter getAdapter() |

| Constructor | Description |
|---|---|
| AdapterTask | public AdapterTask()<br>Default constructor. |
| AdapterTask | public AdapterTask(Adapter pAdapter)<br>Takes the adapter as an input.<br>Parameters: pAdapter - Adapter |

## AttributeValue

public interface AttributeValue

Extending java.lang.Object, this public interface is used to define source attribute constants. It includes the fields and methods summarized below.

| Fields | Description |
|---|---|
| NOT_PINNED | static final AttributeValue NOT_PINNED<br>An attribute value that indicates that the source is not pinned |
| PINNED | static final AttributeValue PINNED<br>An attribute value that indicates that the source is pinned |

| Fields | Description |
|---|---|
| REQUEST | static final AttributeValue REQUEST<br>An attribute value that indicates that the source is consuming request message |
| RESPONSE | static final AttributeValue RESPONSE<br>An attribute value that indicates that the source is consuming response message |
| UNKNOWN | static final AttributeValue UNKNOWN<br>Unknown attribute value |

| Methods | Description |
|---|---|
| getAffinity | public AttributeValue getAffinity()<br>Gets the associated affinity value. This method is involved in Graceful Handling operations. |
| getMessageType() | public AttributeValue getMessageType()<br>Gets the associated message type. This method is involved in Graceful Handling operations. |

## EmbeddedAdapter

Extending class Adapter., this is the base class for all embedded adapters.Closely integrated with the AON runtime engine, an embedded adapter must define MessageReceiveHandler and MessageSendHandler classes. EmbeddedAdapter inherits one field from the Adapter class: mDescriptor. It also includes the methods listed below.

| Methods | Description |
|---|---|
| getDeliveryDispatcher | public abstract IDeliveryGroupDispatcher getDeliveryDispatcher(IDeliveryGroupCallback pCallback)<br>throws AdapterException<br>Parameters:<br>pCallback—callback on which the adapter send notification.<br>Returns a delivery dispatcher associated with this adapter. The delivery dispatcher is called only when the message is delivered in a group. The group may be used for batching or ordering. |
| getEmbeddedAdapterManager | Gets IEmbeddedAdapterManager to generate an embedded adapter manager. |
| getMessageReceiveHandler | Uses MessageReceiveHandler to generate a message receive handler |
| getMessageSendHandler | Uses MessageSendHandler to generate a message send handler. |
| getOutboxHandler | Uses AbstractOutboxHandler to generate an outbox handler for the embedded adapter |

## IAdapterConstants

public interface IAdapterConstants

This interface provides definitions for all constants (adapter request, response, protocol, and so on) used by the adapter. The interface fields are summarized below.

| Fields | Description |
|---|---|
| ADAPTER_REQUEST | public static final int ADAPTER_REQUEST<br>Adapter request. |
| ADAPTER_RESPONSE | public static final int ADAPTER_RESPONSE<br>Adapter response. |
| AONS_ADAPTER_LOGGER | public static final java.lang.String AONS_ADAPTER_LOGGER |
| CLEAR_MODE | public static final int CLEAR_MODE<br>Indicates a clear mode. |
| EVENT_ACCEPT | public static final int EVENT_ACCEPT<br>Accept the event. |
| EVENT_CLOSE | public static final int EVENT_CLOSE<br>Close the channel. |
| EVENT_CONNECT | public static final int EVENT_CONNECT<br>Connect the event. |
| EVENT_INPUT_CLOSE | public static final int EVENT_INPUT_CLOSE<br>Indicates that the input channel is closed. |
| EVENT_OUT_OF_BUFFER | public static final int EVENT_OUT_OF_BUFFER<br>Indicates that the event has an out of buffer status. |
| EVENT_OUTPUT_CLOSE | public static final int EVENT_OUTPUT_CLOSE<br>Indicates that the output channel is closed. |
| EVENT_READ | public static final int EVENT_READ<br>Read event. |
| EVENT_WRITE | public static final int EVENT_WRITE<br>Write event. |
| PROTOCOL | public static final java.lang.String[] PROTOCOL |
| SECURE_MODE | public static final int SECURE_MODE<br>Indicates a secure mode. |
| STATUS_OK | public static final int STATUS_OK<br>Indicates an OK status. |

## IAdapterContext

public interface IAdapterContext

This interface is used to get the adapter context. It includes the single method summarized below.

| Methods | Description |
|---|---|
| getAdvisoryListener | public IAdvisoryListener getAdvistoryListener()<br>Returns the IAdvisoryListener. |

## IAdapterDescriptor

public interface IAdapterDescriptor

This interface defines adapter types such as embedded, standalone, and start up mode.It includes the fields and methods listed below.

| Fields | Description |
|---|---|
| AD_BUILTIN | static final int AD_BUILTIN<br>Built-in type. |
| AD_EMBEDDED | static final int AD_EMBEDDED<br>Embedded type. |
| AD_LENIENT | static final int AD_LENIENT<br>Lenient startup mode. |
| AD_STANDALONE | static final int AD_STANDALONE<br>Standalone type. |
| AD_STRICT | static final int AD_STRICT<br>Strict startup mode. |
| AD_THIRDPARTY | static final int AD_THIRDPARTY<br>Third party. |

| Constructor | Description |
|---|---|
| getAdapterClassName | public java.lang.String getAdapterClassName()<br>Returns (string) the adapter class name. |
| getAdapterName | public java.lang.String getAdapterName()<br>Returns (string) the adapter name. |
| getAttributeDomainNames | public java.lang.String[] getAttributeDomainNames()<br>Returns (string[]) the attribute domain name. |
| getClassLoader | java.lang.ClassLoader getClassLoader()<br>Returns: Classloader |
| getConfigFileName | public java.lang.String getConfigFileName()<br>Returns (string) the configuration file name. |

| Constructor | Description |
|---|---|
| getDomainReader | public DomainReader getDomainReader(java.lang.String pAttributeName)<br>throws AdapterException<br>Parameters:<br>pAttributeName—String<br>Returns the domain reader. |
| getIdentifier | public int getIdentifier()<br>Returns (int) the identifier. |
| getMode | public int getMode()<br>Returns(int) the mode. |
| getNetworkListeners | public java.util.Iterator getNetworkListeners()<br>Returns (iterator) the network listeners. |
| getOutboxHandlerClass | public java.lang.Class getOutboxHandlerClass()<br>throws java.lang.ClassNotFoundException<br>Returns the class.‘ |
| getProtocol | public java.lang.String getProtocol()<br>Returns (string) the protocol name. |
| getProtocolAliases | public java.lang.String[] getProtocolAliases()<br>Returns sString) one or more alternate protocol aliases for the supported protocol. |
| getReceiveHandlerClass() | public java.lang.Class getReceiveHandlerClass()<br>throws java.lang.ClassNotFoundException<br>Returns the class. |
| getResource | public java.io.InputStream getResource(java.lang.String pConfigFile)<br>throws java.util.MissingResourceException<br>Parameters:<br>pConfigFile—String<br>Returns the input stream resource. |
| getSendHandlerClass | public java.lang.Class getSendHandlerClass()<br>throws java.lang.ClassNotFoundException<br>Returns the class. |
| getType | public int getType()<br>Returns (int) the type. |
| isActive | public boolean isActive()<br>Returns a boolean (equivalent to yes/no or is/is not active). |

## IAdapterManager

public interface IAdapterManager

This interface defines the adapter manager. It includes the methods listed below.

| Methods | Description |
|---|---|
| dispatchTask | public void dispatchTask(AdapterTask pTask)<br>Parameters:<br>pTask—IAdapterTask, the task.<br>Schedules the task for immediate execution. |
| getLogger | public Log getLogger()<br>Returns the logger associated with this adapter for logging. The log writes to the standard AON log file with the category "mec.adapters.xyz" where xyz is the name of the adapter. |
| getMessageBuilder | public IAdapterMessageBuilder getMessageBuilder()<br>Gets the message builder to create messages |
| getMessageSizeLimit | double getMessageSizeLimit()<br>Returns the system message size limit (a double). |
| getPersistentManager | public IPersistentManager getPersistentManager()<br>throws AdapterException<br>Returns IPersistentManager, the persistent manager. |
| getProtocol | public int getProtocol()<br>Returns the protocol ID associated with the adapter |
| getResourceManager | public IResourceManager getResourceManager()<br>Returns IResourceManager, the resource manager. |
| handleError | public void handleError(IMessageHandler pHandler,<br>int pErrorCode)<br>Parameters:<br>pErrorCode—Error code<br>Handler for errors. An adapter can use this method to report an error to AON. |
| handleException | public void handleException(IMessageHandler pHandler,<br>java.lang.Exception pEx)<br>Handler for exceptions. The way that AON handles the exception depends on the error code. |
| isLocalHostAddress | public boolean isLocalHostAddress(java.lang.String pHostIp)<br>Checks to determine if HostIp belongs to the local host. |

| Methods | Description |
|---|---|
| registerForAttachment | public int registerForAttachment(java.lang.String pAttachKey)<br>Parameters:<br>pAttachKey—Key that the adapter uses to register the attachment.<br>Returns the key index. IMessageHandler.getAttachment is used with this index to retrieve the attachment.<br>Used to register a key for attachment in IMessageHandler. An adapter may require information to be preserved during message processing for a particular message; for example, from dispatching the request to AON runtime to completion of response. In this case, an adapter can hook in objects of type IMessageAttachment into IMessageHandler object.<br>Each adapter should register for the type of attachment it may have to set at runtime. An adapter registers this attachment during adapter initialization. The adapter is responsible for caching the index returned by this call and using it at a later time to set and retrieve IMessageAttachment objects from IMessageHandler. The registered index is constant throughout the runtime of that AON instance but it does not persist across restarts. |
| registerForDeliveryContext | public int registerForDeliveryContext(java.lang.String pContextKey,<br>java.lang.String pClassName) |
| scheduleTask | public void scheduleTask(SchedulableTask pTask,<br>int pInterval)<br>Parameters:<br>pTask—SchedulableTask, the task<br>pInterval—int the interval specified in seconds<br>Schedules the task for execution after a specified interval. The task will not be removed from this schedule until it is de-registered. |

## IAdvisoryListener

public interface IAdvisoryListener

This interface is used to listen for events. It includes the method summarized below.

| Methods | Description |
|---|---|
| onEvent | public void onEvent(AdapterEvent pEvent) |

## IConnectionContext

public interface IConnectionContext

The IConnectionContext interface has methods that obtain the ConnectionID, Reader, SourceIP, SourcePort, Writer, and SSLCertificate for this connection. Each IConnectionContext object is associated with a MessageReceiveHandler. IConnectionContext includes the fields and methods summarized below.

| Fields | Description |
|---|---|
| CLEAR_MODE | static final int CLEAR_MODE<br>Clear connection mode. |
| SECURE_MODE | static final int SECURE_MODE<br>Secure connection mode. |

| Methods | Description |
|---|---|
| getConnectionId() | public java.lang.String getConnectionId()<br>Returns the connection ID associated with this connection. |
| getDestinationIP | public java.net.InetAddress getDestinationIP()<br>Returns the destination IP. |
| getDestinationPort | public int getDestinationPort()<br>Returns (int) the destination port number. |
| getReader | public IAdapterReader getReader()<br>Returns the IAdapterReader interface to define a reader. |
| getSourceIP | public java.net.InetAddress getSourceIP()<br>Returns (InetAddress) the source IP. |
| getSourcePort | public int getSourcePort()<br>Returns (int) the source port number. |
| getSSLCertificate | public java.security.cert.Certificate getSSLCertificate()<br>Returns the SSL certificate. |
| getWriter | public IAdapterWriter getWriter()<br>Returns the IAdapterWriter to define a writer |
| isConnectionImplicit | boolean isConnectionImplicit()<br>Returns boolean true for implicit' Otherwise, returns false. |
| isSecure | public boolean isSecure()<br>Returns "true" if the connection is a secure (SSL) connection. |

## IConnectionReceiver

public interface IConnectionReceiver

This interface defines the connection receiver. It includes the method listed below.

| Methods | Description |
|---|---|
| release | void release(IAONSMessage pMessage)<br>Parameters: pMessage, the AONSMessage on which release happens.<br>Instructs the receiver to release the connection for this message and cleans itself up. |

## IDeliveryContextCallback

public interface IDeliveryContextCallback

This interface returns the DeliveryContexts. It includes the method listed below.

| Methods | Description |
|---|---|
| recoveredDeliveryContexts | void recoveredDeliveryContexts(java.util.HashMap pContexts) throws AdapterException<br>Parameters: pMsgCtx - message context.<br>The adapter invokes this callback after it has received recoverDeliveryContext on startup. The adapter returns a collection of DeliveryContext objects. The adapter creates delivery context object and calls deserialize. |

## IDeliveryGroup

public interface IDeliveryGroup

This interface defines outbound delivery group. One or more DeliveryGroups may be associated with DeliveryGroups. It includes the method summarized below.

| Methods | Description |
|---|---|
| getGroup | IGroup getGroup()<br>Gets the outbound delivery group. |

## IDeliveryGroupCallback

public interface IDeliveryGroupCallback

This interface defines callbacks on a delivery group. A delivery group is an outbound collection of messages that originate from the same source and are addressed to the same destination. It includes the methods listed below.

| Methods | Description |
|---|---|
| getDeliveryDestination | URI getDeliveryDestination()<br>Returns the destination URI associated with this delivery group. |
| getDeliveryGroupId | java.lang.String getDeliveryGroupId()<br>Returns the unique delivery group identifier associated with this delivery group. |

| Methods | Description |
|---|---|
| getGroup | IGroup getGroup()<br>Returns the group associated with this delivery group. A group is an inbound collection of messages that originate from the same source. When the messages in a group are addressed to the same destination, they are in the same delivery group. Since messages in the same group can be rerouted and addressed to different destinations, one group may be associated with more than one delivery group. |
| notifyCommit | void notifyCommit()<br>Callback indicating that the current batch of messages in the delivery group have been committed by the adapter. This sets a commit point so that all messages delivered up to this point will not have to be redelivered on subsequent rollbacks as indicated by notifyRollback(). After calling this method, an adapter must call notifyReadyToReceive(IAONSMEssage) to continue message delivery in the delivery group. |
| notifyReadyToReceive | void notifyReadyToReceive(IAONSMessage pLastDelivered)<br>Parameters:<br>pLastDelivered—last message written out<br>Callback indicating that an adapter is ready to receive the next message in the delivery group. The next message in the delivery group will be sent to the adapter only after this notification has been invoked or IDeliveryGroupDispatcher.sendMessage(IAONSMessage, IMessageHandler, int) shows an exception. The adapter must send this notification after it has written the message out successfully whether it has committed the message or not. |
| notifyReleaseCompleted | void notifyReleaseCompleted()<br>Callback indicating that the adapter has successfully released this delivery group. The adapter must release this delivery group when IDeliveryGroupDispatcher.releaseDestination() is called. |
| notifyReleaseFailure | void notifyReleaseFailure(AdapterException pException)<br>Callback indicating that an exception occurred during the release of the delivery group. The adapter must release this delivery group when IDeliveryGroupDispatcher.releaseDestination() is called. |
| notifyRollback() | void notifyRollback()<br>Callback indicating that the adapter has rolled back to the last commit point as indicated by notifyCommit(). All messages in the delivery group since the last commit point must be redelivered. After calling this method, an adapter s must call notifyReadyToReceive(IAONSMessage) to continue message delivery in the delivery group. |

## IDeliveryGroupDispatcher

public interface IDeliveryGroupDispatcher

This interface defines methods for dispatching messages in a delivery group. A delivery group is an outbound collection of messages that originate from the same source and are addressed to the same destination. It includes the methods summarized below.

| Methods | Description |
|---|---|
| getAdapter | Adapter getAdapter()<br>Returns the adapter associated with this delivery group dispatcher |
| isTransacted | boolean isTransacted()<br>Returns true if this delivery group dispatch is transacted, otherwise false.<br>Determines if this delivery group dispatcher is transacted. |
| releaseDestination | void releaseDestination()<br>throws AdapterException<br>Releases the destination for the associated delivery group. A delivery group is associated with an adapter on a single network node until the adapter is asked to release it. The adapter may be asked to release the delivery group if another adapter on another network node in the virtual cluster is to be assigned the delivery group.<br>After releasing the destination, the adapter must indicate completion by calling IDeliveryGroupCallback.notifyReleaseCompleted or IDeliveryGroupCallback.notifyReleaseFailure(AdapterException). |
| sendMessage | void sendMessage(IAONSMessage pMessage,<br>IMessageHandler pHandler,<br>int pMsgType)<br>throwsAdapterException<br>Parameters:<br>pMessage—message to be sent<br>pHandler— message handler to be used<br>pMsgType—message type<br>Sends (dispatches) the given message of the given type to the adapter using the given message handler. |

## IEmbeddedAdapterManager

public interface IEmbeddedAdapterManager

Extending IAdapterManager, this interface defines an embedded adapter manager. It includes the methods listed below.

| Methods | Description |
|---|---|
| createConnection | IConnectionContext createConnection(Adapter pAdapter,<br>java.net.InetAddress pIPAddr,<br>int pPort,<br>boolean pIsSecure,<br>boolean pIsCacheable)<br>throws AdapterException<br>Parameters:<br>pAdapter—adapter which is creating this connection<br>pIPAddr—IP address for connection<br>pPort—port for this connection<br>pIsSecure—asks if the connection is secure connection<br>pIsCacheable—connection to be pooled<br>Creates an active connection. An adapter may create outbound active connections for protocol support.<br>**Caution** This method should not be used for the primary connection to endpoints. It could, for example, be used in Active mode FTP> |
| createListener | INetworkListener createListener(Adapter pAdapter,<br>java.net.InetAddress pIpAddress,<br>int pPort,<br>boolean pIsSecure)<br>throws AdapterException<br>Parameters:<br>pAdapter—adapter associated with this listener<br>pIPAddress—IP address associated with this listener<br>pPort—port associated with this listener<br>pIsSecure—asks if the listener is secure.<br>Creates a secure listener. This method is used only to create secondary listeners, not the primary listener. Primary listeners are always created by AON runtime. Primary listeners are specified at the configuration time and managed by AON. This method should be used to create secondary or temporary listeners that will be managed by the adapter that created the listener. For example, this method could be used to create listeners for passive mode FTP. |

| Methods | Description |
|---|---|
| getBufferManager | IBufferManager getBufferManager()<br>Gets IBufferManager to create a buffer manager. The BufferManager provides the interface for all buffer related operations such as reading and writing messages. |
| getDefaultConnKeepAlive | long getDefaultConnKeepAlive()<br>Returns (long) the default connection keep alive value. |
| getDefaultOutboxHandler | AbstractOutboxHandler getDefaultOutboxHandler(IAONSMessage pMessage)<br>Gets the AbstractOutboxHandler class to generate a default hotbox handler. |
| getDeliveryDispatcher | IDeliveryGroupDispatcher getDeliveryDispatcher(IDeliveryGroupCallback pCallback)<br>throws AdapterInitException<br>Gets the delivery dispatcher associated with this adapter. |
| getInitialBufferSize | int getInitialBufferSize()<br>Returns initial buffer size set by AON for reading in messages. The return value will change depending on the system load. The adapter should not cache this value.<br>Returns: int the initial buffer size |
| registerEvent | void registerEvent(IConnectionContext pContext,<br>int pEvent)<br>throws java.nio.channels.ClosedChannelException<br>Registers Read/Write/Connect event on the IConnectionContext. Adapters use this method to register for read/write/connect events with AON runtime. This method is used to register for events only on secondary listeners and active connections created by adapters An active connection could be in the case of Active mode FTP. |

## IErrorCodes

public interface IErrorCodes

Handles error codes. It has the fields summarized below.

**Note** Because these error code handlers are self-descriptive, the following table does not include a separate description. For example, ERR_BAD_URI is associated with the detection of an incorrect URI.

| Fields | Description |
|---|---|
| ERR_BAD_URI | static final int ERR_BAD_URI |
| ERR_CONNECTING_DESTINATION | static final int ERR_CONNECTING_DESTINATION |
| ERR_CONNECTING_NEXTHOP | static final int ERR_CONNECTING_NEXTHOP |

| Fields | Description |
|---|---|
| ERR_CONNECTION_RESET | static final int ERR_CONNECTION_RESET |
| ERR_DUPLICATE_MESSAGE | static final int ERR_DUPLICATE_MESSAGE |
| ERR_INTERNAL_ERROR | static final int ERR_INTERNAL_ERROR<br>Message state error code. |
| ERR_INVALID_MESSAGE_STATE | static final int ERR_INVALID_MESSAGE_STATE |
| ERR_LOOPBACK_MESSAGE | static final int ERR_LOOPBACK_MESSAGE |
| ERR_MSG_SIZE_EXCEED_LIMIT | static final int ERR_MSG_SIZE_EXCEED_LIMIT |
| ERR_NO_FLOW_FOR_MESSAGE | static final int ERR_NO_FLOW_FOR_MESSAGE |
| ERR_ORPHAN_MESSAGE | static final int ERR_ORPHAN_MESSAGE |
| ERR_UNKNOWN_CONTROL_TYPE | static final int ERR_UNKNOWN_CONTROL_TYPE |
| ERR_UNKNOWN_DESTINATION | static final int ERR_UNKNOWN_DESTINATION |
| gMessageErrBase | static final int gMessageErrBase |
| INSUFFICIENT_STORAGE | static final int INSUFFICIENT_STORAGE<br>Indicates an insufficient storage condition |
| IO_ERROR | static final int IO_ERROR<br>Indicates an IO error. |
| REQ_TXN_FAILED | static final int REQ_TXN_FAILED<br>Indicates that a request transaction failed. |

## IGroup

public interface IGroup

Extending IMessageDispatcher, this interface defines methods for dispatching messages in a group. A group is an inbound collection of messages that originate from the same source. Adapters use IResourceManager.registerSource to get an instance of this class. Adapters should use the returned IGroup object to dispatch inbound messages from the registered source. IGroup includes the fields and methods summarized below.

| Fields | Description |
|---|---|
| ORDERED | static final int ORDERED<br>Field indicates that the group is ordered |
| PERSISTENT_GROUP | static final int PERSISTENT_GROUP<br>Field indicates that the group is persistent. |

| Methods | Description |
|---|---|
| getGroupId() | java.lang.String getGroupId()<br>Gets the unique group identifier associated with this group. |

| Methods | Description |
|---|---|
| getResourceId() | long getResourceId()<br>Gets the unique resource identifier associated with this group. |
| getType() | int getType()<br>Gets the type associated with this group |

## IMessageContextStore

public interface IMessageContextStore

This interface handles message context store deletion and recovery operations. It includes the methods summarized below.

| Methods | Description |
|---|---|
| deleteMessageContext | boolean deleteMessageContext(java.lang.String pCorrelationId,<br>URI pURI)<br>throws AdapterException<br>Deletes a specific message context. |
| recoverDeliveryContext | void recoverDeliveryContexts(IDeliveryContextCallback pCallback)<br>throws AdapterException<br>Called to initialize time to recover all the message context from the adapter. After recovery, the contexts are returned via the callback. The adapter may make this callback many times.<br>Parameters:<br>pCallback - IMessageContextCallback |
| recoverMessageContext | IMessageContext recoverMessageContext(java.lang.String pCorrelationId,<br>URI pURI)<br>throws AdapterException<br>Parameters: pCorrelationId - String<br>pURI - URI<br>Returns: IMessageContext |

## IMessageDispatcher

public interface IMessageDispatcher

Dispatches AON messages for processing in the AON run-time system. It includes the methods summarized below.

| Methods | Description |
|---|---|
| dispatch | void dispatch(IAONSMessage pMessage,<br>IMessageHandlerCallback pHandlerCallback)<br>throws AdapterException<br>Dispatches the message read in to AON Inbox for Policy Execution Plans processing The adapter has read the message (context of the adapter thread) and uses this API to dispatch it. |
| dispatch | void dispatch(IAONSMessage pMessage,<br>IMessageHandlerCallback pHandlerCallback, IMessageWriteCompleteCallback pCallback)<br>throws AdapterException<br>Parameters<br>pMessage—message to be dispatched.<br>pCallback—the callback to be called when message write has been completed.<br>Dispatches the message read in to AON Inbox for PEP processing. The adapter has read this message (adapter thread context) and uses this API to dispatch the message. |
| dispatch | void dispatch(StandaloneMessageReader pMessageReader)<br>throws AdapterException<br>Parameters:<br>pMessageReader—StandaloneMessageReader<br>Dispatches a callback to the AON Inbox. The system calls the callback is when the message is ready to be read in. In this case, the message is read in context of the MEC thread. |

## IMessageHolder

public interface IMessageHolder

This interface holds the AON message. It includes the method listed below.

| Methods | Description |
|---|---|
| getMessage | public IAONSMessage getMessage()<br>throws AdapterException<br>Returns IAONSMessage to get the message associated with this handler. The message may not have been completely read in |

## IPersistentManager

public interface IPersistentManager

This interface provides a persistent manager. It includes the methods listed below.

| Methods | Description |
|---|---|
| persist | void persist(java.lang.String pSelectKey,<br>java.io.Serializable pObject,<br>throws AdapterException<br>Parameters:<br>pSelectKey—String<br>pObject—Serializable |
| persist | void persist(java.lang.String pSelectKey,<br>java.io.Serializable pObject,<br>long pExpiration)<br>throws AdapterException<br>Parameters:<br>pSelectKey—String<br>pObject—Serializable<br>pExpiration—long |
| retrieve | java.io.Serializable retreive(java.lang.String pSelectKey)<br>throws AdapterException<br>Parameters:<br>pSelectKey—String<br>Returns: Serializable |

## IResourceManager

public interface IResourceManager

This interface manages adapter resources. Adapters can get an instance of this class using IAdapterManager.getResourceManager(). It includes the methods listed below.

| Methods | Description |
|---|---|
| deregisterSource | void deregisterSource(Source pSource)<br>Parameters:<br>pSource—source to deregister<br>Deregisters a source. This method should be invoked by adapters for all sources as part of the adapter input shutdown process in the adapter specific implementation of Adapter method doShutdownInput(). |
| registerMessageContextStore | int registerMessageContextStore(IMessageContextStore pStore, Adapter pAdapter)<br>Parameters:<br>pSource—message context store to be registered<br>pAdapter—adapter for which the message context store will be registered<br>Registers the given message context store for the adapter. This method should be invoked by adapters that require reliable delivery as part of the adapter startup process in the adapter specific implementation of Adapter method doStart(). |
| registerSource | IGroup registerSource(OrderedSource pSource,<br>AbstractSourceCallback pCallback)<br>Parameters:<br>pSource—ordered source to register<br>pCallback—implements callbacks that will be invoked when the sources is acquired or yielded.<br>Returns the IGroup object associated with the ordered source, registering the ordered source. This method should be invoked by adapters for all ordered sources as part of the adapter startup process in the adapter specific implementation of Adapter method doStart(). Later, the adapter will be contacted to acquire or yield the source using the given ISourceCallback object. |

| Methods | Description |
|---|---|
| registerSource | IGroup registerSource(UnorderedSource pSource,<br>AbstractSourceCallback pCallback)<br>Parameters:<br>pSource—unordered source to register<br>pCallback—implements callbacks that will be invoked when the source is acquired or yielded.<br>Returns the IGroup object associated with the unordered source, registering the unordered source. This method should be invoked by adapters for all unordered sources as part of the adapter startup process in the adapter specific implementation of Adapter method doStart(). Later, the adapter will be notified to acquire or yield the source using the given ISourceCallback object. |
| shouldYield | boolean shouldYield(Source pSource)<br>Parameters:<br>pSource—source to check<br>Returns true if the source can be distributed and should be yielded, otherwise false. This method determines whether or not the adapter should yield the given source to allow an adapter on another network node in the virtual cluster to acquire the source. Adapters should invoke this method before yielding a source. |
| yieldSource | void yieldSource(Source pSource)<br>Parameters:<br>pSource—source that was yielded<br>Callback indicating that the given source has been successfully yielded.<br>Adapters should periodically yield sources that can be distributed to- and acquired by- adapters on other network nodes in the virtual cluster to share message processing load. Adapters may also be requested to yield a source via the ISourceCallback method notifyYield. An adapter should yield as source only if the method shouldYield (Source) returns true. |

## ISourceCallback

public interface ISourceCallback

This interface defines callbacks for adapters to be notified to act on associated sources. Adapters should pass in an object which implements this interface as part of source registration via IResourceManager.registerSource. It includes the methods listed below.

| Methods | Description |
|---|---|
| notifyAcquire | boolean notifyAcquire()<br>throws AdapterException<br>Returns false if the adapter does not want to acquire this source otherwise, true.<br>Callback to the adapter to acquire the associated source. A source is associated with an adapter on a single network node until the adapter yields it.<br>Throws AdapterException if an exception occurs while the adapter is acquiring the source. |
| notifyYield | boolean notifyYield()<br>throws AdapterException<br>Returns false if the adapter does not want to yield this source otherwise, true.<br>This callback to the adapter requests that it to yield the associated source. The adapter is asked to yield the source if another adapter on another network node in the virtual cluster is to be assigned the source. Adapters can also yield sources without being notified.<br>Before attempting to yield, the adapter should determine if it is necessary by calling the shouldYield(Source) method. After the adapter yields, it must indicate completion by calling the yieldSource(Source) method of IResourceManager.<br>Throws AdapterException if an exception occurs as the adapter is yielding the source. |

## IStandaloneAdapterManager

public interface IStandaloneAdapterManager

extends IAdapterManager, IMessageDispatcher

This interface represents the standalone adapter manager. It includes all dispatch methods. IStandaloneAdapterManager inherits the following methods from IAdapterManager: dispatchTask, getLogger, getMessageBuilder, getPersistentManager, getProtocol, getResourceManager, handleError, handleException, isLocalHostAddress, registerForAttachment, registerForDeliveryContext, and scheduleTask. The interface has the additional method summarized below.

| Methods | Description |
|---|---|
| dispatch | void dispatch(AbstractOutboxHandler pOutboxHandler)<br>throws AdapterException<br>Dispatches the outbox handler again. This method is primarily used when the adapter is unable to send messages out immediately but wants to retry at a later time. |

## LargeMessageSizeException

public class LargeMsgSizeException

extends AdapterException

This exception class inherits methods (create, equals, getArgs, getErrorCode, getResourceMessage, and so on) from AONSExceptionand includes the constructors summarized below.

| Constructors | Description |
|---|---|
| LargeMsgSizeException | public LargeMsgSizeException(java.lang.String pMessage,<br>int pErrorCode,<br>java.lang.Throwable pException) |
| LargeMsgSizeException | public LargeMsgSizeException(java.lang.String pMessage,<br>java.lang.Throwable pEx) |
| LargeMsgSizeException | public LargeMsgSizeException(int pErrorCode,<br>java.lang.Throwable pEx) |
| LargeMsgSizeException | public LargeMsgSizeException(int pErrorCode) |
| LargeMsgSizeException | public LargeMsgSizeException(java.lang.String pMessage,<br>int pError) |
| LargeMsgSizeException | public LargeMsgSizeException(java.lang.String pMessage) |

## MessageIOHandler

Extended from java.lang.Object, this is the base class for message handlers MessageReceiveHandlerand MessageSendHandler. MessageIOHandler includes the methods and fields listed below.

| Methods | Description |
|---|---|
| completeHandshake | public abstract int completeHandshake(int pEvent,<br>IConnectionContext pContext)<br>throws AdapterException<br>Parameters:<br>pEvent—int<br>pContext—IConnectionContext<br>After connection initialization, this method is called until the adapter makes the connection handshake. |
| discardMessage | public void discardMessage()<br>throws AdapterException<br>Called to discard a message |
| doInitialize | public abstract void doInitialize(int pMode,<br>int pMessageType,<br>IConnectionContext pContext)<br>throws AdapterException<br>Called to initialize the handler. |
| fetchMessage | public abstract int fetchMessage(int pEvent,<br>IConnectionContext pContext)<br>throws AdapterException<br>Parameters:<br>pEvent—IO event on which this method has been invoked.<br>pContext—IConnectionContext<br>Returns:<br>STATUS_OK—if the message has been read in completely. EVENT_READ—if the message has been read in partially, more bytes needs to be read from Channel.<br>EVENT_WRITE—if the adapter has to write<br>This non-blocking call fetches messages from the input stream. This method reads in the bytes available in the stream from the channel. If more bytes must be read to complete the message, this method returns STATUS_REGISTER_READ. |
| getAdapter | public Adapter getAdapter()<br>Gets the adapter associated with this handler |
| getConnectionContext | public final void setConnectionContext(IConnectionContext pConnCtx)<br>Gets the connection context associated with this adapter handler. |

| Methods | Description |
|---|---|
| getHandlerType | public final int getHandlerType()<br>Returns the handler type:<br>HANDLER_RECEIVE—type if the adapter is of type AdapterReceiveHandler<br>HANDLET_SEND—type if the adapter is of type AdapterSendHandler |
| getMessage | public abstract IAONSMessage getMessage()<br>throws AdapterException<br>Returns the message associated with this handler. This method returns the IAONSMessage object. The message may not have been read in completely. |
| getMessageHandler | public final IMessageHandler getMessageHandler()<br>Returns the message handler associated with this message context. The handler is null for an inbound message. A new handler is created only when the inbound message is pulled-dispatched into the framework message. |
| getReader | public IAdapterReader getReader()<br>throws AdapterException<br>Gets the adapter reader associated with this adapter handler. |
| getWriter | public IAdapterWritergetWriter()<br>throws AdapterException<br>Gets the adapter writer associated with this adapter handler. |
| initialize | public final void initialize(Adapter pAdapter,<br>IConnectionContext pConnCtx,<br>IMessageHandler pMsgHandler,<br>int pMode,<br>int pMessageType)<br>throws AdapterException<br>Parameters:<br>pAdapter—adapter to which this handler belongs.<br>pMode—protocol mode for this handler:<br>ADAPTER_REQUEST—mode if the handler has to make a protocol request.<br>ADAPTER_RESPONSE—mode if the handler has to make a protocol reply.<br>Initializes the adapter handler. |

| Methods | Description |
|---|---|
| keepConnectionAlive | public long keepConnectionAlive()<br>Returns:<br>0—connection will be close immediately<br>> 0—connection will remain open for specified amount of time.<br>Returns the length time (in seconds) that the connection needs to be kept alive after the response has been sent and before the next request is received. If the connection has to be closed immediately, this method must return 0. |
| onConnection | protected abstract int onConnection(IConnectionContext pContext)<br>throws AdapterException<br>Called when a new connection is established.<br>Returns:<br>EVENT_READ—if the adapter needs to read in data first. (for example, HTTP).<br>EVENT_WRITE—if the adapter needs to write some data first. (SMTP, FTP, and so on) |
| setConnectionContext | public final IConnectionContext getConnectionContext()<br>Gets the connection context associated with this adapter handler. |

## MessageReceiveHandler

Extended from the MessageIOHandler, this is the base class for all receive handlers associated with EmbeddedAdapter. Every EmbeddedAdapter class must extend this class to define its own Receive handler.

A receive handler reads in a message and dispatches it to AONS. An instance of MessageReceiveHandler is created for each message. All messages (requests and responses) to AONS are read using MessageReceiveHandler which, in turn, uses MessageIOHandler methods (see list, below).

A MessageReceiveHandler is always associated with an IConnectionContext object. An IConnectionContext object applies across multiple MessageReceiveHandler objects. MessageReceiveHandler calls the MessageIOHandler methods onConnection and completeHandshake for first message in the connection. After that, it invokes only fetchHeader and fetchMessage.

MessageReceiveHandlerinherits the following methods from MessageIOHandler:

- completeHandshake
- discardMessage
- doInitialize
- fetchMessage
- getAdapter
- getConnectionContext
- getHandlerType
- getMessage
- getMessageHandler

- getReader
- getWriter
- initialize
- keepConnectionAlive
- onConnection
- setConnectionContext

MessageReceiveHandler includes the methods and field summarized below.

| Methods | Description |
|---|---|
| doneMessageProcessing | public abstract boolean doneMessageProcessing()<br>throws AdapterException<br>This method is called after message processing is complete. |
| fetchHeader | public abstract int fetchHeader(int pEvent,<br>IConnectionContext pContext)<br>throws AdapterException<br>Parameters:<br>pEvent—int<br>pContext—IConnectionContext<br>Returns (int) the message header. |
| isHeaderComplete | public abstract boolean isHeaderComplete()<br>throws AdapterException<br>Checks the status of header reading. Returns true if the header is read in completely. This happens after fetchHeader returns STATUS_OK.<br>This method also returns true if the protocol that does not require a message header. |
| isMessageReadComplete | public abstract boolean isMessageReadComplete()<br>throws AdapterException<br>Returns true if the message has been read in completely. This happens after fetchMessage returns STATUS_OK. This method also returns true if the message was read in completely during fetchHeader phase. This can happen if the message is short. |
| onMessageWriteComplete | public void onMessageWriteComplete(boolean pSuccess)<br>Parameters:<br>pSuccess—true if the write was successful. Otherwise, false.<br>This callback is made into receive handler when the message received through this message receive handler has been written out. At this time the receive handler can update its internal state if required. By default this method does nothing. An interested MessageReceiveHandler may override this method to do interesting things. |

| Methods | Description |
|---|---|
| releaseConnection | public void releaseConnection(IAONSMessage pMessage)<br>After the message read is complete, this method Releases the underlying connection. |
| updateMessageContext | public abstract void updateMessageContext(IMessageHandler pMsgHandler)<br>throws AdapterException<br>Parameters:<br>pMessageHandler—IMessageHandler<br>Updates the context of a request messages. This method is not called for a response message<br>This method is called after the adapter has dispatched a request message and just before a dispatch worker starts executing it.<br>Adapter writers update the message context once IMessageHandler object has been created for handle request/response message. At this point, an adapter writer could update IMessageHandler with attachments that it registered at initialization. |

| Field | Description |
|---|---|
| mConnHandler | protected IConnectionReceiver mConnHandler<br>This is the connection receiver associated with this instance of MessageReceiveHandler. |

## MessageSendHandler

Extending MessageIOHandler, this class handles the send side of an AON node. Each protocol adapter must implement this class for the send side of the adapter. The class includes the methods and files summarized below.

| Methods | Description |
|---|---|
| getMessage | public final IAONSMessage getMessage()<br>Get the message associated with this handler. This method is returns the IAONSMessage object. The message may not have been read in completely. |
| initialize | public final void initialize(IAONSMessage pMessage,<br>Adapter pAdapter,<br>IConnectionContext pConnCtx,<br>IMessageHandlerpMsgHandler,<br>int pMode,<br>int pMessageType)<br>throws AdapterException<br>Parameters<br>pMessage—AON message<br>pAdapter—adapter for this handler<br>pMsgHandler—message handler<br>pMode—protocol mode for this handler<br>ADAPTER_REQUEST—mode if the handler is to make a protocol request<br>ADAPTER_RESPONSE -- mode if the handler is to make a protocol reply<br>Initializes the adapter handler. |
| reset | public void reset()<br>throws AdapterException<br>Clears the handler. |
| writeMessage | protected abstract int writeMessage(int pEvent,<br>IConnectionContext pContext)<br>throws AdapterException<br>Writes the message to the output socket channel. |

| Field | Description |
|---|---|
| mMessage | protected IAONSMessage mMessage |
| mWriteAONP | protected boolean mWriteAONP |

## OrderedSource

This abstract class (com.cisco.aons.adapter.OrderedSource) represents an adapter resource for messages that require ordered delivery. OrderedSource inherits the following methods from Source: getAdapter, getAssociateGroup, getSourceGroup, getSourceSetId, getURI, and setGroup. It also inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). In addition, OrderedSource includes the method and field summarized below.

| Method | Description |
|---|---|
| getResourceId | public final java.lang.String getResourceId()<br>Returns (string) the unique resource identifier associated with this OrderedSource. |
| isSourcePinned | public boolean isSourcePinned()<br>The default value is false for ordered source. Adapter can override this method and return true if the source has to be pinned.<br>Specified by: isSourcePinned in class Sourcee<br>Returns: boolean false |

| Field | Description |
|---|---|
| mResourceId | protected java.lang.String mResourceId<br>Unique resource identifier associated with this OrderedSource. |

## SchedulableTask

This method represents a task that runs at specified intervals. ScheduleableTask can be deregistered by calling the deregister method. SchedulableTask inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). It also includes the methods summarized below.

| Methods | Description |
|---|---|
| deregister | public final void deregister()<br>Deregisters the tasks. |
| isDeregistered | public final boolean isDeregistered()<br>Returns true if the task is deregistered, otherwise false. |

## Source

This abstract class (com.cisco.aons.adapter.Source) represents an adapter resource. A resource can be ordered or unordered. Source inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). It also includes the methods and fields summarized below.

| Methods | Description |
|---|---|
| activateSource | public final void activateSource()<br>Activates the source. |
| deactivateSource | public final void deactivateSource()<br>Deactivates the source. |
| getAdapter | public final Adapter getAdapter()<br>Returns the adapter associated with this source. |
| getAffinity | public AttributeValue getAffinity()<br>Indicates whether this source can be migrated to another blade if case of a forced yield.<br>Adapter returns: AttributeValue.NOT_PINNED (default, source can migrate) or AttributeValue.PINNED. |
| getAssociatedGroup | public final IGroup getAssociatedGroup()<br>Returns the IGroup object associated with this source. |
| getMessageType | public AttributeValue getMessageType()<br>Returns the message type consumed by this source. The default<br>implementation returns AttributeValue.UNKNOWN. An adapter must override this method and return either AttributeValue.REQUEST or AttributeValue.RESPONSE.<br>Returns: AttributeValue AttributeValue.REQUEST if the source processes request messages. Returns AttributeValue.RESPONSE if the source processes response messages. |
| getSourceGroup | public final java.lang.String getSourceGroup()<br>Returns the source group associated with this Source. |
| getSourceInfo | public final SourceInfo getSourceInfo()<br>Returns the source attributes. An adapter must fill in the necessary attributes before registering the source. Returns: SourceInfo, the source information. |
| getSourceSetId | public abstract java.lang.String getSourceSetId()<br>Returns the source set identifier associated with this Source. A source set is a logical collection of sources. For example, a queue based adapter may group together all reply-to sources into the same source set. The sources in a source set will be distributed as evenly as possible across the network nodes in a virtual cluster. |
| getURI | public abstract URI getURI()<br>Returns the URI associated with this source. |

| Methods | Description |
|---|---|
| isActive | public final boolean isActive()<br>Returns boolean true if the source is active. If not, returns false. |
| isSourcePinned | public abstract boolean isSourcePinned()<br>Indicates whether or not this source can be migrated to another blade in a virtual cluster.<br>Returns boolean true if the source is pinned, false if it is not pinned. |
| setGroup | public void setGroup(IGroup pGroup)<br>Parameters: pGroup—IGroup object to associate with this object<br>Sets the IGroup object associated with this source. |

| Fields | Description |
|---|---|
| mAdapter | protected Adapter mAdapter<br>Adapter associated with this source. |
| mGroup | protected IGroup mGroup<br>Group associated with this source. |

| Constructor | Description |
|---|---|
| Source | public Source(Adapter pAdapter)<br>Constructs a new source object and associates it with the given adapter. |

## SourceAttribute

public interface SourceAttribute

This interface defines source attributes. It includes the single field summarized below.

| Fields | Description |
|---|---|
| MessageType | static final SourceAttribute MessageType<br>When a source is queried for this attribute, it returns either "aon.adapter.message.type.request" or "aon.adapter.message.type.response" string indicating the type of message that it processes. |

## SourceInfo

public final class SourceInfo

Extending java.lang.Object, this interface provides information about a source. It includes the methods and constructors summarized below.

| Method | Description |
|---|---|
| getSourceAttribute | public java.lang.Object getSourceAttribute(SourceAttribute pAttr)<br>Parameters:<br>pAttr - SourceAttribute the source attribute<br>Returns a value of a source attribute. Returns null if there is no value. |
| setSourceAttribute | public void setSourceAttribute(SourceAttribute pAttr,<br>java.lang.Object pValue)<br>Sets a source attribute<br>Parameters:<br>pAttr - SourceAttribute the source attribute<br>pValue - Object |

| Constructor | Description |
|---|---|
| SourceInfo | public SourceInfo() |

## StandaloneAdapter

Extended from the Adapter class, this base class must be included in all standalone adapters.

StandaloneAdapter inherits the mDescriptor field and the following methods from Adapter: doInitialize, doShutdown, doShutdownInput, doShutdownOutput, soStart, getAdapterDescriptor, getAdapterManager, getAdaptername, getInteractionStyle, getType, isLoopBack, and reload.

StandalonAdapter also inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). It also includes the methods and field listed below.

| Methods | Description |
|---|---|
| getDeliveryDispatcher | public IDeliveryGroupDispatcher getDeliveryDispatcher(IDeliveryGroupCallback pCallback)<br>throws AdapterException<br>Returns a delivery dispatcher associated with this adapter. The delivery dispatcher is called only when the message is delivered in a group. The group may be used for batching or ordering. |
| getOutboxHandler | public AbstractOutboxHandler getOutboxHandler(IMessageHandler pHandler,<br>IAONSMessage pMessage,<br>int pMsgType)<br>throws AdapterException<br>Parameters:<br>pMessage—IAONSMessage<br>Returns the OutboxHandler associated with this adapter. An OutboxHandler handles the message in the outbox, sending them out of AON. A handled message can be sent out as REQUEST or RESPONSE. Adapters extend the AbstractOutboxHandler class during registration. |
| getStandaloneAdapterManager | public final IStandaloneAdapterManager getStandaloneAdapterManager()<br>Returns a standalone adapter. |
| initialize | protected void initialize(IAdapterDescriptor pDescriptor,<br>IAdapterManagerpAdapterManager)<br>throws AdapterException<br>Parameters:<br>pDescriptor—IAdapterDescriptor<br>pAdapterManager—IAdapterManager<br>Overrides:<br>initialize in class Adapter<br>Initializes the standalone adapter. |

| Field | Description |
|---|---|
| mAdapterManager | protected IStandaloneAdapterManager mAdapterManager |

## StandaloneMessageReader

Extended from PooledJob, this is the base class for implementing reading message in the context of AON thread. Standalone adapters should extend this class, implementing the readMessage method. Extended classes may override the default implementation of updateMessageContext if the message handler has to be updated before it is dispatched for execution. This method is only called for request messages.

StandaloneMessageReader inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). It also includes the methods and field summarized below.

| Methods | Description |
|---|---|
| execute | public void execute()<br>This is the execute for this class. |
| readMessage | public abstract void readMessage()<br>throws AdapterException<br>Returns the read message.<br>Derived classes should implement this method. This method enables the adapter to read messages. This method returns the message after it is completely read. |

| Field | Description |
|---|---|
| mAdapter | protected StandaloneAdapter mAdapter<br>Adapter that delivered the message. |

## UnorderedSource

This abstract class (com.cisco.aons.adapter.UnorderedSource) represents an adapter resource for messages that do not require ordered delivery.

UnorderedSource inherits the following methods from Source: getAdapter, getAssociatedGroup, getSourceGroup, getSourceSetId, getURI, and setGroup. It also inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). In addition, UnorderedSource includes the fields and methods summarized below.

| Fields | Description |
|---|---|
| mAdapter | protected Adapter mAdapter<br>Adapter associated with this Source. |
| mGroup | protected IGroup mGroup<br>Group associated with this Source. |

| Methods | Description |
|---|---|
| isSource | public boolean isSourcePinned()<br>The default return value is false for unordered source. If the source IOS is not pinned, the source can migrate to another blade in a virtual cluster. An adapter can override this method and return true if the source has to be pinned.<br>Specified by: isSourcePinned in class Source<br>Returns: boolean false |

| Constructor | Description |
|---|---|
| UnorderedSource | public UnorderedSource (Adapter pAdapter) |

# IO Package

The IO Package (**com.cisco.aons.io**) contains the IO handling interfaces and classes described in the following sections:

- Interfaces, page 3-117
- Classes, page 3-127

For more information, see Adapter Package, page 3-76, Message Package, page 3-127. Net Package, page 3-131, Utilities Package, page 3-131, Utilities Pool Package, page 3-144, and Exception Package, page 3-145.

## Interfaces

The IO package interfaces are summarized below.

### IAdapterReader

This interface defines the adapter reader for reading data from the underlying TCP stream. It includes the methods listed below.

| Methods | Description |
|---|---|
| clear | public void clear()<br>Clears the underlying buffers. |
| getAONSBuffer | public IAONSBuffer getAONSBuffer()<br>Returns IAONSBuffer, the underlying buffers. |
| getBytesRead | public int getBytesRead()<br>Returns (int) the actual bytes read from the last read operation. |
| getContext | public IConnectionContext getContext()<br>Returns the underlying connection context associated with this reader. |

| Methods | Description |
|---|---|
| readData | public int readData(int pSize)<br>throws java.io.IOException<br><br>Parameters:<br>pSize—int specifies the size of the data to read, actual bytes read could be retrieve by calling getBytesRead()<br>Reads data from the underlying channel stream. Returns the int status of the read operation:<br>0—successful operation<br>< 0—an error occurred |
| reset | public void reset()<br>Initializes the reader to read a new message. |

| Fields | Description |
|---|---|
| BUFFERS_FULL | public static final int BUFFERS_FULL<br>Indicates that internal buffers are full. |
| BUFFERS_UNAVAILABLE | public static final int BUFFERS_UNAVAILABLE<br>Indicates that buffers are unavailable. |
| CHANNEL_CLOSED | public static final int CHANNEL_CLOSED<br>Indicates that the underlying channel is closed. |
| EOF | public static final int EOF<br>End of stream. |
| OK | public static final int OK<br>Successful status. |

### IAdapterWriter

This interface defines the adapter writer for writing data to the underlying TCP stream. IAdapterWriter includes the following methods:

| Methods | Description |
|---|---|
| getContext | public IConnectionContext getContext()<br>Gets the underlying connection context associated with the writer. |

| Methods | Description |
|---|---|
| write | public long write(java.nio.ByteBuffer[] pBuffers,<br>int pOffset,<br>int pLength)<br>throws java.io.IOException<br>Parameters:<br>pBuffers—ByteBuffer[] The buffers from which bytes are to be retrieved<br>pOffset—Offset (int) within the buffer array of the first buffer from which bytes are to be retrieved; must be non-negative and no larger than srcs.length<br>pLength—Maximum number (int) of buffers to be accessed. This must be non-negative number and no larger than srcs.length - offset<br>Returns number (long) of bytes written, possibly zero This method writes a sequence of bytes to the channel from the given buffer. An attempt is made to write up to *r* bytes to the channel, where *r* is the number of bytes remaining in the buffer at the moment that this method is invoked.<br>If this method writes a byte sequence of length *n* (where 0 <= *n* <= *r)*, this byte sequence will be transferred from the buffer starting at index *p*, where *p* is the buffer's position at the moment this method is invoked. The index of the last byte written will be $p + n - 1$.<br>On return, the buffer's position will be equal to $p + n$ and its limit will not have changed. Unless otherwise specified, a write operation will return only after writing all of the *r* requested bytes.<br>Depending on their state, some channels may write only some bytes or none at all. For example, a socket channel in an unblocking mode cannot write more bytes than those that are free in the socket's output buffer. This method may be invoked at any time. However, if another thread has already initiated a write operation on this channel, this method will be blocked until the first operation is complete. |

| Methods | Description |
|---|---|
| write | public int write(java.nio.ByteBuffer pBuffer)<br>throws java.io.IOException<br>Parameters:<br>pBuffer—ByteBuffer The buffer from which bytes are to be retrieved<br>Returns (int) the number of written bytes, possibly zero.<br>This method writes a sequence of bytes to this channel from a sequence of the given buffers.<br>An attempt is made to write up to *r* bytes to this channel, where *r* is the total number of bytes remaining in the specified subsequence of the given buffer array at the moment that this method is invoked.<br>If a byte sequence of length *n* is written (where 0 <= n <= r), this method first writes up to the first srcs[offset].remaining() bytes of this sequence from buffer srcs[offset], then writes up to the next srcs[offset+1].remaining() bytes from buffer srcs[offset+1], and so forth. This process continues until the entire byte sequence is written.<br>As many bytes as possible are written from each buffer. Thus, the final position of each updated buffer, except the last updated buffer, will be equal to that buffer's limit. Unless otherwise specified, a write operation will return after all of the requested bytes are written.<br>Depending on their state, some channels may write some of the bytes or none at all. For example, a socket channel in an unblocking mode cannot write more bytes than those that are free in the socket's output buffer. This method may be invoked at any time. However, if another thread has already initiated a write operation on this channel, this method will be blocked until the first operation is complete. |

## IAONSBuffer

This interface encapsulates the native buffers. It is used to write data to the buffers, create multiple read-only views of buffers, and release them fully or partially. IAONSBuffer includes the methods listed below.

| Methods | Description |
|---|---|
| available | public int available()<br>Returns (long) the available size for write operations. |
| clear | public void clear()<br>Clears the underlying buffers for reuse. |
| createDataReader | public IDataReader createDataReader()<br>Returns a data reader. |
| getDataSize | public int getDataSize()<br>Returns (int) the data size. |
| isReleased | public boolean isReleased()<br>Returns true if the buffer is already released, otherwise false. |
| isWaitingForAllocation | public boolean isWaitingForAllocation()<br>Returns true if this buffer is waiting for new allocation. |

| Methods | Description |
|---|---|
| release | public void release()<br>Releases all buffers back to the pool. |
| release | public void release(int pIdx)<br>Releases all buffers up to the specified index. |
| size | public int size()<br>Returns the current size of allocated buffers. |
| write | public void write(byte[] b)<br>throws java.io.IOException<br>Parameters:<br>b—byte[] the data<br>Writes the specified bytes into the native buffers |
| write | public void write(byte[] b,<br>int off,<br>int len)<br>throws java.io.IOException<br>Parameters:<br>b—byte[] the data<br>off—offset (int)<br>len—length (int)<br>Writes data from the byte array to native buffers from the specified offset, up to the given length. |
| write | public void write(byte b)<br>throws java.io.IOException<br>Parameters:<br>b—byte<br>Writes a single byte to native buffers |

| Methods | Description |
|---|---|
| write | public void write(java.nio.CharBuffer pCharBuffer,<br>java.nio.charset.CharsetEncoder mEncoder)<br>throws java.io.IOException<br>Parameters:<br>pCharBuffer—CharBuffer the data<br>mEncoder—CharsetEncoder the encoder<br>Writes data from the char buffer to native buffers using the supplied charset encoding. |
| write | public int write(IAdapterReader pReader)<br>throws java.io.IOException<br>Parameters:<br>pReader—IAdapterReader the adapter reader<br>Returns (int, number of bytes) size of the extracted data. Extracts data from the adapter reader and writes it into native buffers. |

### IBufferManager

This interface defines the adapter buffer manager. A buffer manager maintains a pool of native buffers primarily used by embedded adapters to read and write data from TCP sockets. This interface provides methods for allocating buffers and increasing the size of existing allocations. Allocated buffers should be released when they are no longer in use so that they can be recycled back into the pool. IBufferManager includes the methods and fields summarized below.

| Methods | Description |
|---|---|
| getBuffer | public IAONSBuffer getBuffer(int pSize,<br>int pQualifier)<br>throws BufferException<br>Parameters:<br>pSize—requested size requested (int)<br>pQualifier—qualifier (int) indicating the processing state of the message<br>Increases the buffer by the specified size. The qualifier indicates the processing state of the message for which buffers are requested. |

| Methods | Description |
|---|---|
| getBuffer | public IAONSBuffer getBuffer(IAONSBuffer pAONSBuffer,<br>int pSize,<br>int pQualifier)<br>throws BufferException<br>Parameters:<br>pAONSBuffer—AONSBuffer the existing AONS buffer<br>pSize—size (int) increase for the buffer size.<br>pQualifier—qualifier indicating the processing state of the message<br>Increases the size of an existing AON buffer by the specified size. |

| Fields | Description |
|---|---|
| IN_BOUND | public static final int IN_BOUND<br>Specifies an inbound message. |
| IN_PROC | public static final int IN_PROC<br>Specifies a message in process. |
| OUT_BOUND | public static final int OUT_BOUND<br>Specifies an outbound message. |

### IDataReader

This interface defines a read-only view of the IAONSBuffer. It is used to extract data from native buffers. IDataReader includes the methods summarized below.

| Methods | Description |
|---|---|
| available | public int available()<br>throws java.io.IOException<br>`Returns (int) number of bytes that can be read from this data reader.` |
| close | public void close()<br>Closes the read-only view and decreases the count of open read-only views by one. |
| compact | public int compact()<br>throws java.io.IOException<br>Discards data that as been read from the underlying native buffers. The current read offset of this data reader is used to discard the data. If there are multiple read-only views, other views may go into an inconsistent state. This method should be called when there is only one read-only view of the underlying native buffers. |
| getParent | public IAONSBuffer getParent()<br>Returns the parent AON Buffer. |
| mark | public void mark()<br>Marks the current read offset so that reset call will set the read offset to this saved state. |

| Methods | Description |
|---|---|
| read | public int read(byte[] b)<br>throws java.io.IOException<br>Parameters:<br>b—byte[] the buffer into which data is read<br>Returns (int) the total number of bytes read into the buffer or returns -1 if the end of the data stream has been reached.This method reads a specified number of bytes from the input stream and stores them into the buffer array b. The number of bytes actually read is returned as an integer. I<br>If b is null, a NullPointerException is thrown. If the length of b is zero, no bytes are read and 0 is returned. Otherwise, the method attempts to read at least one byte. If no byte is available, the value -1 is returned. Otherwise, at least one byte is read and stored in b.<br>The first byte read is stored into element b[0], the next into b[1], and so on. At the most, the number of bytes read is equal to the length of b.<br>If k is the number of bytes actually read, these bytes are stored in elements b[0] through b[k-1], leaving elements b[k] through b[b.length-1] unaffected. If the first byte cannot be read for any reason other than end of file, an IOException is thrown. An IOException is thrown if the input stream has been closed. The read(b) method for class InputStream has the same effect as read(b, 0, b.length). |

| Methods | Description |
|---|---|
| read | public int read(byte[] b,<br>int off,<br>int len)<br>throws java.io.IOException<br>Parameters:<br>b—byte[] the buffer into which the data is read<br>off—start offset (int) in array b at which the data is written.<br>len—maximum number (int) of bytes to read<br>Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes but a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.<br>If b is null, a NullPointerException is thrown. If off is negative, or len is negative, or off+len is greater than the length of the array b, an IndexOutOfBoundsException is thrown. I<br>f len is zero, no bytes are read and 0 is returned. Otherwise, this method attempts to read at least one byte.<br>If no byte is available because the stream is at end of file, the value -1 is returned. Otherwise, at least one byte is read and stored into b. The first byte read is stored into element b[off], the next into b[off+1], and so on. At the most, the number of bytes read is equal to len.<br>If k is the number of bytes actually read, these bytes are stored in elements b[off] through b[off+k-1], leaving elements b[off+k] through b[off+len-1] unaffected.<br>In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.<br>If the first byte cannot be read for any reason other than end of file, an IOException is thrown. An IOException is thrown if the input stream is closed. |
| read | public int read(java.nio.CharBuffer pCharBuffer,<br>java.nio.charset.CharsetDecoder pDecoder)<br>throws java.io.IOException<br>Parameters:<br>pCharBuffer—CharBuffer the char buffer<br>pDecoder—CharsetDecoder the Decoder to apply on the raw data<br>Returns the total number (int) of bytes read into the buffer or -1 if the end of the stream has been reached<br>Reads up to size of the char buffer of data from the native buffers into a CharBuffer. An attempt is made to read as many bytes as the char buffer size but a smaller number may be read, possibly zero. The decoder decodes the data as it is read into the char buffer. The number of bytes actually read is returned as an integer. |

| Methods | Description |
|---|---|
| readByte | public byte readByte()<br>throws java.io.IOException,<br>java.io.EOFException<br>Reads a single byte. |
| reset | public void reset()<br>throws java.io.IOException<br>Repositions the data stream to the position at the time the mark method was last called on this data reader. |
| rewind | public void rewind(int len)<br>throws java.io.IOException<br>Parameters:<br>`len`—int<br>Rewinds the read offset by a specified length. |
| shift | public void shift()<br>throws java.io.IOException<br>The available bytes from current read offset to the end of the data are shifted to the offset saved by the mark method call. If there are multiple read-only views, other views may go into an inconsistent state. This method should be called when there is only one read-only view of the underlying native buffers |
| skip | public int skip(int len)<br>throws java.io.IOException<br>Parameters:<br>len—number (int) of bytes to be skipped<br>Returns the number (int) of bytes actually skipped. Skips over n bytes of data from this data reader. The skip method may end up skipping over a smaller number of bytes, possibly 0. For example, reaching the end of file before n bytes have been skipped. The actual number of bytes skipped is returned. If n is negative, no bytes are skipped. |
| truncate | public void truncate(int pLen)<br>throws java.io.IOException<br>Truncates data from the underlying native buffer by the a specified length If there are multiple read-only views, other views may go into an inconsistent state. |

| Methods | Description |
|---|---|
| update | public void update()<br>Synchronizes with parent AON Buffer. Typically, this method is called after new data is written to the underlying native buffers. |
| writeTo | public int writeTo(IAdapterWriter pWriter)<br>throws java.io.IOException<br>Parameters:<br>pWriter—IAdapterWriter adapter writer for data writing<br>Returns (int) -1 if the stream associated with adapter writer is closed, otherwise returns zero. Writes to adapter writer up to bytes available. It may not write fully in one single call. |

#### INetworkListener

This interface identifies the network that will be listened to by the adapter. It includes the methods summarized below.

| Methods | Description |
|---|---|
| getInetAddress | Returns the IP address. |
| getPort | Returns the port. |

### Classes

The IO package currently includes a single class.

#### BufferException

This class extends the AONSException class. For more information, see the description of AONSException.java in Appendix A, “AONS Common Specification.”

## Message Package

The Adapter API Message package (**com.cisco.aons.util**) includes a set of interfaces and classes that are used by custom adapters to process messages:

- Interfaces, page 3-127
- Classes, page 3-130

For more information, see Adapter Package, page 3-76, IO Package, page 3-117, Net Package, page 3-131, Utilities Package, page 3-131, Utilities Pool Package, page 3-144, and Exception Package, page 3-145.

### Interfaces

The Message package includes the interfaces summarized below.

### IAdapterMessageBuilder

An extension to message builder, this interface provides methods for creating each type of content. It includes the methods listed below.

| Methods | Description |
|---|---|
| createMapContent | Creates map content using the input content decoder. |
| createMIMEContent | Creates MIME content using the input content decoder. |
| createSOAPContent | Creates SOAP content using the input content decoder. |
| createStreamContent | Creates stream content using the input content decoder |
| createXMLContent | Creates XML content using the input content decoder |

### IContentCanonicalizer

This interface provides a mechanism for decoding and encoding each type of content. It is implemented by each content type implementation. It includes the methods summarized below.

| Methods | Description |
|---|---|
| decode | Decodes the content. |
| encode | Encodes the content. |
| getContent | Returns content associated with this interface. |
| getContentStreameable | Returns a handle that determines if the content can be streamed. |

### IContentDecoder

This interface provides a mechanism for decoding content. It includes the methods listed below.

| Methods | Description |
|---|---|
| canUnderstand | Called by the canonical content type while the message is being decoded. |
| getContentType | Returns the content type. |
| getDataReader | Returns data without applying any decoding. |
| getEncoding | Returns the encoding of the content, if any. |
| getInputStream | Returns an input stream that applies any decoding needed. |
| parse | Parses the message content and populates the canonical content type. |

### IContentEncoder

This interface provides a mechanism for encoding content. It includes the methods listed below.

| Methods | Description |
|---|---|
| canUnderstand | Called by the canonical content type as the message is decoded. |
| createDataReader | Creates a data reader from the AON buffer. |
| getContentSize | Returns content size, or “–1” if size is unavailable. |

| | |
|---|---|
| getContentType | Returns the content type. |
| getEncoding | Returns a nonzero value if encoding is needed or if the adapter cannot transfer the AON buffer. |
| getOutputStream | Returns the output stream if the adapter cannot transfer the AON buffer or if some encoding is needed. |
| serialize | Serializes canonical content type into message content. |
| setAONSBuffer | Sets the AON buffer that contains the content. |
| setContentSize | Sets the content size. |
| setContentType | Sets the content type. |
| setEncoding | Sets the encoding. |

### IContentStreamable

This interface provides a mechanism to stream content. This enables the adapter to read content as a stream instead of all at once. If an adapter does not support this feature, it should return "false" for the method isStreameable. This interface includes the methods listed below.

| Methods | Description |
|---|---|
| getContentSize | Returns the content size, or "–1" if size is unavailable. |
| isReadComplete | Returns "true" if the message is completely read, otherwise "false." |
| isStreamable | Returns "true" if an adapter can support content streaming. It returns "false" if the adapter cannot support streaming or the message size is small. |
| reset | Clears the internal buffers to read the next chunk of data. |

### IMessageHandler

This interface defines a message handler. It includes the methods summarized below.

| Methods | Description |
|---|---|
| getAttachment | Gets the attachment. |
| getAttachment | Gets the attachment. |
| getFlowType | Gets the type. |
| getMessage | Returns the message associated with this handler. |
| setAttachment | |
| setAttachment | |
| updateState | Updates the message handler state from the successful state. |
| updateStateError | |

### IMessageHandlerAttachment

This interface implements objects that are attached to IMessageHandler. Objects attached to MessageHandler exist via a response-request unless they are removed.

### IMessageHandlerCallback

The interfaces sued for message callbacks.

### IMessageWriteCompleteCallback

This interface is used for message write complete callbacks.

## Classes

The Message package classes are summarized below.

### DefaultContentDecoder

This class represents the default decoder. It includes the methods listed below.

| Methods | Description |
|---|---|
| canUnderstand | Returns “false,” by default. |
| getContentSize | Returns content size (integer). |
| getContentType | Returns the content type (string). |
| getDataReader | Returns the data reader. |
| getEncoding | Returns content encoding. |
| getInputStream | Constructs an input stream from the data reader. |
| isReadComplete | Default implementation returns “true.” |
| isSteamable | Default implementation returns “false.” |
| parse | Null implementation. |
| reset | Null implementation. |

### DefaultContentEncoder

This class represents the default encoder. It includes the methods listed below.

| Methods | Description |
|---|---|
| canUnderstand | Returns “false,” by default. |
| createDataReader | Returns a data reader. |
| getContentSize | Returns the content size. |
| getContentType | Returns the content type of the message. |
| getEncoding | Returns zero, by default, if not set. |
| getOutputStream | Returns the output stream to which a canonical content type will write. |
| serialize | Null implementation. |
| setAONSBuffer | Sets the AON buffer. |
| setContentSize | Sets the content size. |
| setContentType | Sets the message content type. |
| setEncoding | Sets the message encoding. |

# Net Package

The Net package (com.cisco.aons.net) contains one class URI. For more information, see Adapter Package, page 3-76, IO Package, page 3-117, Message Package, page 3-127. Utilities Package, page 3-131, Utilities Pool Package, page 3-144, and Exception Package, page 3-145.

## URI

public class URI

Implementing java.lang.Cloneable, the class is used when a URI is encoded or decoded. It assists with query string formatting. Unlike java.net.URL, this class does not provide communications ability.

By default, this class uses ISO_8859_1 encoding for % encoded characters. This may be overridden with the org.mortbay.util.URI.charset system property.

# Utilities Package

The Utilities package (**com.cisco.aons.util**) contains the byte buffer processing classes. It includes one interface, a set of classes, and one exception. These package elements are summarized in the following sections.

For more information, see Adapter Package, page 3-76, IO Package, page 3-117, Message Package, page 3-127. Net Package, page 3-131, Utilities Pool Package, page 3-144, and Exception Package, page 3-145

## Interface DomainReader

public interface DomainReader

This interface can be used by clients to get the information about domains. Each domain has associated descriptions and properties. DomainReader can be used with AONS extension mechanisms Adapter, Custom Bladelet, Transformation, and others. The interface has the methods summarized below.

| Methods | Description |
|---|---|
| createDomainReader | DomainReader createDomainReader(java.lang.String domainName)<br>throws DomainException<br>Creates a new domain reader with the specified name<br>Parameters: domainName - String DomainName to access<br>Throws: DomainException if an error occurs. |
| getAttrNames | java.util.Set getAttrNames()<br>throws DomainException<br>Returns all attribute names defined in the set<br>Returns: Set the attributes names as set<br>Throws: DomainException if an error occurs. |

| Methods | Description |
|---|---|
| getDomains | java.util.Set getDomains()<br>throws DomainException<br>Returns all the domains with the given profile<br>Returns: All the domain names as a set<br>Throws: DomainException if an error occurs. |
| getSet | java.util.HashMap getSet(java.lang.String setName)<br>throws DomainException<br>Returns all the information which is defined in the set<br>Parameters:<br>setName - String gets all the information given Set name as key<br>Returns: HashMap Key is attribute name. Value is List of values.<br>Throws: DomainException if an error occurs. |
| getSetsForDomain | java.util.Set getSetsForDomain()<br>throws DomainException<br>Returns all sets that are defined in the domain<br>Returns: Set All keys for the sets in the domain returned<br>Throws: DomainException if an error occurs. |
| getValues | java.util.List getValues(java.lang.String setName,<br>java.lang.String attrName)<br>throws DomainException<br>Returns all attribute values for a given attribute<br>Parameters:<br>setName - String key for Property Set<br>attrName - String key for Attribute Name<br>Returns: Lists all values for the attribute as List.<br>Throws: DomainException if an error occurs. |

| Methods | Description |
|---|---|
| setDomain | void setDomain(java.lang.String domainName)<br>throws DomainException<br>Sets the domain for access<br>Parameters: domainName - String DomainName to access<br>Throws: DomainException if an error occurs. |
| setProfile | void setProfile(java.lang.String pType,<br>java.lang.String pName)<br>throws DomainException<br>Sets the profile for the DomainReader<br>Parameters:<br>pType - String example values are AONSGlobalPolicy, AONSNetworkNodePolicy<br>pName - String name of the profile<br>Throws: DomainException if parameters are invalid. |

## ByteBufferArrayInputStream

This I/O utility class wraps an IDataReader interface. It provides all standard I/O methods. The class includes the methods and constructors summarized below.

| Methods | Description |
|---|---|
| available | public int available()<br>throws java.io.IOException<br>Returns the number of bytes that can be read (or skipped) from this input stream by the next caller of a method for this input stream.<br>The next caller could be the same thread or another thread. The available method for class InputStream always returns 0. Returns: Number of bytes that can be read from this input stream.<br>Overrides: available in class java.io.FilterInputStream<br>Throws: java.io.IOException if an I/O error occurs. |
| close | public void close()<br>throws java.io.IOException<br>Closes this input stream and releases any system resources associated with the stream.<br>Specified by: close in interface java.io.Closeable<br>Overrides: close in class java.io.FilterInputStream<br>Throws: java.io.IOException if an I/O error occurs. |
| getDataReader | public final DataReader getDataReader()<br>Returns: IDataReader, the underlying data reader. |

| Methods | Description |
|---|---|
| mark | public void mark(int pReadLimit)<br>Marks the current position in this input stream. A subsequent call to the reset method repositions this stream to the last marked position so subsequent reads reread the same bytes.ReadLimit arguments tell this input stream to allow that number of bytes to be read before the mark position is invalidated.<br>If the method markSupported returns true, the stream remembers all the bytes read after the call to mark and is ready to supply the same bytes again if the method reset is called. However, the stream does not have to remember any data at all if more than ReadLimit bytes are read from the stream before reset is called.<br>Overrides: mark in class java.io.FilterInputStream<br>Parameters:<br>pReadLimit - int readlimit - maximum limit of bytes that can be read before the mark position is invalid. |
| markSupported | public boolean markSupported()<br>Checks to determine whether or not this input stream supports the mark and reset methods. The markSupported method of this class always returns true.<br>Overrides: markSupported in class java.io.FilterInputStream<br>Returns: boolean |
| read | public final int read()<br>throws java.io.IOException<br>Reads the next byte of data from the input stream. The value byte is returned as an int in the range 0 to 255. If no byte is available because the end of the stream has been reached, the value -1 is returned.<br>Overrides: read in class java.io.FilterInputStream<br>Returns:<br>int the next byte of data, or -1 if the end of the stream is reached<br>Throws: java.io.IOException if an I/O error occurs. |

| Methods | Description |
|---|---|
| read | public final int read(byte[] b)<br>throws java.io.IOException<br>Reads a number of bytes from the input stream and stores them into the buffer array b. The number of bytes actually read is returned as an integer. The first byte read is stored into element b[0], the next one into b[1], and so on.<br>The maximum number of bytes read is equal to the length of b. Let k be the number of bytes actually read; these bytes will be stored in elements b[0] through b[k-1], leaving elements b[k] through b[b.length-1] unaffected. If the first byte cannot be read for any reason other than end of file, an IOException is thrown. This exception is thrown if the input stream has been closed. The read(b) method for class InputStream has the same effect as: read(b, 0, b.length)<br>Overrides: read in class java.io.FilterInputStream<br>Parameters:<br>b - byte[] the buffer into which the data is read.<br>Returns: int the total number of bytes read into the buffer, or -1 is there is no more data because the end of the stream has been reached<br>Throws: java.io.IOException if an I/O error occurs. |

| Methods | Description |
|---|---|
| read | public final int read(byte[] b,<br>int off,<br>int len)<br>throws java.io.IOException<br>Reads up to len bytes of data from the input stream into an array of bytes. Although it attempts to read as many as len bytes, a smaller number may be read, possibly zero. The number of bytes actually read is returned as an integer.<br>If b is null, a NullPointerException is thrown. If off is negative, or len is negative, or off+len is greater than the length of the array b, an IndexOutOfBoundsException is thrown.<br>If len is zero, no bytes are read and 0 is returned. Otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned. Otherwise, at least one byte is read and stored into b.<br>The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len.<br>Let k be the number of bytes actually read; these bytes will be stored in elements b[off] through b[off+k-1], leaving elements b[off+k] through b[off+len-1] unaffected. In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.<br>If the first byte cannot be read for any reason other than end of file, then an IOException is thrown. This exception is thrown if the input stream has been closed.<br>Overrides: read in class java.io.FilterInputStream<br>Parameters:<br>b - byte[] the buffer into which the data is read<br>off - int the start offset in array b at which the data is written<br>len - int the maximum number of bytes to read<br>Returns: int the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached<br>Throws: java.io.IOException if an I/O error occurs |

| Methods | Description |
|---|---|
| reset | public void reset()<br>throws java.io.IOException<br>Repositions this stream to the position at the time the mark method was last called on this input stream.<br>Overrides: reset in class java.io.FilterInputStream<br>Throws: java.io.IOException if this stream has not been marked or if the mark is invalid. |
| skip | public long skip(long pLen)<br>throws java.io.IOException<br>Skips over and discards n bytes of data from this input stream. The skip method may, for a variety of reasons, end up skipping over some smaller number of bytes, possibly 0. This may result from any of a number of conditions; reaching end of file before n bytes have been skipped is only one possibility. The actual number of bytes skipped is returned. If n is negative, no bytes are skipped.<br>Overrides:<br>skip in class java.io.FilterInputStream<br>Parameters:<br>pLen - long the number of bytes to be skipped<br>Returns: long the actual number of bytes skipped.<br>Throws: java.io.IOException if an I/O error occurs |

| Constructors | Description |
|---|---|
| ByteBufferArrayInputStream | public ByteBufferArrayInputStream(IDataReader pReader)<br>Default constructor<br>Parameters: pReader - IDataReader, the data reader. |

## ByteBufferArrayOutputStream

This I/O utility class defines an OutputStream interface to write to the AONJ buffer. It includes the methods, field, and constructors summarized below.

| Methods | Description |
|---|---|
| close | public void close()<br>throws java.io.IOException<br>Closes this output stream and releases any system resources associated with this stream<br>Specified by: close in interface java.io.Closeable<br>Overrides: close in class java.io.FilterOutputStream<br>Throws: java.io.IOException if an I/O error occurs |
| flush | public void flush()<br>throws java.io.IOException<br>Specified by: flush in interface java.io.Flushable<br>Overrides: flush in class java.io.FilterOutputStream<br>Throws: java.io.IOException |
| getAONSBuffer | public IAONSBuffer getAONSBuffer()<br>Returns IAONSBuffer, the underlying AONS buffer. |
| isClosed | public boolean isClosed()<br>Returns true if the output stream is close. Otherwise, returns false.<br>Returns: boolean |
| write | public void write(int i)<br>throws java.io.IOException<br>Writes the specified byte to this output stream. One byte is written to the output stream. The byte to be written is the eight low-order bits of the argument b. The 24 high-order bits of b are ignored<br>Overrides: write in class java.io.FilterOutputStream<br>Parameters:<br>i - int the byte<br>Throws: java.io.IOException if an I/O error occurs. This exception may be thrown if the output stream is closed. |

| Methods | Description |
|---|---|
| write | public void write(byte[] b)<br>throws java.io.IOException<br>Writes b.length bytes from the specified byte array to this output stream. Write(b) should have exactly the same effect as the call write(b, 0, b.length).<br>Overrides: write in class java.io.FilterOutputStream<br>Parameters:<br>b - byte[] the data<br>Throws: java.io.IOException if an I/O error occurs |
| write | public void write(byte[] b,<br>int off,<br>int len)<br>throws java.io.IOException<br>Writes len bytes from the specified byte array starting at offset off to this output stream. For write(b, off, len), some of the bytes in the array b are written to the output stream in order. Element b[off] is the first byte written and b[off+len-1] is the last written by this operation. If b is null, a NullPointerException is thrown. If off is negative, or len is negative, or off+len is greater than the length of the array b, then an IndexOutOfBoundsException is thrown.<br>Overrides: write in class java.io.FilterOutputStream<br>Parameters:<br>b - byte[] the data<br>off - int the start offset in the data<br>len - int the number of bytes to write<br>Throws: java.io.IOException if an I/O error occurs. This exception is thrown if the output stream is closed. |

| Fields | Description |
|---|---|
| mClose | protected boolean mClose |

| Constructors | Description |
|---|---|
| ByteBufferArrayOutputStream | public ByteBufferArrayOutputStream(BufferManager pBufferMgr,<br>int pInitialCapacity,<br>int pIncrementSize)<br>throws java.io.IOException<br>Takes a buffer manager to create new AONS buffer. Initial capacity of the AONS buffer and increment size<br>Parameters:<br>pBufferMgr - IBufferManager<br>pInitialCapacity - int<br>pIncrementSize - int<br>Throws: java.io.IOException |
| ByteBufferArrayOutputStream | public ByteBufferArrayOutputStream(BufferManager pBufferMgr,<br>IAONSBuffer pAONSBuffer,<br>int pIncrementSize)<br>Takes a buffer manager and AONS buffer and increment size as parameters.<br>Parameters:<br>pBufferMgr - IBufferManager<br>pAONSBuffer - IAONSBuffer<br>pIncrementSize - int |

## ByteBufferArrayWriter

This I/O utility class defines a writer interface to write to the AONS buffer. The class inherits the lock field from class.java.io.Writer and includes the methods and constructors summarized below.

| Methods | Description |
|---|---|
| close | public void close()<br>throws java.io.IOException<br>Releases the underlying AONS buffer.<br>Specified by: close in interface java.io.Closeable<br>Specified by: close in class java.io.Writer<br>Throws: java.io.IOException if an I/O error occurs. |
| flush | public void flush()<br>throws java.io.IOException<br>Null implementation.<br>Specified by: flush in interface java.io.Flushable<br>Specified by: flush in class java.io.Writer<br>Throws: java.io.IOException if an I/O error occurs. |
| getAONSBuffer | public IAONBuffer getAONSBuffer()<br>Returns: IAONSBuffer (AON Buffer). |
| write | public void write(int c)<br>throws java.io.IOException<br>Writes a single character. The character to be written is contained in the 16 low-order bits of the given integer value; the 16 high-order bits are ignored<br>Overrides: write in class java.io.Writer<br>Parameters:<br>c - int<br>Throws: java.io.IOException if an I/O error occurs. |
| write | public void write(char c)<br>throws java.io.IOException<br>Writes a single character.<br>Parameters:<br>c - char<br>Throws: java.io.IOException if an I/O error occurs. |

| Methods | Description |
|---|---|
| write | public void write(char[] c,<br>int off,<br>int len)<br>throws java.io.IOException<br>Writes a portion of an array of characters.<br>Specified by: write in class java.io.Writer<br>Parameters:<br>c - char[] Array of characters<br>off - int Offset from which to start writing characters<br>len - int Number of characters to write<br>Throws: java.io.IOException if an I/O error occurs. |
| write | public void write(java.lang.CharSequence c)<br>throws java.io.IOException<br>Writes data from a char sequence.<br>Parameters:<br>c - CharSequence<br>Throws: java.io.IOException if an I/O error occurs. |
| write | public void write(java.nio.CharBuffer c)<br>throws java.io.IOException<br>Writes data from the char buffer.<br>Parameters:<br>c - CharBuffer<br>Throws: java.io.IOException if an I/O error occurs. |

| Methods | Description |
|---|---|
| write | public void write(java.lang.String str)<br>throws java.io.IOException<br>Writes a string.<br>Overrides: write in class java.io.Writer<br>Parameters:<br>str - String to be written<br>Throws: java.io.IOException if an I/O error occurs. |
| write | public void write(java.lang.String str,<br>int off,<br>int len)<br>throws java.io.IOException<br>Writes a portion of a string.<br>Overrides: write in class java.io.Writer<br>Parameters:<br>str - String<br>off - int Offset from which to start writing characters<br>len - int If an I/O error occurs<br>Throws: java.io.IOException if an I/O error occurs. |

| Methods | Description |
|---|---|
| ByteBufferArrayWriter | public ByteBufferArrayWriter(BufferManager pBufferMgr,<br>int pInitialCapacity,<br>int pIncrementSize,<br>java.nio.charset.CharsetEncoder pEncoder)<br>throws BufferException<br>Takes a buffer manager to create new AONS buffer. It has the initial capacity of the AONS buffer and increment size<br>Parameters:<br>pBufferMgr - IBufferManager<br>pInitialCapacity - int<br>pIncrementSize - int<br>pEncoder - CharsetEncoder<br>Throws: BufferException |
| ByteBufferArrayWriter | public ByteBufferArrayWriter(BufferManager pBufferMgr,<br>int pInitialCapacity,<br>int pIncrementSize,<br>java.lang.String pEncoding)<br>throws BufferException<br>Takes a buffer manager and the AONS buffer and increment size as parameters.<br>Parameters:<br>pBufferMgr - IBufferManager<br>pInitialCapacity - int<br>pIncrementSize - int<br>pEncoding - String<br>Throws: BufferException |

# Utilities Pool Package

The AON Utilities Pool package (com.cisco.aons.util.pool) includes the single class described in the next section. For more information, see Adapter Package, page 3-76, IO Package, page 3-117, Message Package, page 3-127. Net Package, page 3-131, Utilities Package, page 3-131, and Exception Package, page 3-145

## PooledJob

public abstract class PooledJob

Extending java.lang.Object, PooledJob includes the fields, methods, and constructors summarized below.

| Fields | Description |
|---|---|
| mIsJobAlreadyPooled | public boolean mIsJobAlreadyPooled |
| mNext | public PooledJob mNext |
| mPooledExecution | public boolean mPooledExecution |

| Methods | Description |
|---|---|
| execute | public void execute()<br>This method is invoked when a pooled job becomes runnable i.e pulled from the queue for execution. |
| Methods inherited from class java.lang.Object | clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait |
| handleOutOfMemory | public void handleOutOfMemory() |
| getPriority | public int getPriority()<br>Returns the priority of the PooledJob |
| setPriority | public void setPriority(int pPriority)<br>Sets the priority of thr PooledJob<br>Parameters:<br>pPriority - QOS enhancements |

| Constructor | Description |
|---|---|
| PooledJob | public **PooledJob**() |
| | |
| | |

# Exception Package

The AONS Exception package (com.cisco.aons.exception) provides exception handling. The classes and methods are briefly summarized in the following sections:

- ExceptionType, page 3-146
- AONSException, page 3-146
- AONSRuntimeException, page 3-146
- ExtServiceException, page 3-146
- InitializationException, page 3-146

For more information, see Adapter Package, page 3-76, IO Package, page 3-117, Message Package, page 3-127. Net Package, page 3-131, Utilities Package, page 3-131, andUtilities Pool Package, page 3-144.

## ExceptionType

The ExceptionType class (com.cisco.aons.exception.ExceptionType) is used to get the exception type. For more information, see “ExceptionType” in “Appendix A. AONS Common Specification.”

## AONSException

The AONSException class (com.cisco.aons.exception.AONSException) provides exception services. For more information, see For more information, see “AONSException” in “Appendix A. AONS Common Specification.”

## AONSRuntimeException

Extending java.lang.RuntimeException, the AONSRuntimeException class (com.cisco.aons.exception. AONRuntimeException) provides runtime exception services. For more information, see “AONSRuntimeException” in “Appendix A. AONS Common Specification.”

## ExtServiceException

Extending java.lang.Exception, AON ExtServiceException indicates exception conditions that service clients are expected to catch. The class inherits methods from class.java.lang.Throwable and class.java.lang. Object. For more information, see “ExtServiceException” in “Appendix A. AONS Common Specification.”

## InitializationException

Extending AONSException, the InitializationException class com.cisco.aons.exception.AONSException) inherits methods from class.java.lang.Throwable and class.java.lang. Object. For more information, see “InitializationException” in “Appendix A. AONS Common Specification.”

CHAPTER 4

# AON Optimization - Fastpath

AON enables optimization of some Policy Execution Plans (PEPs), using a specialized process called Fastpath. With Fastpath, AON can optimally process messages that are classified with PEPs and meet specific conditions. This optimization substantially increases throughput.

Only PEPs meeting one of the following conditions can be processed using Fastpath (also referred to as optimized processing):

- incorporating only the supplied bladelets Send, Validate, and SetDestination.
- incorporating any the above supplied bladelets and/or custom bladelets that are developed specifically for Fastpath processing.

For a message to be processed by Fastpath optimized PEPs using bladelets optimized for Fastpath processing, the associated ACL/Classifiers must be appropriately configured. For details, see the configuration section in the *AON Installation and Administration Guide* and the description of optimization in the *AON Development Studio Guide*.

# Custom Bladelet for Fastpath Processing

You can choose a template from those included in the SDK, to develop your own Fastpath optimized custom bladelets. For more information, see Sample Fastpath Custom Bladelet, page 4-4 and Developing a Custom Bladelet for Fastpath Processing, page 4-1.

## Developing a Custom Bladelet for Fastpath Processing

You can develop a custom bladelet for incorporation into PEPs that qualify for Fastpath processing. In most cases, bladelets for Fastpath processing will conform to the following conventions:

- Two versions are created for each custom bladelet, one in C++ and one in Java (The C++ version is referred to as the Fastpath custom bladelet).

  The SDK provides the framework and base classes required for bladelet development.

- If the bladelet developer does not want conventional processing for the bladelet functions, the Java version of the bladelet can simply throw *BladeletException* when it is invoked.

Follow the steps listed below.

**Step 1** **Write the custom bladelet code.**

1.1 Using a text editor, write code that, at a minimum:

- Follows the example in the sample file LogBladelet.cpp, page 4-4.
- Includes a subclass extended from BladeletDefinition. See the sample in LogBladeletDefinition.cpp, page 4-5.

1.2 Using a text editor, write the bladelet code in Java.

All AON custom bladelets must conform to the AON bladelet schema. See Using the AON Bladelet Schema, page 2-17 for a detailed description of this requirement.

**Step 2** **Compile the custom bladelet code**.

Follow the makefile sample shown in makefile.sample, page 4-8. A Fastpath custom bladelet should be built using Red Hat Enterprise Linux 3. If you do not have Red Hat Enterprise Linux 3, use GCC3.2.3 and link against glibc 2.3.2 and linux threads.

**Step 3** **Package the files**.

Use the AON Development Studio (ADS) to package the custom bladelet files as a scar file. The package could contain:

- A native shared object that contains the bladelet implementation.
- A Java archive file (.jar) that contains the Java version of the bladelet.
- Additional Java archive files (.jar) and share objects that are required by the implementations.
- The Fastpath custom bladelet's information file (bladelet-info.xml) must contain the following attribute settings:

fastpathEnabled = true

fastpathCompatibilityEvalClass = See the example below.

```
<bladelet allowInException="false"
bladeletClass="com.cisco.aons.bladelet.core.MessageSend"
bundle="com.cisco.aons.bladelet.v1.ForwardRequest" categoryKey="routing.category.key"
displayNameKey="ForwardRequest.bladelet.name"
faspathCompatibilityEvalClass="com.cisco.aons.fastpath.compatibilityevaluator.rulesv12.For
wardRequestBladeletCheck" fastpathEnabled="true" name="ForwardRequest:1"
```

For the packaging steps, see Packaging the Custom Bladelet, page 2-10.

**Step 4** **Using AMC, Upload and Register the custom bladelet**

This action uploads, register, and deploys the custom bladelet package to the AMC. The newly registered package (containing the new custom bladelet) is listed in AMC and ready for use. If the registered package contains a custom bladelet, the system notifies AON about the new bladelet. Follow the steps listed in Uploading the Custom Bladelet to AMC, page 2-15.

For a description of the sample Fastpath enabled custom bladelet, see Sample Fastpath Custom Bladelet, page 4-4. For additional information, see the *AON Installation and Administration Guide.*

# Fastpath Custom Bladelet SDK Files

The AON software development kit (SDK) provides all files and support software necessary for the developer to create new custom bladelets. When the compressed SDK file (.tar.gz ) is expanded, it creates all necessary folders, header files, and so on. These include a folder (include/bdksample) that contains a sample Fastpath Log bladelet.

To learn how to create a Fastpath custom bladelet, see Developing a Custom Bladelet for Fastpath Processing, page 4-1.

The SDK includes the header files (defining base classes) listed below.

- alloc_specialization.h—Contains an internal interface: class template specialization for reference counting and allocator adapter for per-thread STL buffers.
- BaseParameter.h—Base class of the various parameter types.
- Bladelet.h—Abstract base class for instances of Fastpath custom bladelets.
- BladeletContext.h—Base class for metadata for all instances of a Fastpath custom bladelet.
- BladeletDefinition.h—This is the initial setup up for developing Fastpath custom bladelets.
- Command.h—Base class for the command pattern.
- config.h—Optimization-internal configuration interface.
- DataBlock.h—Defines a block of data.
- FlowContext.h—Defines data in a PEP instance that depends on the message.
- FlowContextItem.h—Contains extra items in a PEP context.
- IntParameter.h—Contains the class for representing integer parameters.
- ListParameter.h—Contains the class for representing list parameters.
- Lock.h—Contains the external interface describing a lock used in mutual exclusion.
- lock.h—Contains the internal interface describing a lock used in mutual exclusion.
- Logging.h—Contains the internal interface used in PEP optimization logging.
- MapParameter.h—Contains the class for representing map parameters.
- mempool.h—Internal interface describing parameters for per-thread memory pools.
- MessageContext.h—Contains metadata about the message. It returns a data block view of the message content, source address, destination address, and a property map.
- MessageContextBuilder.h—Contains interfaces to build metadata about the message.
- MessageContextTypes.h—Describes the types of message context for a message.It defines a data block view of the message type, the source address, and a property map.
- ParameterList.h—Contains metadata about the message.
- StringParameter.h—Class for representing string parameters.
- syshdr.h—Contains the internal optimization interfaces for system-specific features.
- ThrContextFCItem.h—Contains extra items in a PEP context.
- thread.h—Internal optimization interface to describe various aspects of PEP optimization threading.

The binary form of each file is supplied in the AON SDK.

For examples of Fastpath custom bladelet files, see Sample Fastpath Custom Bladelet, page 4-4.

# Sample Fastpath Custom Bladelet

The AON SDK includes a sample Fastpath Log bladelet and the needed support software. The sample includes the files listed below.

- LogBladelet.cpp—C++ source file for the Log bladelet.
- LogBladeletContext.cpp—C++ source file for the Log bladelet context.
- LogBladeletDefinition.cpp—C++ source file for the Log bladelet definition.
- LogBladelet.h—Header file for the custom Log bladelet.
- LogBladeletContext.h—Header file for the Log bladelet context.
- LogBladeletDefinition.h—Header file for the Log bladelet definition.
- makefile.sample—Makefile for the custom bladelet.

For more information, see Sample CustomLog Bladelet-Info File, page 2-46.

You create a parallel set of files to develop a new Fastpath custom bladelet. To learn how to create your own Fastpath custom bladelet, see Developing a Custom Bladelet for Fastpath Processing, page 4-1.

## LogBladelet.cpp

```
#include "LogBladelet.h"
#include "FlowContext.h"
#include "DataBlock.h"
#include "Logging.h"
#include <string>

LogBladelet::LogBladelet() {}

LogBladelet::~LogBladelet() {}

LogBladelet::LogBladelet(LogBladeletContext* lbc, FlowContext& fc)
{
    mBldCtx = lbc;
    mFlowCtx = &fc;
}

int LogBladelet::execute()
{
    string text = mBldCtx->getCustomText();

    aon_logInfo(__FILE__, __LINE__, __FUNCTION__, "Log Bladelet Message: %s",
text.c_str());

    return (BladeletResult::R_SUCCESS);
}
```

## LogBladeletContext.cpp

```
#include "LogBladeletContext.h"
#include "LogBladeletDefinition.h"

LogBladeletContext::LogBladeletContext(BladeletDefinition *blDef):
    mBlDef(blDef) {}
```

```
LogBladeletContext::~LogBladeletContext() {}

LogBladeletContext::LogBladeletContext (const LogBladeletContext& rhs):
    mBlDef(rhs.mBlDef), mCustomText(rhs.mCustomText) {}

BladeletDefinition* LogBladeletContext::getBladeletDefinition()
{
    return (mBlDef);
}

const string & LogBladeletContext::getCustomText()
{
    return (mCustomText);
}

void LogBladeletContext::setCustomText(const char * text)
{
    mCustomText = text;
}
```

# LogBladeletDefinition.cpp

```
#include "LogBladeletDefinition.h"
#include "LogBladeletContext.h"
#include "StringParameter.h"
#include "IntParameter.h"
#include "LogBladelet.h"
#include "Logging.h"

// static constant definitions
const string LogBladeletDefinition::PARAM_CUSTOM_TEXT = "CustomText";

DEFINE_BLADELET_INIT(customlog, LogBladeletDefinition)

LogBladeletDefinition::LogBladeletDefinition () {}

LogBladeletDefinition::~LogBladeletDefinition () {}

const char *LogBladeletDefinition::info (int flags) const
{
    if (flags & BladeletDefinitionFlags::FL_NAME) {
        return ("CustomLogBladelet"); // Used in .far file
    }
    if (flags & BladeletDefinitionFlags::FL_VERBOSE) {
        return ("Custom Log Bladelet v1.0"); // The version string for logging
    }

    return ("Log");
}

BladeletDefinitionResult::result_t LogBladeletDefinition::onLoad ()
{
    return (BladeletDefinitionResult::R_SUCCESS);
}

BladeletDefinitionResult::result_t LogBladeletDefinition::onUnload ()
{
    return (BladeletDefinitionResult::R_SUCCESS);
}

BladeletDefinitionResult::result_t LogBladeletDefinition::flowRegistered (
    BladeletContext *blCtx, int flags)
```

```
{
    return (BladeletDefinitionResult::R_SUCCESS);
}

BladeletDefinitionResult::result_t LogBladeletDefinition::flowUnregistered (
    BladeletContext *blCtx, int flags)
{
    return (BladeletDefinitionResult::R_SUCCESS);
}

/* createBladeletContext: parse parameters, parameter sanity checking and
    create LogBladeleContext */
BladeletContext* LogBladeletDefinition::createBladeletContext
    (const ParameterList &paramList)
{
    LogBladeletContext* ctx = new LogBladeletContext(this);
    StringParameter* strp = NULL;

    if ((strp = dynamic_cast<StringParameter*>
            (paramList.getElementByName(LogBladeletDefinition::PARAM_CUSTOM_TEXT)))){
        string str = strp->getElement();

        aon_logInfo(__FILE__, __LINE__, __FUNCTION__,
            "Log Bladelet: received parameter (%s=\"%s\").",
            LogBladeletDefinition::PARAM_CUSTOM_TEXT.c_str(),
            str.c_str());

        ctx->setCustomText(str.c_str());
    }

    return (ctx);
}

BladeletContext *LogBladeletDefinition::createBladeletContext (
    const BladeletContext& rhs)
{
    return (new LogBladeletContext(dynamic_cast<const LogBladeletContext&>(rhs)));
}

Bladelet* LogBladeletDefinition::createBladelet (BladeletContext *blCtx,
    FlowContext& flCtx)
{
    LogBladelet* bladelet = NULL;
    LogBladeletContext* svCtx = NULL;

    if ((svCtx=dynamic_cast<LogBladeletContext*>(blCtx))) {
        // Optimization: Could take out of pool here!
        bladelet = new LogBladelet(svCtx, flCtx);
    }

    return (bladelet);
}

void LogBladeletDefinition::destroyBladelet (Bladelet *b)
{
    // Optimization: Could reinsert into pool here!
    delete b;
}
```

# LogBladelet.h

```
#ifndef _LOGBLADELET_H_
#define _LOGBLADELET_H_

#include "Bladelet.h"
#include "FlowContext.h"
#include "LogBladeletContext.h"

using namespace std;

class LogBladelet: public Bladelet
{
public:
    LogBladelet();
    ~LogBladelet();
    LogBladelet(LogBladeletContext* cbc, FlowContext& fc);
    int execute();

private:
    LogBladeletContext* mBldCtx;
    FlowContext* mFlowCtx;
};

#endif //_LOGBLADELET_H_
```

# LogBladeletContext.h

```
#ifndef _LOGBLADELETCONTEXT_H_
#define _LOGBLADELETCONTEXT_H_

#include "BladeletContext.h"

using std::string;

class LogBladeletContext: public BladeletContext {
    friend class LogBladeletDefinition;
public:
    ~LogBladeletContext();
    LogBladeletContext(BladeletDefinition *blDef);
    LogBladeletContext(const LogBladeletContext& rhs);
    virtual BladeletDefinition *getBladeletDefinition();
    const string & getCustomText();
    void setCustomText(const char *);
private:
    LogBladeletContext();   // Prevent from being called
    BladeletDefinition *mBlDef;
    string mCustomText;
};

#endif //_LOGBLADELETCONTEXT_H_
```

# LogBladeletDefinition.h

```
#ifndef _LOGBLADELETDEFINITION_H_
#define _LOGBLADELETDEFINITION_H_

#include "BladeletDefinition.h"
```

```
# ifdef  __cplusplus
extern "C" {
# endif

DECLARE_BLADELET_INIT(customlog)

# ifdef  __cplusplus
}
# endif

class LogBladeletDefinition: public BladeletDefinition
{
DECLARE_BLADELET_INIT_FRIEND(customlog)

public:
    ~LogBladeletDefinition();
    virtual const char *info(int flags) const;
    virtual BladeletDefinitionResult::result_t onLoad();
    virtual BladeletDefinitionResult::result_t onUnload();
    virtual BladeletDefinitionResult::result_t flowRegistered(
        BladeletContext *blCtx, int flags);
    virtual BladeletDefinitionResult::result_t flowUnregistered(
        BladeletContext *blCtx, int flags);
    virtual BladeletContext* createBladeletContext(const ParameterList &paramList);
    virtual BladeletContext* createBladeletContext(const BladeletContext& rhs);
    virtual Bladelet* createBladelet(BladeletContext *blCtx, FlowContext &flCtx);
    virtual void destroyBladelet(Bladelet *b);

private:
    LogBladeletDefinition();
    static const string PARAM_CUSTOM_TEXT;
};

#endif // _LOGBLADELETDEFINITION_H_
```

# makefile.sample

```
# Change the BASEDIR to point where the BDK is installed
BASEDIR = ../..

# Select optimization level
# DEBUG
#OPTIMIZE = -g -O0
# Optimized
OPTIMIZE = -O3

# Make common variables for building
#   $(BASEDIR)/include/stlport directory may not exist in some BDKs
LDFLAGS += -L. -L$(BASEDIR)/lib
CFLAGS  += -pthread -fpic $(OPTIMIZE) -D_XOPEN_SOURCE=500
CPPFLAGS+= -I. -I$(BASEDIR)/include/stlport -I$(BASEDIR)/include $(DIAGS)
CXXFLAGS+= -pthread -fpic $(OPTIMIZE) -D_XOPEN_SOURCE=500

# Local rules
%.o: %.c
    @rm -f $@
    $(strip $(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ $<)

%.o: %.cpp
    @rm -f $@
    $(strip $(CXX) $(CXXFLAGS) $(CPPFLAGS) -c -o $@ $<)
```

```
# List shared library object files
SOOBJS = \
    LogBladeletContext.o \
    LogBladeletDefinition.o \
    LogBladelet.o

# Shared object name (local name only, no directory)
SONAME = \
    libcustomlog.so

# Extra libraries for SONAME and EXECUTABLES builds.
EXTRALIBS = \
    $(BASEDIR)/lib/libaonfpbdk.so

all: $(SONAME)

clean:
    $(strip rm -f $(SOOBJS) $(SONAME) *.o *.d)
    rm -f $(SONAME).$(SOVERSION)

$(SONAME): $(SOOBJS) $(EXTRALIBS) $(DEPENDS)
    @rm -f $@
    $(strip $(CXX) $(CFLAGS) $(LDFLAGS) -shared -o $@ $(SOOBJS) \
        $(filter -l%,$(patsubst lib%.so,-l%,$(patsubst lib%.a,-l%,$(patsubst
$(BASEDIR)/lib/lib%,lib%,$^ $(LDLIBS))))))
```

CHAPTER 5

# External Services

AON External Services provide a set of APIs that enable bladelet and adapter customizations. Customers and partners can easily extend existing AON service functions by creating their own custom bladelets that can be incorpated in new and/or existing Policy Execution Plans (PEPs).

**Note** Although the External Services package is part of AONSCommon, it is described separately in this chapter. For descriptions of other AONSCommon components, see Appendix A, "AONSCommon Specification".

Generally, External Services APIs:

- Can be implemented in software, hardware, and both
- Use underlying internal service implementations
- Will not store across multiple invocations (stateless)
- Use valuable existing resources efficiently
- Provide an extensible interface permitting the addition of new functions and removal of obsolete functions

This chapter describes the External Services API, focusing on:

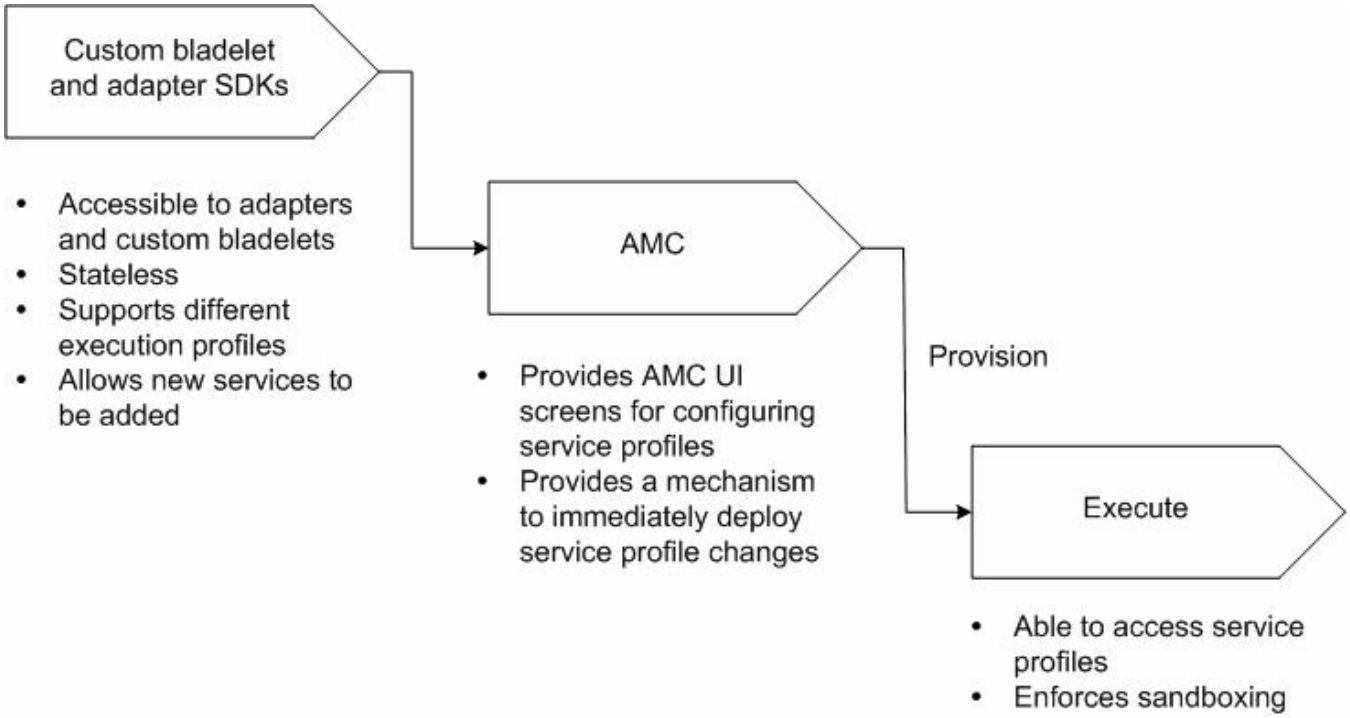


For background information, see Chapter 2, "Custom Bladelets" and Chapter 3, "Custom Adapters" in this guide, the *AON Installation and Administration Guide*, and *AON Development Studio Guide*.

# API Lifecycle

As Figure 4-1 shows, External Services meet demanding requirements for the Custom Bladelet and Adapter SDKs, the AON Management Console (AMC), and provisioning to AON nodes.

*Figure 5-1 External Services API Life Cycle*



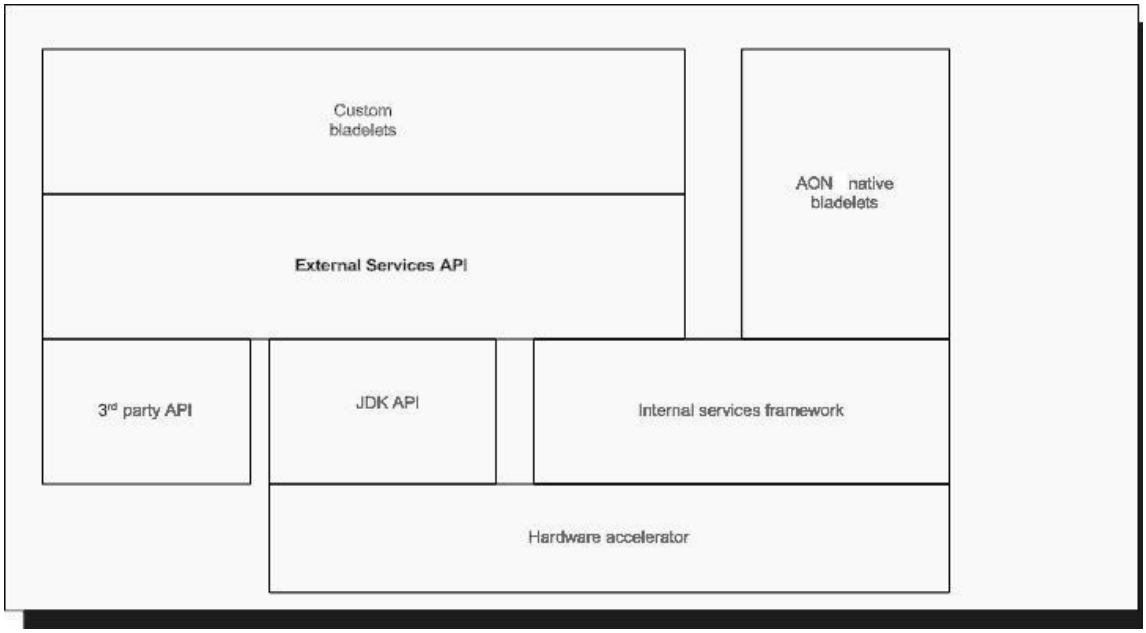

A service profile is a set of attributes that describe the service. Each service has one profile. A profile contains multiple named contexts. You use AMC windows to create named contexts.

The Service API is used to access a context. A supplied interface is used to obtain the attributes of a given context from the profile. A profile is associated with an attribute domain; a context is associated with a property set.

# External Services Architecture

The External Services API incorporates a pluggable architecture. The relationship with AON and other components is shown in Figure 4-2.

*Figure 5-2 External Services Architecture*




# Developing Content Parser Extensions

You can develop a new content parser plugin extension. These procedures are summarized in the following sections:

- Creating a Transformer Extension, page 5-3
- Packaging the Transformer Extension, page 5-4
- Creating a Content Parser Extension, page 5-7
- Packaging the Content Parser, page 5-8

## Creating a Transformer Extension

You may want to create a transformer extension. Follow the steps listed below.

**Step 1** Define a class that implements AONSTransformerFactory.

**Step 2** Package the Transformer including all necessary files.

Use AON Development Studio (ADS) windows for this step. See Packaging the Transformer Extension, page 5-4.

The package must contain a transform information file (transform-info.xml) and one or more XSL T files. For more information, see Transform Extension Information File, page 6-11.

**Step 3** Upload and Register the Transformer Package.

Use AON Management Console (AMC) windows for this step.

**Step 4** Define a Content Parser policy that specifies the class of AONSTransformerFactory defined in step 1.

Use the AMC windows to perform this task.

**Step 5** Define a Policy Execution Plans that uses the Transform bladelet and the Content Parser policy defined above.

---

For more information, see *AON Installation and Administration Guid*e, and the *AON Development Studio Guide*.

# Packaging the Transformer Extension

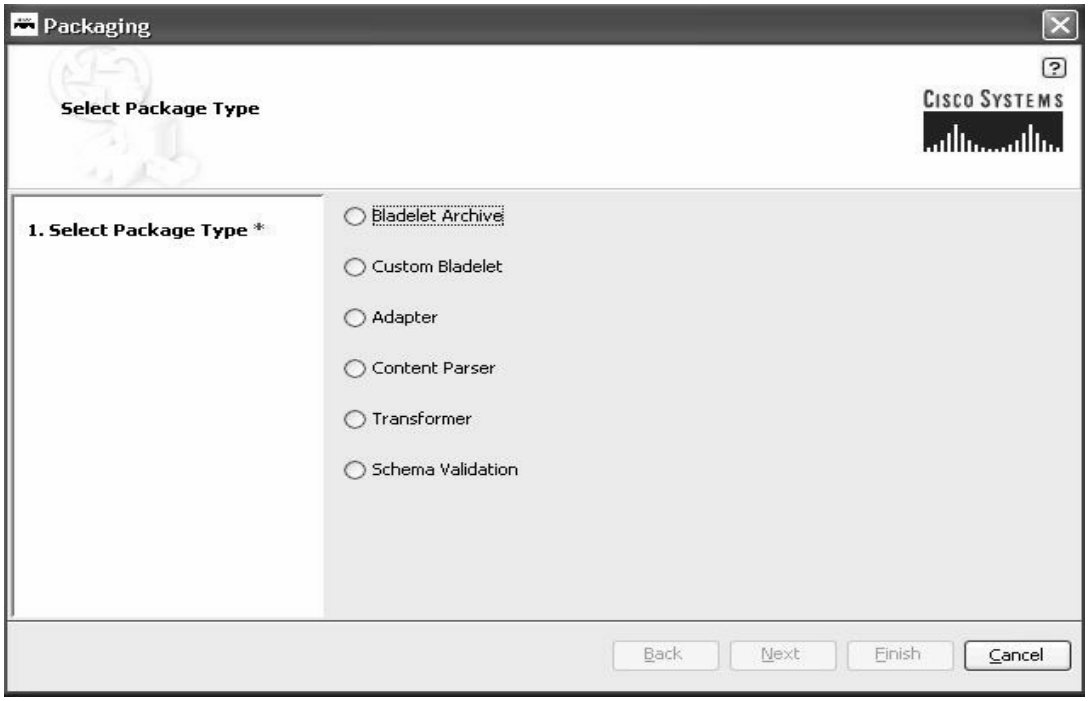
You use ADS windows to package the transformer extension. Later, you use AMC windows to upload the transform to AON. To package the extension, follow the steps listed below.

---

**Step 1** Using the ADS, select **Tools > Packaging**.

The AON Packaging window appears.

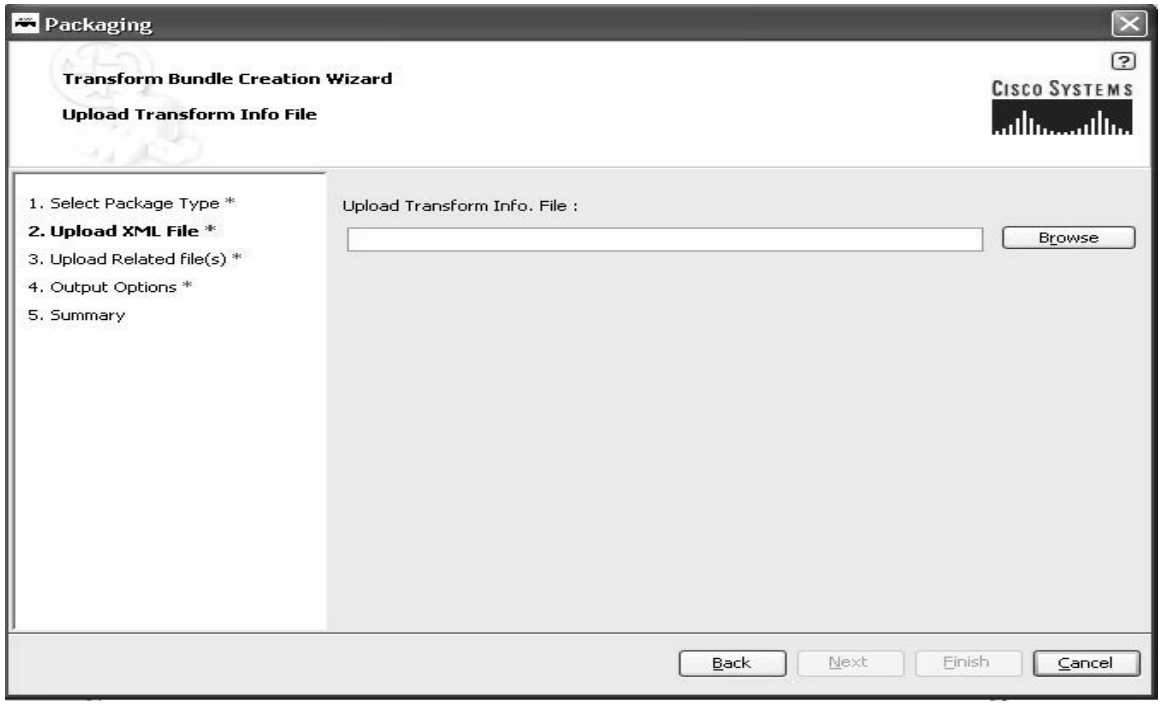
***Figure 5-3 AON Packaging window***




**Step 2** Select **Transformer** and click **Next**.

The Upload Transform Info File screen appears.

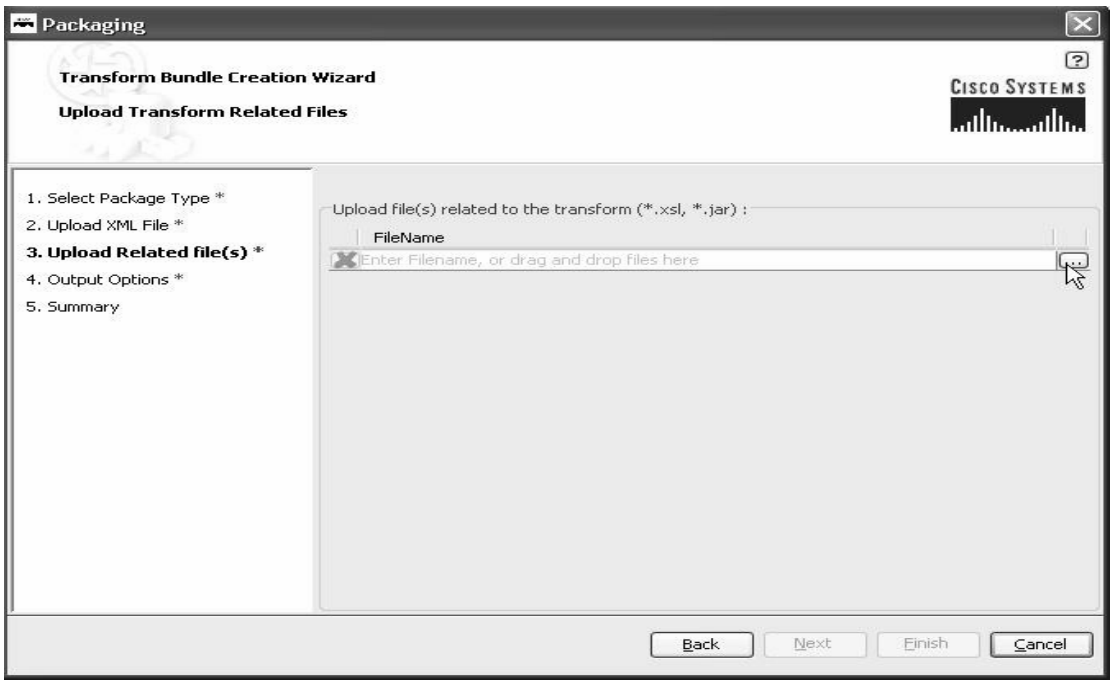
*Figure 5-4 Upload Transform Info File window*




**Step 3** Using the **Browser**, locate the transform information file (.xml) and click **Next**.

The Transform Bundle Creation Wizard - Upload Transform Related Files screen appears.

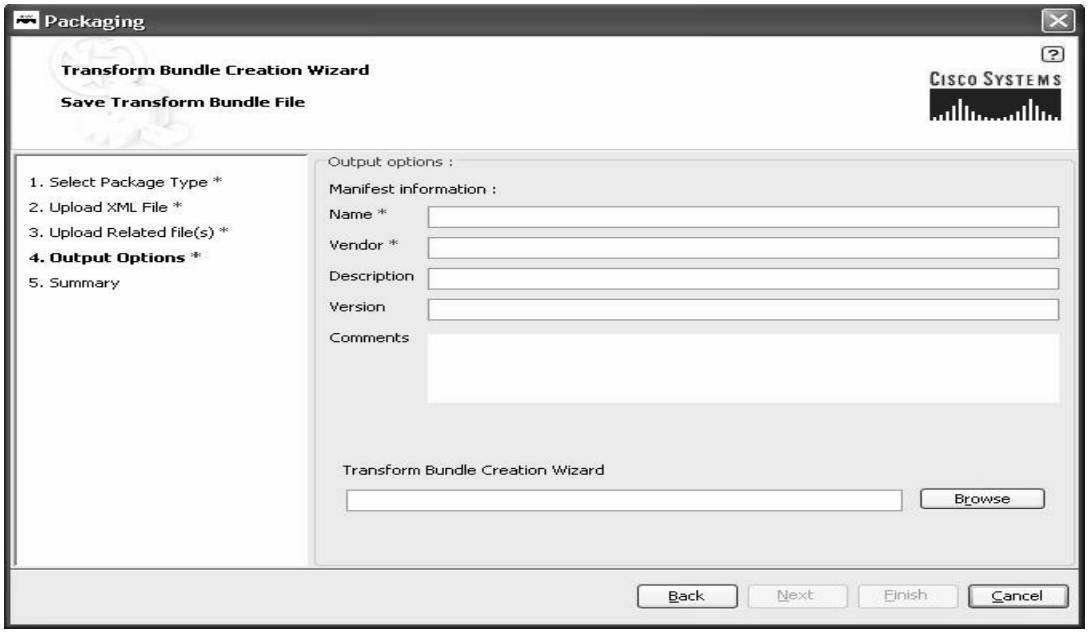
*Figure 5-5 Upload Transform Related Files window*




**Step 4** Using the selection button, locate the transform (.xsl or .jar) file and click **Next**.

The Transform Bundle Creation Wizard - Save Transformer Bundle File window appears.

*Figure 5-6 Save Transform Bundle File window*



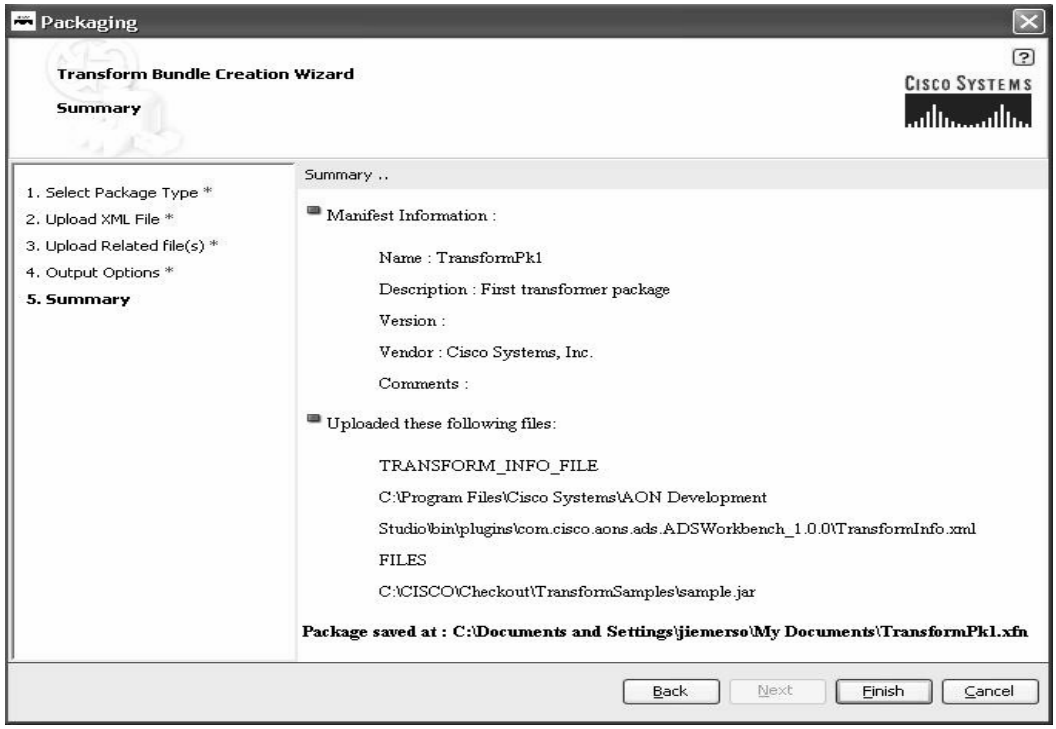

**Step 5** Fill in the manifest fields and click **Next**.

*Table 5-1 Manifest Information Fields*

| Field | Description |
|---|---|
| Name | Manifest name. |
| Vendor | Vendor. For example, Cisco Systems, Inc. |
| Description | (Optional) Description of the manifest contents. |
| Comments | (Optional) Comments about the manifest. |
| Transform Bundle Creation Wizard | Automatically filled with the Name field name and current default location. You may use the Browse button to find a different location. |

A Packaging Summary window appears. For example:

*Figure 5-7 Summary window*




**Step 6** Review the summary and click **Finish**.

A pop-up displays a **Package Created** message.

**Step 7** Click **OK** to return to the ADS main window.

To learn how packages are uploaded to the AON Management Console (AMC) and incorporated into AON, see the *AON Administration and Installation Guide*.

# Creating a Content Parser Extension

You may have to create a content parser extension that can be used to parse input data and convert it to an equivalent XML format on which AON XSLT Based Transformation can be applied. The steps are summarized below.

**Step 1** Define a class that implements `XMLReader`.

**Step 2** Create a content parser package.

For detailed steps, see Packaging the Content Parser, page 5-8.

**Step 3** Upload and register the content parser package.

Use AMC windows for this task.

**Step 4** Define a content parser policy that specifies the class name of parser plug-in class defined in step 1.

Use AMC windows for this task.

**Step 5** Define a Policy Execution Plans that uses Transform bladelet and uses the content parser policy defined above.

Use the ADS PEP Developer for this task.

---

For more information, see "Creating a Transformer Extension" section on page 5-3 and the *AON Administration and Installation Guid*e and the *AON Development Studio Guide*.

# Packaging the Content Parser

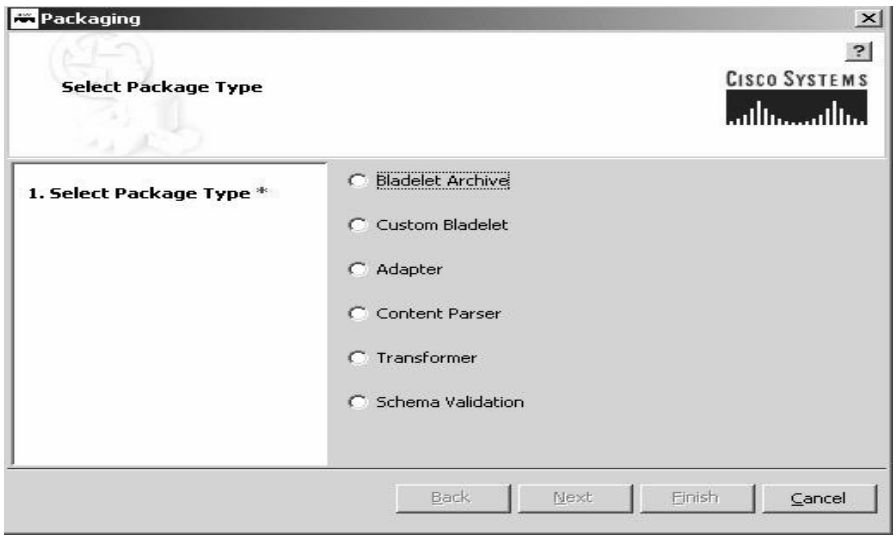
You use ADS windows to package the new content parser. Later, you use AMC windows to upload the content parser to AON. To package the transform, follow the steps listed below.

---

**Step 1** Using the ADS, select **Tools > Packaging**.

The AON Packaging window appears.

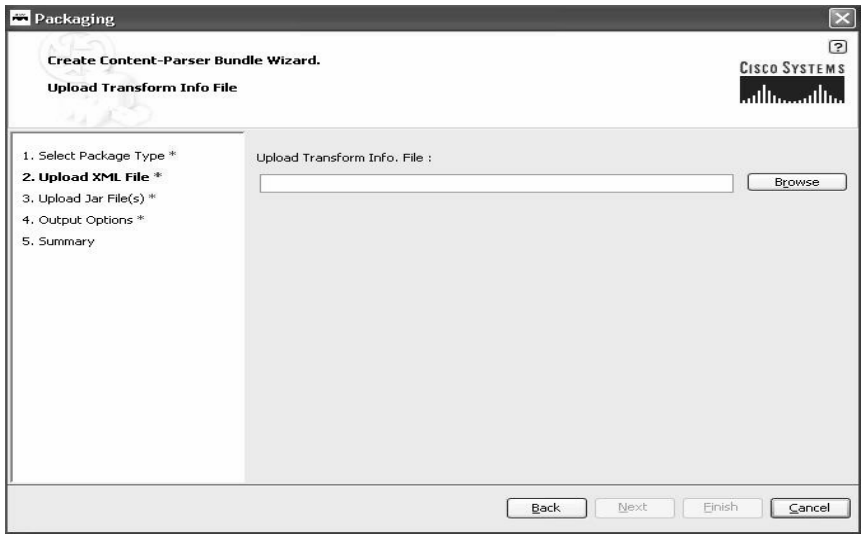
***Figure 5-8 AON Packaging Window***




**Step 2** Select **Content Parser a**nd click **Next**.

The Create Content Parser Bundle Wizard - Upload Transform Info File screen appears. For example:

*Figure 5-9 Upload Transform Info File window*

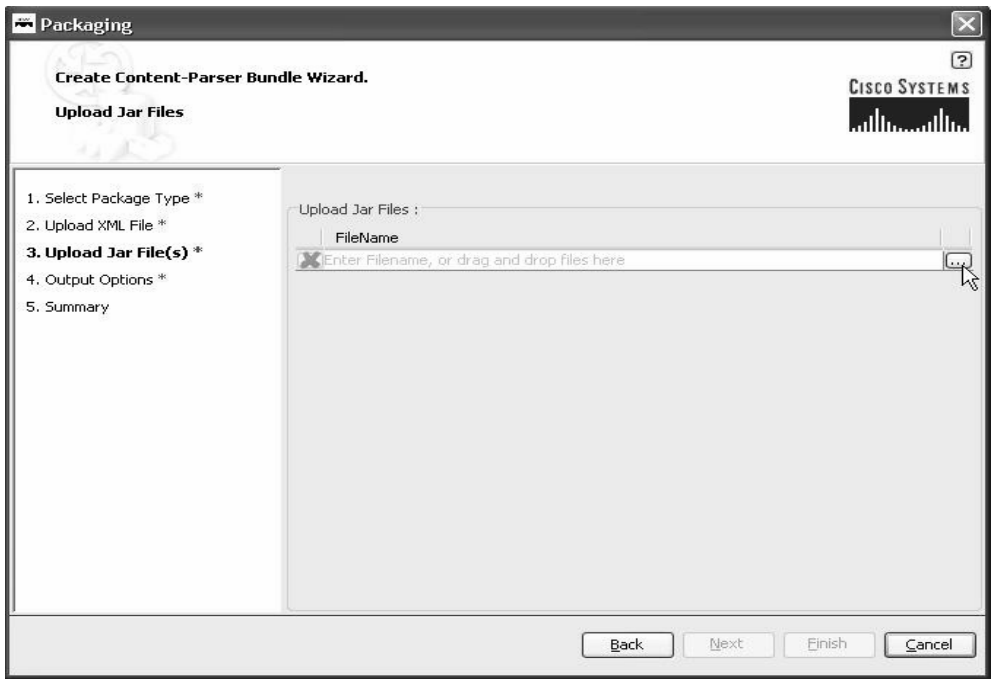



**Step 3** Click **Browse,** select a transform bundle (.xml file), then click **Next**.

The Upload Jar Files window appears.

**Note** You can create sample transform bundle (cdatalog.xml), using the CD Collection Content Parser XML File Sample, page 5-28.

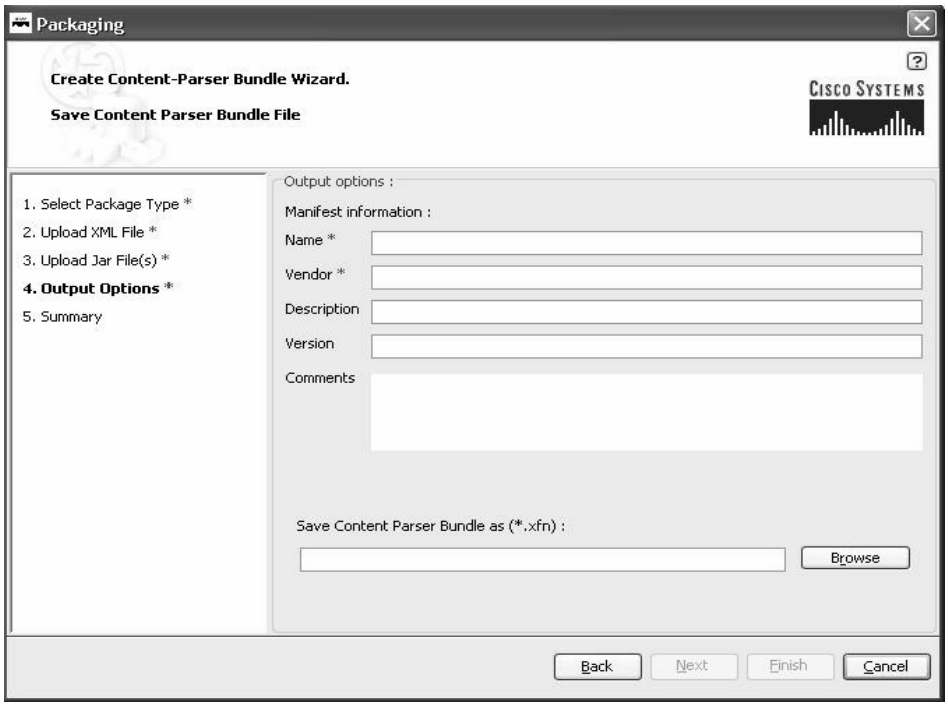
*Figure 5-10 Upload Jar Files window*




**Step 4** Using the selection button, locate the content parser Java archive file (.jar) and click **Next**.

The Save Content Parser Bundle File - Output Options screen appears.

*Figure 5-11 Save Content Parser Bundle File window*



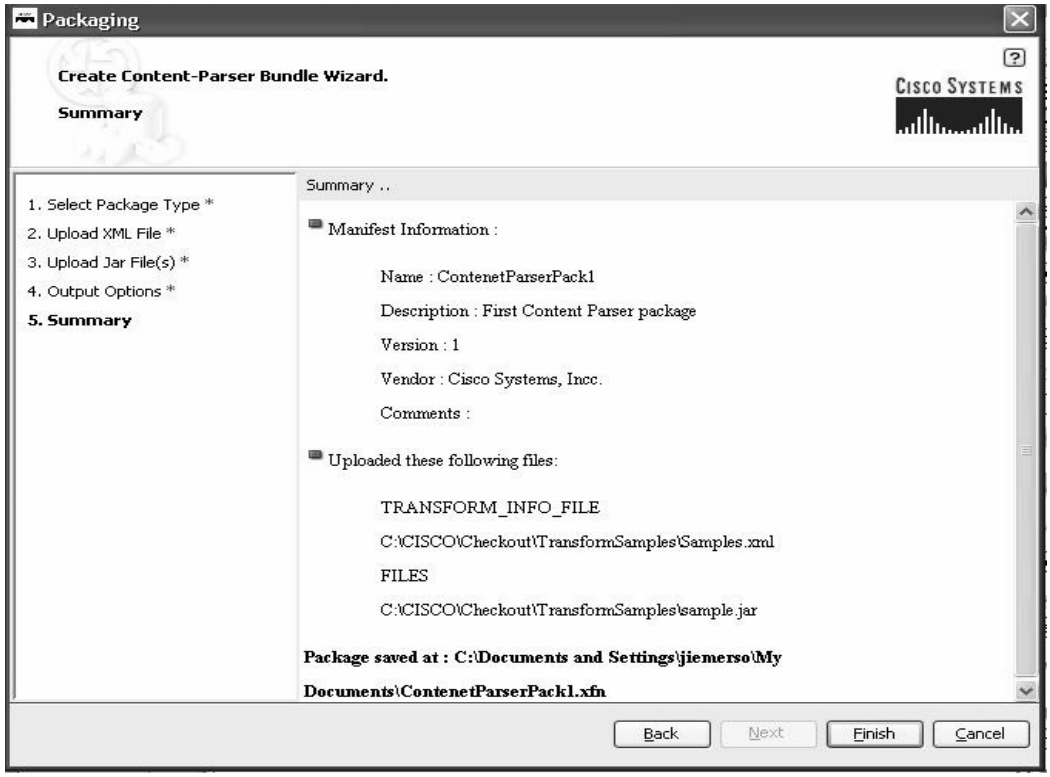

**Step 5** Fill in the manifest information and click **Next**.

*Table 5-2 Manifest Information Fields*

| Field | Description |
|---|---|
| Name | Manifest name. |
| Vendor | Vendor. For example, Cisco Systems, Inc. |
| Description | (Optional) Description of the manifest contents. |
| Comments | (Optional) Comments about the manifest. |
| Save Content Parser Bundle as | Automatically filled in the Name field name and current default location. You may use the Browse button to find a different location. |

A Packaging Summary window appears. For example:

*Figure 5-12 Content-Parser summary window*




**Step 6** Review the summary for completeness and click **Finish**.

After the package is created, you use the AMC to upload it to AON. For additional information, see the *AON Administration and Installation Guide*.

# External Services API Specification

The AON External Services package (com.cisco.aons.aonscommon.com.cisco.aons.extservice) incorporates the Java-coded interfaces listed below.

- ExtServiceProfile, page 5-21
- MessageLog, page 5-22
- MIME, page 5-23
- ServiceFactory, page 5-25
- Signature, page 5-26
- Transform, page 5-26

**Note** Although the External Services package is part of AONSCommon Specification, it is described separately in this chapter for convenience.

These essential components are described in the next sections.

# AONSTransformer

AONSTransformer defines a transformer object in AON. The interface takes XML data as input (SAXSource or DOMSource) and puts the transformed result in the target result object.

AONSTRansformer is implemented in the AON transformation service that provides XSLT Based Transformation in AON. An AONSTransformer object is created by a com.cisco.aons.service.transform.AONSTranformerFactory. In this process, the caller gets an instance of a specific Transformer using specific AONSTransformerFactory object. The class method is summarized in the following table.

| Method | Description |
|---|---|
| transform | public void **transform** (java.lang.String pluginProfileName,<br>java.lang.String stylesheetProfileName,<br>javax.xml.transform.Source src,<br>javax.xml.transform.Result target,<br>javax.xml.transform.Templates xsltTemplate,<br>org.xml.sax.XMLReader xmlReader)<br>throws ExtServiceException<br>**Parameters:**<br>pluginProfileName—Specifies the parser plugin profile to be used.<br>stylesheetProfileName—Specifies the transform policy profile to be used.<br>src—Input data.<br>target—Contains transformation results.<br>xsltTemplate—Template object corresponding to the style sheet specified in the stylesheetProfileName property set.<br>xmlReader —XML Reader obtained by instantiating the parser class specified in the pluginProfileName property.<br><br>Performs transformation on XML data. The data source may be a ISAXSource or DOMSource. The transformation result is put into the target Result object. |

For additional information, see Developing Content Parser Extensions, page 5-3.

# AONSTransformerFactory

AONSTransformerFactory defines a factory for AONSTransformer object. Each type of transformation implements this interface to provide a specific transformer object factory.

**Note** The Transformation bladelet is included in message PEPs to provide transformation services. See the "AON Bladelets" section in the *AON Development Studio Guide*.

The AONSTransformerFactory method is summarized in the following table.

| Method | Description |
|---|---|
| createTransformer | public AONSTSransformer **createTransformer**()<br>throws ExtServiceException<br>**Returns:**<br>Transformer.<br>Creates a transformer object. |

For additional information, see Developing Content Parser Extensions, page 5-3 and Creating a Content Parser Extension, page 5-7.

# Authentication

Extending ExtService, Authentication authenticates the given user credentials. The class method is summarized in the following table.

| Method | Description |
|---|---|
| authenticateAgainstLDAP | public boolean **authenticateAgainstLDAP**(java.lang.String userid,<br>java.lang.String password,<br>java.lang.String LDAPProfile)<br>throws<br>com.cisco.aons.exception.ExtServiceException<br>LDAPProfile—The LDAP entry that is defined using AMC. |

# CacheService

Extending ExtService, CacheService puts the object into a cache, gets the object from the cache, and removes the object from the cache.

While the cache is shared across PEPs on a blade, it is not distributed. Instead, it is local to each blade.

The object cache uses the LRU replacement algorithm. The associated key should be unique across instances of PEPs.

The class methods are summarized in the following table.

| Method | Description |
|---|---|
| put | public boolean **put** (java.lang.Object key,<br>java.lang.Object value,<br>java.lang.Long timeoutInSeconds)<br>throws ExtServiceException<br>**Parameters:**<br>key—The key must be a String type (unless a string is being computed). It must be unique across PEPs and messages. Users may decide to use the same key to identify a particular object.<br>value—Objects are stored in a local variable cache in memory. If a distributed cache is exposed, the object must be serializable.<br>timeoutInSeconds—Timeout value<br>**Returns:**<br>True if the operation is successful, otherwise false<br>Stores the object in the cache. |
| get | public java.lang.Object **get** (java.lang.Object key)<br>throws ExtServiceException<br>**Parameters:**<br>key—Serializable object used in the put operation It is unique across PEPs, instances of PEPs, and messages.<br>**Returns:**<br>Cached object<br>Retrieves the object from the cache |
| remove | public boolean **remove** (java.lang.Object key)<br>throws ExtServiceException<br>**Parameters:**<br>key—Serializable object used in the put operation It is unique across PEPs, instances of PEPs, and messages.<br>**Returns:**<br>True if the object is removed successfully from cache, otherwise false<br>Removes the object from the cache. |

# Compression

Extending ExtService, Compression compresses and decompresses input data. The class methods are summarized in the following table.

| Method | Description |
|---|---|
| compress | public byte[] **compress**(com.cisco.aons.message.IContent inputData)<br>throws ExtServiceException<br>**Parameters:**<br>inputData —Content data to be compressed<br>**Returns:**<br>byte[] compressed output data as byte array<br>Compresses input data and generates the output data. |
| decompress | public com.cisco.aons.message.IContent **decompress** (byte[] inputData,<br>int contentType)<br>throws ExtServiceException<br>**Parameters:**<br>inputData —Byte array data to be decompressed<br>contentType—Should be one of the content types defined ex: XML_CONTENT_TYPE<br>**Returns:**<br>Content decompressed output content<br>Decompresses input data and creates content |

# ContentLookup

Extending ExtService, ContentLookup evaluates expressions on the given content and produces a collection of objects. This interface is used to lookup regular expressions. The class methods are summarized in the following table.

| Method | Description |
|---|---|
| lookup | public java.util.HashMap **lookup**(com.cisco.aons.message.IContent input,<br>java.util.HashMap exps)<br>throws ExtServiceException<br>**Parameters:**<br>input—Content to be evaluated<br>exps—Hashmap contains key and value. For regular expressions, the resulting value is an array of expressions. For XPath expressions, the output is an expression.<br>**Returns:**<br>Output Hashmap contains key and value. For regular expressions, the resulting value contains the list of strings. For XPath expressions, the output is a string.<br>Looks up regular expression content suing the current context. If there is no current context, the default context is used. If there is no default context, an exception is thrown. |
| lookup | public java.util.HashMap **lookup**(com.cisco.aons.message.IContent input,<br>java.util.HashMap exps,<br>ExtService Context context)<br>throws ExtServiceException<br>Regular expression lookup content.<br>**Parameters:**<br>input—Content to be evaluated<br>exps— Hashmap contains key and value. For regular expressions, the resulting value is an expression. For XPath expressions, the output is an expression.<br>context—Input configurable parameters such as lookup type.<br>**Returns:**<br>Output Hashmap contains key and value. For regular expressions, the resulting value contains a list strings. For XPath expressions, the output is a string. |

# ContentValidation

The ContentValidation (also known as "Validate") interface validates the input document with the given XML schema or DTD (document type definition). Schema/DTD meta data is already stored in the AON network node profile as domain (property) sets. AON parses XML grammar, already stored in cache. Policies are stored in attribute domain files as com.cisco.aons.policies.validation.ContentValidation. The class methods are summarized below.

| Method | Description |
|---|---|
| validate | public boolean **validate**(IContent content)<br>throws ExtServiceException<br>**Parameters:**<br>content—Input content to be validated<br>**Returns:**<br>True if successfully validates the input stream, otherwise false<br>Validates the input content with the schema/DTD. Uses the current context. If there is no current context, the default context will be used. If there is no default context, an exception will be thrown. |
| validate | public boolean **validate**(IContent content,<br>ExtServiceContext context)<br>throws ExtServiceException<br>**Parameters:**<br>content—Input content to be validated.<br>context—ExtService Context to pass information such as type of validation (xsd,schema).<br>**Returns:**<br>True if successfully validates the input stream, otherwise false<br>Validates the input content with the schema-DTD. |

For more information, see Chapter 7, "Schema Validation"

# Encryption

Extending ExtService, Encryption encrypts the input and decrypts previously encrypted documents. The class methods are summarized in the following table.

| Method | Description |
|---|---|
| encrypt | public boolean **encrypt** (com.cisco.aons.message.IContent content,<br>java.lang.String dataToEncrypt)<br>throws ExtServiceException<br>**Parameters:**<br>content—Input/output content<br>dataToEncrypt—Data to encrypt<br>**Returns:**<br>True if encryption is successful otherwise false<br>Encrypts the given input content and produces the output content Uses the current context. If there is no current context, the default context will be used. If there is no default context exception will be thrown. |
| encrypt | public boolean **encrypt** (com.cisco.aons.message.IContent content,<br>java.lang.String dataToEncrypt,<br>ExtServiceContext context)<br>throws ExtServiceException<br>**Parameters:**<br>content—Input/output content<br>dataToEncrypt—Data to encrypt<br>context—Service Context parameters<br>**Returns:**<br>True if encryption is successful otherwise false<br>Encrypts the given input content and produces the output content |

| Method | Description |
|---|---|
| decrypt | public int **decrypt**(com.cisco.aons.message.IContent content,<br>java.lang.String elementsToDecrypt)<br>throws ExtServiceException<br>**Parameters:**<br>content—Input/output content<br>elementsToDecrypt—Elements to be decrypted<br>**Returns:**<br>0—If successful<br>-1—If decryption element not found<br>-2—If decryption key not found<br>Decrypts the encrypted content.xml content Uses the current context. If there is no current context, then default context will be used. If there is no default context exception will be thrown. |
| decrypt | public int **decrypt**(com.cisco.aons.message.IContent content,<br>java.lang.String elementsToDecrypt,<br>ExtServiceContext context)<br>throws ExtServiceException<br>**Parameters:**<br>content—input/output content<br>elementsToDecrypt—Elements to be decrypted<br>**Returns:**<br>0—If successful<br>-1—If decryption element not found<br>-2—If decryption key not found)<br>Decrypts the encrypted content.xml content |

# ExtService

Public interface ExtService is used to get the name and profile of a service. It is implemented by the interfaces listed below.

- Authentication
- CacheService
- Compression
- ContentLookup
- Encryption
- Encryption
- ExtService

- MessageLog
- MIME
- Transform
- ContentValidation

ExtService methods are summarized in the following table.

| Method | Description |
|---|---|
| getName | public java.lang.String **getName**()<br>**Returns:**<br>String name of the service |
| geProfile | public ExtServiceProfile **getProfile**()<br>**Returns:**<br>Service profile associated with the service if any. |

# ExtServiceContext

Public interface ExtServiceContext is used to get the context name and attribute information, and set attributes. The class methods are summarized in the following table.

| Method | Description |
|---|---|
| getAttribute | public java.util.List **getAttribute**(java.lang.String attrName)<br>**Parameters:**<br>attrName—Input attribute name defined in the context<br>**Returns:**<br>List of all attribute values for the attribute name. |
| getAttributes | public java.util.Map **getAttributes**()<br>**Returns:**<br>Map Key is attribute name, Value is List of values<br>Returns all information defined in the context. |
| setAttributes | public void **setAttributes**(java.util.Map attrMap) |
| getName | public java.lang.String **getName**()<br>**Returns:**<br>String name of the context |

# ExtServiceProfile

Public interface ExtServiceProfile is used to get the profile and service names, contexts defined in the profile, get the current context, and set the context. Class methods are summarized in the following table.

| Method | Description |
|---|---|
| getContext | public ExtServiceContext **getContext**(java.lang.String contextName)<br>**Parameters:**<br>contextName—Input context name<br>**Returns:**<br>Service context defined in the profile. |
| getContexts | public java.util.List **getContexts**()<br>**Returns:**<br>Contexts defined in the profile. |
| getCurrentContext | public ExtServiceContext **getCurrentContext**()<br>**Returns:**<br>Current context set in the service. The context must be defined already in profile. |
| setCurrentContext | public void **setCurrentContext** (java.lang.String contextName)<br>throws ExtServiceException<br>**Parameters**:<br>contextName - Name of the context.<br>Sets the current context. |
| setCurrentContext | public void **setCurrentContext** (java.lang.String contextName,<br>java.util.HashMap attrMap)<br>throws ExtServiceException<br>**Parameters**:<br>contextName— Name to be set for the current context.<br>attrMap—Contains:<br>name—attribute name<br>value—list of string values for the attribute.<br>Sets the current context. |
| getDefaultContext | public ExtServiceContext **getDefaultContext**()<br>**Returns:**<br>defaultContext ExtServiceContext type<br>Returns the default context from the profile (name as “default”). |

| Method | Description |
|---|---|
| getName | public java.lang.String **getName**()<br>**Returns:**<br>Name (string) of the profile. |
| getServiceName | public java.lang.String **getServiceName**()<br>**Returns:**<br>Name (string) of the service to which the profile belongs. |

# MessageLog

Extending ExtService, MessageLog provides database backed logging with predefined schema. In addition to attributes of the request/response message, users can select message contents by XPath. Users can also log Java objects that have appropriate String representation. In addition, message logging can be synchronous or asynchronous. Users can choose a destination from a pre-configured set of data sources configured on AMC. The fields are briefly summarized below.

| Fields | Description |
|---|---|
| EX_FAILED | public static final java.lang.String **EX_FAILED** |
| EX_TIMEOUT | public static final java.lang.String **EX_TIMEOUT** |
| LOG_BASIC | public static final short **LOG_BASIC** |
| LOG_BODY | public static final short **LOG_BODY** |
| LOG_BY_EXPRESSION | public static final short **LOG_BY_EXPRESSION** |
| LOG_HEADER | public static final short **LOG_HEADER** |
| LOG_WHOLE_MESSAGE | public static final short **LOG_WHOLE_MESSAGE** |
| serviceName | public static final java.lang.String **serviceName** |

# MIME

Extending ExtService, MIME provides add, update, delete, extract operations on MIMEContent. the class methods are summarized in the following table.

| Method | Description |
|---|---|
| add | **add** (IMIMEContent input, IContent newContent)<br>**Parameters:**<br>input—MIME input content<br>newContent—Content to be added<br>**Returns:**<br>output MIME content<br>Adds new content |
| delete | **delete** (IMIMEContent input, IContent deleteContent)<br>**Parameters:**<br>input—MIME input content.<br>deleteContent—Content to be deleted.<br>**Throws:**<br>ExtServiceException<br>**Returns:**<br>output MIME content<br>Deletes the (part) content in the given MIME content |
| delete | **delete** (MIMEContent input, int index)<br>**Parameters:**<br>input—MIME input content.<br>index—At which content to be deleted.<br>**Throws:**<br>ExtServiceException<br>**Returns:**<br>output MIME content<br>Deletes the (part) content in the given MIME content |

| Method | Description |
|---|---|
| extract | extract (IMIMEContent content, java.lang.String selector)<br>**Parameters:**<br>content—Input MIME input content.<br>selector—format is field==value<br>**Throws:**<br>ExtServiceException<br>**Returns:**<br>list of the parts for the given selector<br>Extracts the given fields from the MIME Content |
| insert | **insert** (IMIMEContent input, IContent newContent, int index)<br>**Parameters:**<br>input—MIME input content<br>newContent—Content to be inserted<br>index—Index in which it has to be inserted<br>**Throws:**<br>ExtServiceException<br>**Returns:**<br>output MIME content<br>Inserts new context at the index. |

| Method | Description |
|---|---|
| replace | **replace** (IMIMEContent input, IContent newContent)<br>**Parameters:**<br>input—MIME input content.<br>newContent—Content to be inserted.<br>**Throws:**<br>ExtServiceException<br>**Returns:**<br>output MIME content<br>Replaces the input content with the new one |
| replace | **replace** (.IMIMEContent input, IContent newContent, int index)<br>**Parameters:**<br>input—MIME input content.<br>newContent—Content to be inserted.<br>index—At which the content has to be replaced.<br>**Throws:**<br>ExtServiceException<br>**Returns:**<br>output MIME content<br>Replaces the input content with the new one. |

# ServiceFactory

Public interface ServiceFactory is used to get the current service. The single class method is summarized in the following table.

| Method | Description |
|---|---|
| getService | public ExtService **getService**(java.lang.String serviceName)<br>throws ExtServiceException<br>**Returns:**<br>String name of the service<br>Returns the service. |

# Signature

The Signature (also known as "DigitalSignature") interface signs the input XML document. The class methods are summarized in the following table.

| Method | Description |
|---|---|
| sign | boolean sign (IContent content, String nodeToSign, ExtServiceContext context) throws ExtServiceException<br>**Parameters:**<br>content—Input to be signed<br>nodeToSign—Node to be signed<br>ExtServiceContext—Input context.<br>**Returns:**<br>True if successful. Otherwise, returns false.<br>Signs an input XML document |
| verify | boolean verify (IContent content, ExtServiceContext serviceContext)<br>throws ExtServiceException<br>**Parameters:**<br>content—Input to be verified<br>serviceContext—Input service context<br>**Returns:**<br>True if successful. Otherwise, returns false.<br>Verifies the signed XML document. |

# Transform

Extending ExtService Transform defines an input source document with the given profile. The interface is implemented by the default transformer in the AON transformation service that provides XSL Transformation (XSLT) based transformation. An AONSTranformer object is created by a com.cisco.aons.service.transform.AONSTransformerFactory. The caller gets an instance of a specific transformer using the particular transformer factory object.

To use this service, first setup the transform policy in AON. The following tables summarize the transform field and method.

| Field | Description |
|---|---|
| serviceName | public static final java.lang.String **serviceName** |

| Method | Description |
|---|---|
| transform | public void **transform** (java.lang.String pluginProfileName,<br>java.lang.String stylesheetProfileName,<br>javax.xml.transform.Source src,<br>javax.xml.transform.Result target)<br>throws ExtServiceException<br>This function performs the transformation. It applies the XSLT transform on a specified src document and produces a result document.<br>**Parameters:**<br>pluginProfileName— Specified profile to use for the parser plugin. This should be the property set key defined in file ContentParser.xml which has attribute domain as com.cisco.aons.policies.transform.ContentParser.<br>stylesheetProfileName—Specified stylesheet profile to be used. This should be the property set key defined in policy file Transformation.xml which has attribute domain as com.cisco.aons.policies.transform.Transformation.<br>src—Specifies the input source document.<br>target—Result of the transformation.<br>**Throws:**<br>ExtServiceException<br>Performs transformation. It takes an XML Data as input on which transformation is performed. The data source can be a SAXSource or DOMSource. The transformation output is put in the target result object. |

# ExtServiceException

Part of the AONSCommon Exception Package, the ExtServiceException class defines exception conditions to be captured by service clients. The constructors are summarized in the following table. For more information, see ExtServiceException, page A-7.

| Constructor | Description |
|---|---|
| ExtServiceException(java.lang.Exception e) | Creates an ExtServiceException by wrapping the exception. |
| ExtService Exception (java.lang.String message) | Creates an ExtServiceException in response to an input message. |
| ExtServiceException(java.lang.String message, java.lang.Exception e) | Creates an ExtServiceException by wrapping the exception |

Although ExServiceException is not in the External Services group, it is described in this chapter because it is thrown by many External Services class methods such as transform.

# Content Parser Sample

The following section provides a sample of a content parser bundle: CD Collection Content Parser XML File Sample, page 5-28.

## CD Collection Content Parser XML File Sample

The following sample CD Collection is a content parser xml file.

```
<?xml version="1.0" encoding="ISO-8859-1"?><xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Title</th>
      <th align="left">Artist</th>
      <th align="left">Country</th>
      <th align="left">Company</th>
    </tr>
    <xsl:for-each select="catalog/cd">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
      <td><xsl:value-of select="country"/></td>
      <td><xsl:value-of select="company"/></td>
    </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template></xsl:stylesheet>
```

CHAPTER 6

# XSLT Transformation

The XSL Transformation (XSLT) service determines how an input message is transformed to produce the output message. The service ca n takes an incoming message of type that you specify and generates a transformed outgoing message.

This chapter provides an overview of message transformation, describes the transform parsers,and how to use them, focusing on the following:



**Note** See External Services API Specification, page 5-11 for descriptions of the key interfaces AONSTransformer and AONSTransformerFactory.

# Prerequisites

Before you perform the activities summarized in this chapter, you must:

- Create a valid Extensible Stylesheet Language Transformation (XSLT) file with any XSL editor. A sample file is shown in Friends.xsl, page 6-15.
- Use ADS to create transform and content parser packages. A transform package requires one or more XSLT files and the Transform Extension Information File (transform-info.xml). See Creating a Content Parser Extension, page 5-7.
- Use the AMC to register the packages so that they can be deployed on AON nodes.
- Use AMC property management windows to create transformation and content parser properties. These can be created at a node or global level.
- Use ADS to create transform Policy Execution Plans (PEP) and associated message classification properties.

For details, see the *AON Administration and Installation Guide* and the *AON Development Studio Guide*.

# Message Transformation

The Transform bladelet is used to transform an incoming XML or non-XML message to a new XML or non-XML message. You include the bladelet in a PEP to perform the following transformations

- XML to XML—Use XSLT (or a Java-coded plugin) to transform the message.
- XML to non-XML—Use XSLT or a Java-coded plugin.
- Non-XML to XML—Use a Java-coded plugin.
- Non-XML to non-XML—Use a Java-coded plugin.

For more information, see Creating the Transform Parser, page 6-2, Packaging the Transform Parser, page 6-3, and Transformation Examples, page 6-8.

# Creating the Transform Parser

Follow the steps listed below to create and upload a transform parser. For more information, see Transformation Examples, page 6-8 and the description of Transform bladelet in the *AON Development Studio Guide*.

**Step 1** Create a transform bundle containing the XSLT or Java plugin.

**Step 2** Package the transform including all necessary files.

Using the ADS to package the files. See Packaging the Transform Parser, page 6-3.

**Step 3** Upload the package bundle to AMC.

Using AMC, select **Admin >Extensions > Transform Package > Upload**.

**Step 4** Create policy files for the transformation.

Using AMC, select **Properties > Application > Global > Transformation**.

**Step 5** Synchronize ADS with AMC.

Using ADS, click **Sync**.

This action makes the newly created policy file available for use.

**Step 6** Create a PEP.

Using ADS, create a new Policy Execution Plans including the Transform bladelet.

**Step 7** Upload the PEP to AMC.

Using ADS, click **Sync**.

**Step 8** Deploy the PEP to AON.

Using AMC, select **Deploy**.

**Step 9** Send the XML file to the end server.

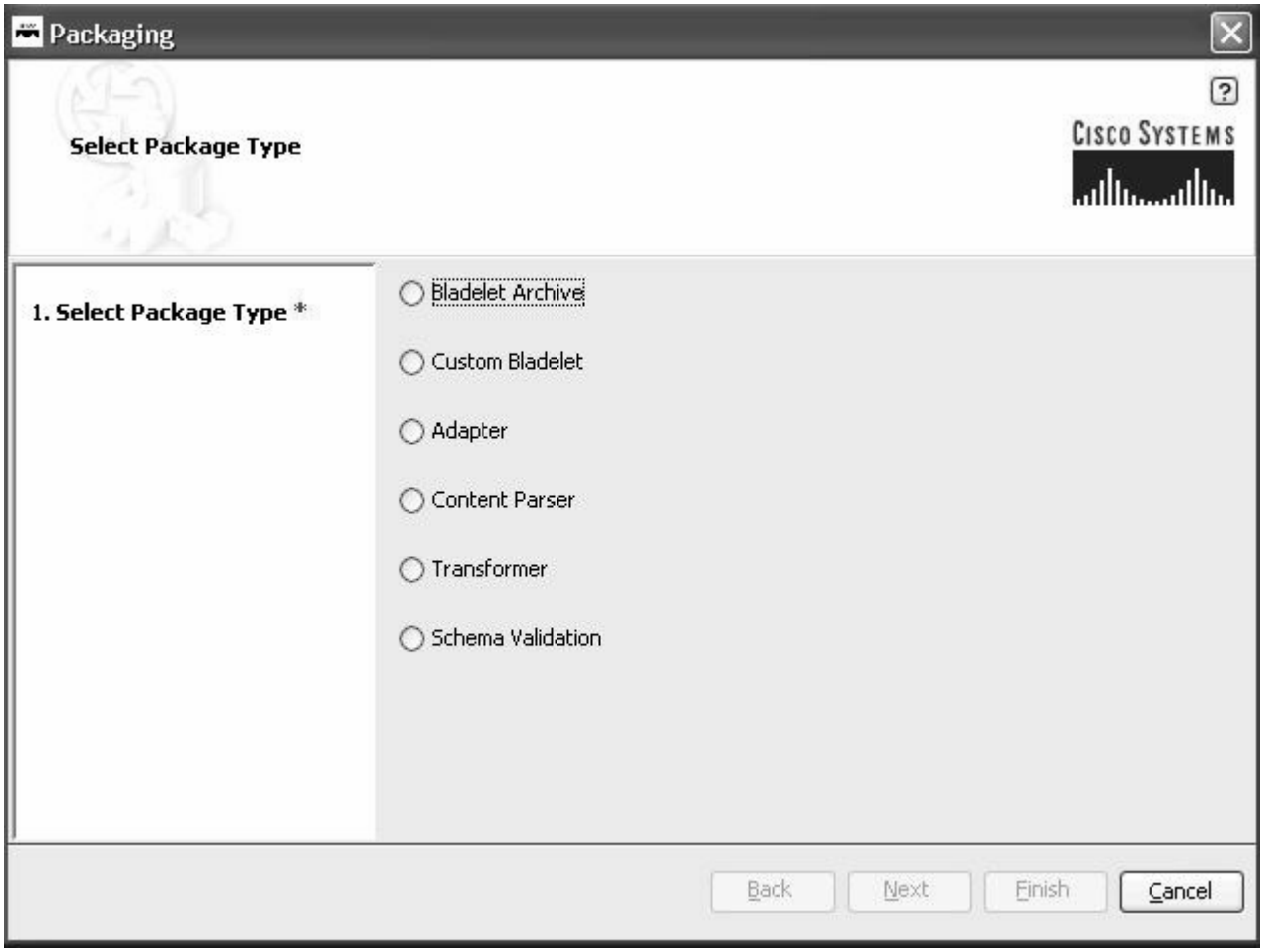
# Packaging the Transform Parser

You use ADS windows to package the XSL transform. Later, you use AMC windows to upload the transform to AON. To package the transform, follow the steps listed below.

**Step 1** Using the ADS, select **Tools > Packaging**.

The AON Packaging window appears.

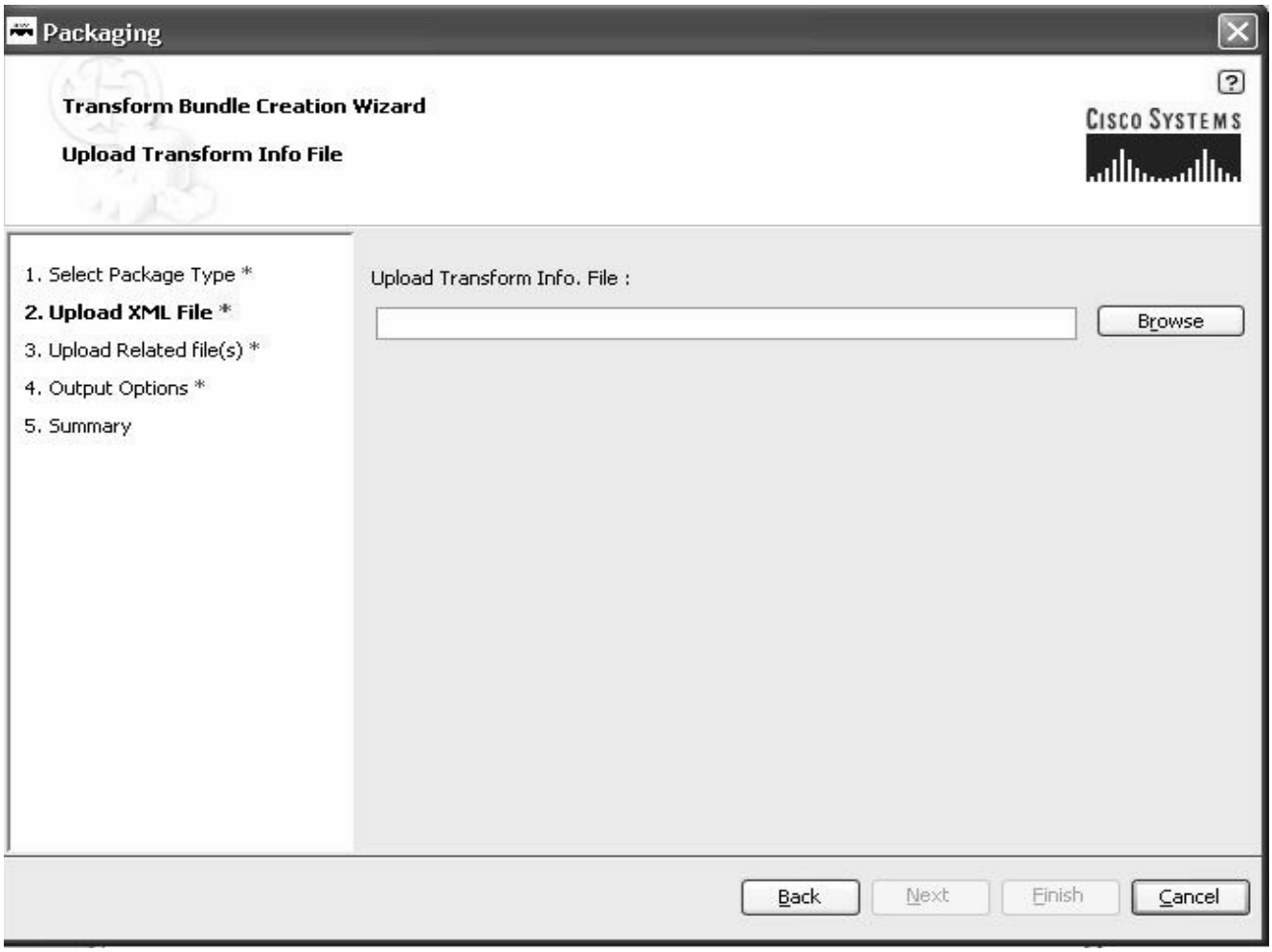
***Figure 6-1 AON Packaging Window***




**Step 2** Select **Transformer** and click **Next**.

The Upload Transform Info File screen appears.

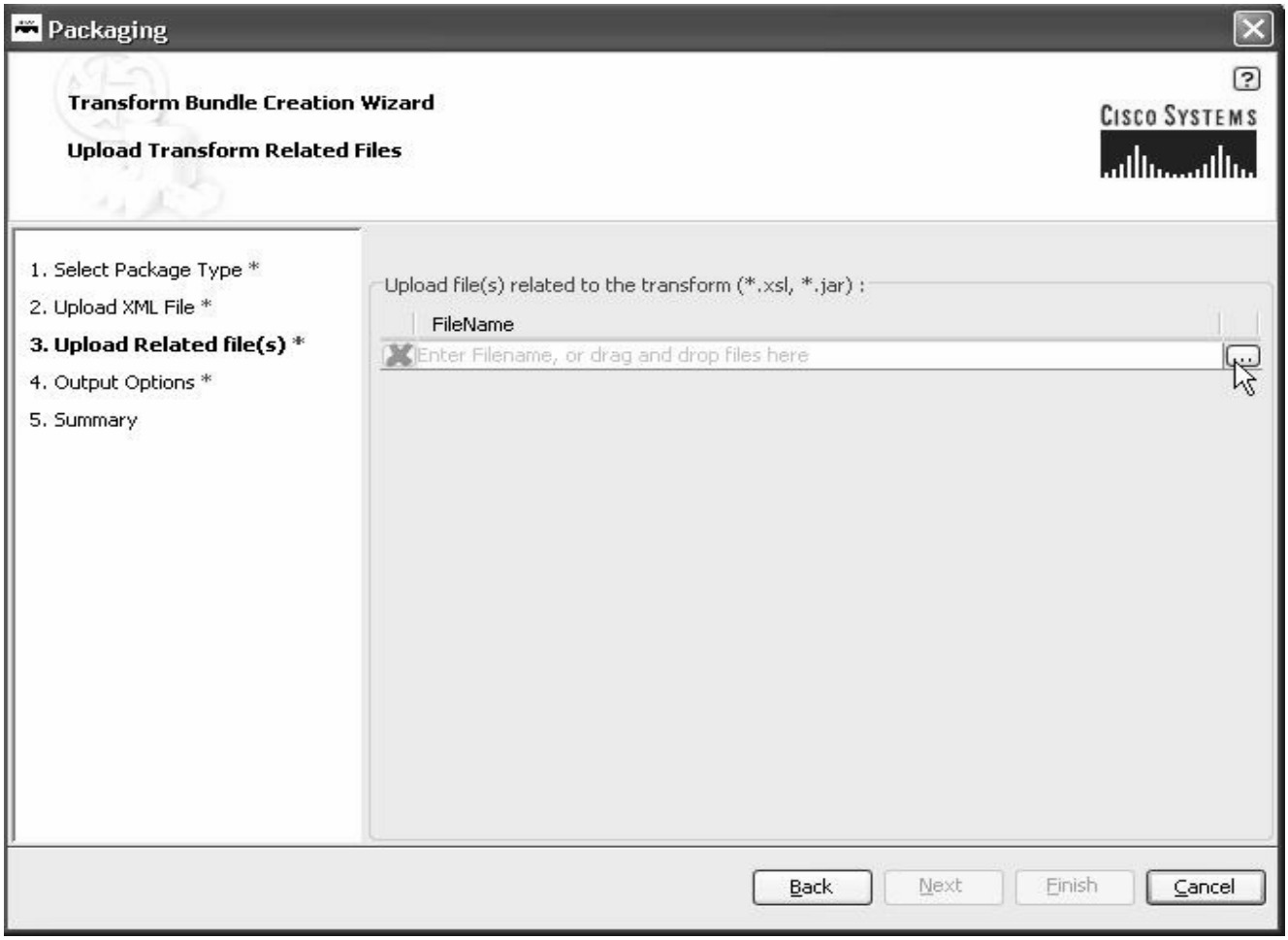
*Figure 6-2 Upload Transform Info File window*




**Step 3** Using the **Browser**, locate the transform information file (.xml) and click **Next**.

The Transform Bundle Creation Wizard - Upload Transform Related Files screen appears.

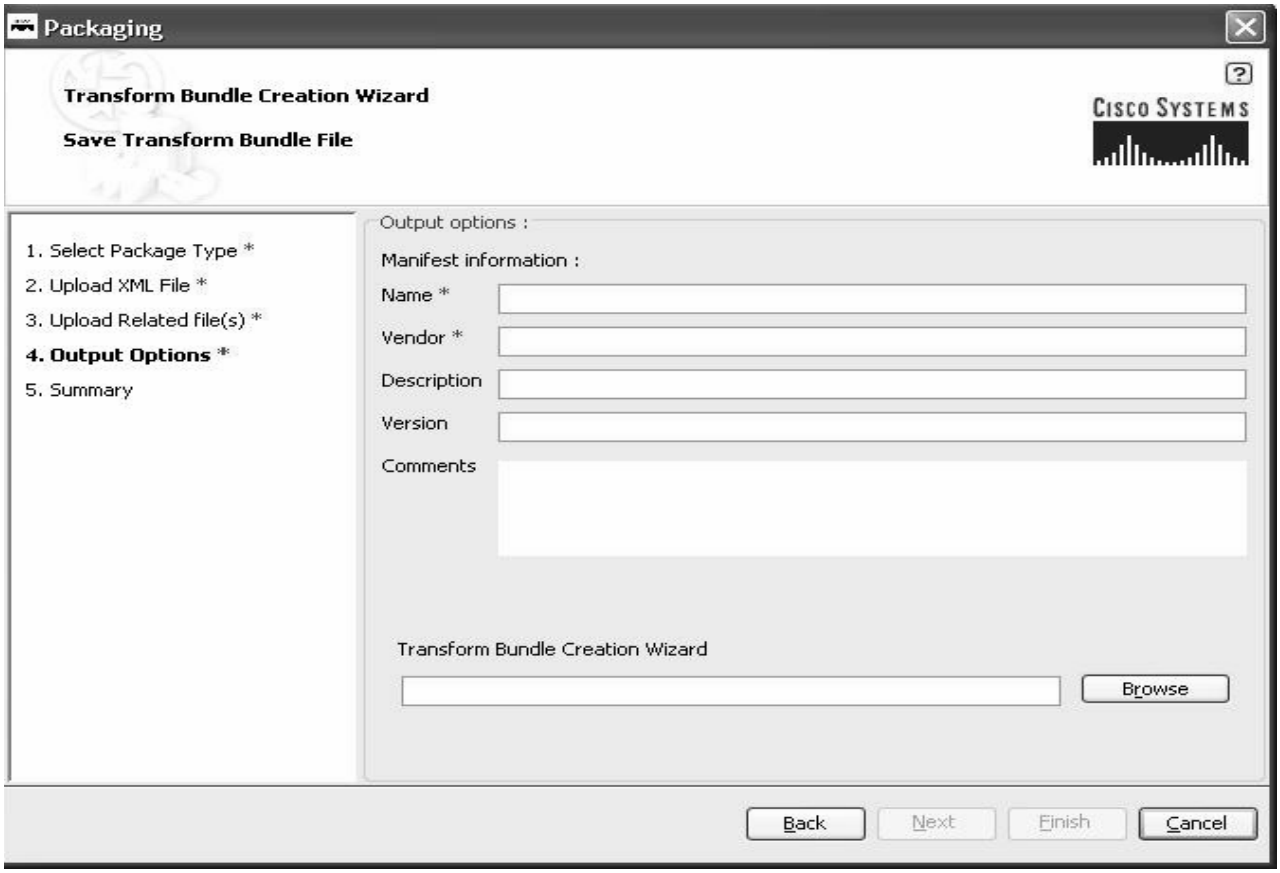
*Figure 6-3 Upload Transform Related Files window*




**Step 4** Using the selection button, locate the transform (.xsl or .jar) file and click **Next**.

The Transform Bundle Creation Wizard - Save Transformer Bundle File screen appears.

*Figure 6-4 Save Transform Bundle File window*



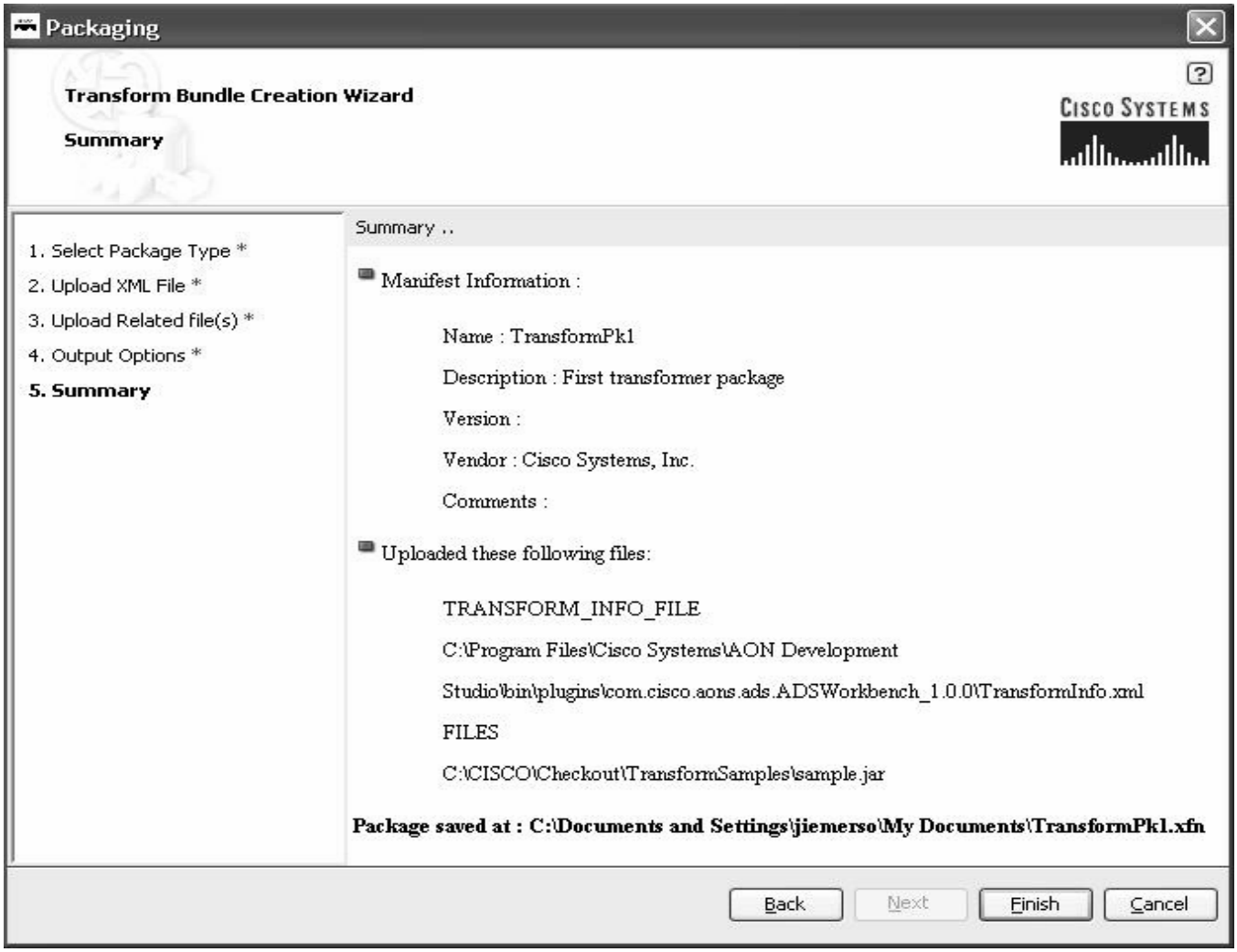

**Step 5** Fill in the manifest fields and click **Next**.

*Table 6-1 Manifest Information Fields*

| Field | Description |
|---|---|
| Name | Manifest name. |
| Vendor | Vendor. For example, Cisco Systems, Inc. |
| Description | (Optional) Description of the manifest contents. |
| Comments | (Optional) Comments about the manifest. |
| Transform Bundle Creation Wizard | Automatically filled with the Name field name and current default location. You may use the Browse button to find a different location. |

A Packaging Summary screen appears. For example:

*Figure 6-5 Summary*




**Step 6** Review the summary and click **Finish**.

A success message appears.

*Figure 6-6 Success Message*


**Step 7** Click **OK** and you are returned to the ADS main window.

To learn how packages are uploaded to the AON Management Console (AMC) and incorporated into AON, see the *AON Administration and Installation Guide*.

# Transformation Examples

The following examples show how to use the AON XSLT transformation or a Java plugin to transform a document from one format to another.


- XSLT Transformation Example, page 6-8
- Java Plugin Transformation Example, page 6-9


For more information, see the *AON Development Studio Guide* description of the Transform bladelet.

## XSLT Transformation Example

In this example, HTTP client SOAPTest sends friends.xml to AON over HTTP. AON reads the incoming XML message, associates it with the message type (and therefore, with the PEP), and transforms it to HTML

Example Files:


- Input file = Friends.xml, page 6-15
- XSLT file = Friends.xsl, page 6-15

- Output file= friends.xml (HTML message)
- Transform bundle = transform.xfn (zipped file containing bundle component files)

**Note** The bundle file (.xfn) is not shown.

***Example 6-1***

**Step 1** Step 1) Create the Transform bundle and save it.

- Using ADS, create a Transform bundle including friends.xsl file in the bundle.

  Using ADS, select **Tools > Packaging > Transformer**. Follow the screen directions. See Packaging the Transform Parser, page 6-3 for details.
- Save this file on the local hard disk as transform.xfn.

**Step 2** Step; 2) Upload and register the bundle.

- Using AMC, select **Admin > Extensions > Transform packages**.
- Click **Upload** to upload the transform.xfn file.
- Click **Register**.

**Step 3** Step 3) Select a new transformation policy, select the XSLT file, and save the policy.

- Using AMC, select **Properties > Application > Global > Transformation**.
- Click **New**.
- Select a new transformation policy.

Name the policy friends_xml2html.

- Select the XSLT file from the uploaded bundle (friends.xsl, in this example).
- Save the policy.

  For more policy creation details, see the ADS usage steps in *AON Development Studio Guide* and AMC usage steps in *AON Administration and Installation Guide*.

**Step 4** Step 4) Download the new policy files from AMC.

- Using ADS, click **Sync** to receive the newly created policy files from AMC.

**Step 5** Step 5) Create a new PEP including the Transform bladelet.

- Using ADS, create a new PEP.
- Add the Transform bladelet to the PEP.
- In the Transform bladelet properties screen, under the Stylesheet drop down menu, select the policy file friends_xml2html.

**Step 6** Step 6) Upload the PEP and message type to AMC.

After setting all the bladelet parameters and creating the corresponding message type:

- Using ADS, click **Sync** to upload the PEP and message type to AMC.

**Step 7** Step 7) Deploy the file to AON.

Using AMC, select **Deploy**.

**Step 8** Step 8) Send the XML file to the end server.

- Using any HTTP client software (such as SOAPTest or SOAPBox), send the friends.xml file to the end server.

  For example, you could use an echo server written in JSP as the end server. This server bounces the message back to the client.
- Set the client proxy pointing to the AON address and port.

## Java Plugin Transformation Example

In this example, HTTP client SOAPTest sends friends.xml to AON over HTTP. AON reads the incoming Comma Separated Values (CSV) message, associates it with the message type (and therefore, with a Policy Execution Plans), and transforms it to XML.

**Note** CSV is a simple text format used for importing to and from spreadsheets, HTML editors, and SQL databases.

Example files:

- Input file = friends.csv
- XSLT file = identity.xsl
- Parser Plugin = parserplugin.CSV2XML_Req.far (zipped file)
- Output = XML message
- Transform bundle = transform.xfn (zipped file containing bundle component files)
- Parser bundle = myparser.xfn (zipped file containing bundle component files)

**Note** The bundle files (.xfn) are not shown.

**Step 1** Create a Content Parser bundle and save it.

- Using ADS, create a Content Parser including the sample.jar.
- Save this file on the local hard disk as myparser.xfn.

**Step 2** Upload and register the bundle.

- Using AMC, select **Admin > Extensions > Transform Parser** packages.
- Click **Upload** to upload the myparser.xfn file.
- Click **Register**.

**Step 3** Select a new transformation policy and save it.

- Using AMC, select **Properties > Application > Global > Content Parser**.
- Click **New** and select a new transformation policy.

  Name the policy "CSV Friends Reader."
- Select the java class file (sample.csvplugin.newCSVReader) from the uploaded bundle to be used for transformation.
- Save the policy.

  See the *AON Development Studio Guide* for detailed descriptions of these steps.

**Step 4** Select a new transformation policy and save it.

- Using AMC, select **Properties > Application > Global > Transformation**.
- Click **New** and select a new transformation policy.
- Select the xslt file (identity.xsl) from uploaded bundle to be used for transformation.

  Name the policy "Identity."
- Save the policy.

  See the *AON Development Studio Guide* for detailed descriptions of these steps.

**Step 5** Synchronize.

- Using ADS, click the **Sync** button to receive all the newly created policy files from AMC.

**Step 6** Create a new PEP including the Transform bladelet and set parameters.

- Using ADS, create a new PEP.
- Add the Transform bladelet.
- In the Transform bladelet properties, under the Stylesheet drop down menu, select the policy file Identity.
- In the Content Parser drop down menu, select the "CSV Friends Reader" policy.

**Step 7** Create the corresponding message type and upload it to AMC.

- Using ADS. create the message type.
- Click **Sync** to upload the message type to the AMC.

**Step 8** Deploy the file to AON.

- Using AMC, select **Deploy** to deploy the file to AON.

**Step 9** Send the csv file to the end server.

- Using any HTTP client software (such as SOAPTest or SOAPBo), send the file to the end server.

  For example, you could use an echo server written in JSP as the end server. This server bounces the message back to the client.
- Set the client proxy pointing to the AON address and port.

# Transform Extension Information File

Transformation services involve the Transform and Content Parser packages. These files are created using the AON Development Studio (ADS) and uploaded to AON via the AON Management Console (AMC). Both packages are based on the transform extension information file (transform-info.xml), created in the AON Development Studio (ADS), and uploaded to the AON Management Console (AMC).

This section describes the Transform Extension Information File. To learn how to create and upload these packages, see the following sections and chapters in this document:

- "Creating the Custom Bladelet" in Chapter 2, "Custom Bladelets".
- "Packaging the Custom Adapter" in Chapter 3, "Custom Adapters".

Also, see the *AON Development Studio Guide* and the *AON Administration and Installation Guide*.

## File Layout

The general layout of transform-info.xml is shown below. The package developer uses an editor to insert transform-specific information into the <*packagetype*Info>... </*packagetype*Info>section.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<TransformExtensionInfo
            xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">
    <packagetypeInfo>... </packagetypeInfo>
    <AONVersion version = "1.0"/>
</TransformExtensionInfo>
```

The rest of the file is left unchanged. After the package (transform or content parser) is created, you use the ADS and AMC windows to incorporate it into AON.

## Transform Packages

A transform package defines the XSL transform and Java archive (.jar) files that contain Java extensions used by the XSL Transforms. The package contains the transform-info.xml file which specifies the XSLT transform extensions defined by the package.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<TransformExtensionInfo
            xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance">
    <TransformInfo>
        <XSLTTransformName name = "friends.xsl"
                                                          display-name = "XML to HTML
Transform"
                                                           version = "1"/>
    </TransformInfo>
```

```
    <AONVersion version = "1.0"/>
</TransformExtensionInfo>
```

The file sections are described in Table 6-2.

*Table 6-2 Transform Packages File Section*

| File Section | Description |
|---|---|
| <TransformExtensionInfo> ... </> | This is the root element in transform-info.xml. All extensions are children of this section. This section includes one or more TransformInfo sections. |
| <TransformInfo> ... </> | This section specifies an XSLT extension that is defined in the Transform package. |
| name | This attribute specifies the name of the XSLT file (physical file name including the file extension) included in the package. You must create a TransformInfo section fro each XSLT file that is specified in a Transform policy. If you include an XSLT file in other XSLT files, it does have to be specified. |
| display-name | This attribute specifies a display name or a more descriptive name of the extension. It appears in AMC after the package is uploaded. |
| version | This attribute specifies the extension version. The developer who creates the package sets this field. It appears in AMC after the packaged is uploaded. |
| AONVersion | This section specifies the version of AON in which this package is used. |

For more information, see Creating a Transformer Extension, page 5-3 and the description of packaging in *AON Development Studio Guide*.

# Content Parser Packages

A Content Parser package defines a content parser plugin and transformer plug-in that are used in an AON Transformation. The package contains the transform-info.xml file which specifies the package-defined parser plug-in and transformer plug-in extensions.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<TransformExtensionInfo
       xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" >
    <ParserPluginInfo>
        <ParserClassName
                            class-name = "sample.csvplugin.CSVfriendsReader"
                               display-name = "CSV Friends Parser"
                                version = "1"/>
        <ParserClassName
                               class-name = "sample.plugin.TransformerFactory"
                               display-name = "Sample Transformer Factory"
                                version = "1"/>
    </ParserPluginInfo>
    <AONVersion version = "1.0"/>
</TransformExtensionInfo>
```

The file sections are described in Table 6-3.

*Table 6-3 Content Parser Package File Sections*

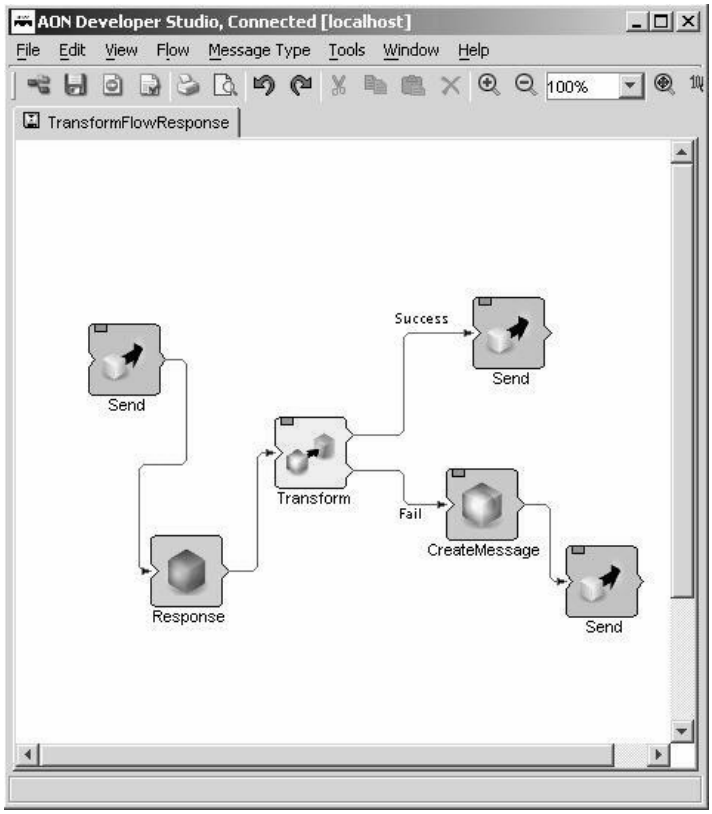
| File Section | Description |
|---|---|
| <TransformExtensionInfo> ... </> | This is the root element in the transform-info.xml file. All extensions are children of this section. It includes one or more ParserPluginInfo sections. |
| <ParserPluginInfo> ... </> | This section specifies a Java class name that implements a Parser Plugin or a Custom Transformer factory. You implement a Parser Plugin by implementing the XMLReader class. You implement a Custom Transformer by implementing the AONSTransformerFactory class.<br><br>In the sample above, a Content Parser class sample.csvplugin.CSVfriendsReader implements XMLReaser class while sample.plugin.TransformerFactory implements AONSTransformerFactory.<br><br>If the package only contains a Content Parser, you only have to specify the class that implements XMLReader. |
| class-name | This attribute specifies the name of the Java class that implements the XMLReader or AONSTransformerFactory class. |
| display-name | This attribute specifies a display name or a more descriptive name of the extension. It appears in AMC after the package is uploaded. |
| version | This attribute specifies the extension version. The developer who creates the package sets this field. It appears in AMC after the packaged is uploaded. |
| AONVersion | This section specifies the version of AON in which this package is used. |

For more information, see Creating a Content Parser Extension, page 5-7 the description of packaging in the *AON Development Studio Guide*.

# Using XSLT Transformation

To use XSLT transformation, you design a message Policy Execution Plans (PEP) that uses various AON bladelets (including transformation) to extract data, transform it, and compose the output message with transformed content. Figure 6-7 illustrates this process for a simple XML to HTML transformation on a message received in response to an HTTP GET request by an AON node.

*Figure 6-7 PEP Incorporating Transform Bladelet*




1. A client application or browser issues a request to fetch an XML Document. This PEP is designed to act on a response message to this type of request.
2. Transform bladelet
   - Extracts the content from the Response Message using a PEP variable that holds Response Message.
   - Applies the XSL Transform (specified in the Friends.xsl, page 6-15) which converts an XML message to an HTML message. This style sheet is made available to AON by uploading a transform package in AMC and provisioning it to AON.
3. The transformation result is placed in Response Message if the message is successfully transformed. Otherwise, an error message is created using a Create Message Bladelet and an error message is sent to the client application.

**Note** This sample PEP was created in the AON Development Studio (ADS) PEP Developer.

For descriptions of the key Transformation bladelet and others, see the *AON Development Studio Guide*.

# Sample Transformation Files

The following files listed below show sample XML input and XSL transformation:



## Friends.xml

The following XML file is input for this sample.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <contacts>
- <friend>
<name>Pat Smith</name>
<address>1234 Main Street</address>
<address2 />
<city>San Jose</city>
<state>California</state>
<country>US</country>
<dob>June 06, 1976</dob>
</friend>
- <friend>
<name>Lynn Jones</name>
<address>13423 First Avenue</address>
<address2>Apt 7071</address2>
<city>San Jose</city>
<state>California</state>
<country>US</country>
<dob>Nov 02, 1978</dob>
</friend>
- <friend>
<name>Terry Johnson</name>
<address>111762 State Street</address>
<address2 />
<city>San Jose</city>
<state>CA</state>
<country>US</country>
<dob>Feb 24, 1977</dob>
</friend>
</contacts>
```

## Friends.xsl

The following style sheet transforms the data in a friends XML document into HTML, and formats that data into a table.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
- <xsl:template match="/">
- <html>
- <body>
<h2>My Friends</h2>
<hr />
- <font face="arial">
- <table border="0" cellspacing="2" cellpadding="2">
- <xsl:for-each select="//friend">
- <tr>
```

```
<td bgcolor="orange" align="left">Name</td>
- <td bgcolor="orange">
<xsl:value-of select="name" />
</td>
</tr>
- <tr>
<td bgcolor="beige" align="left">Address</td>
- <td bgcolor="beige">
<xsl:value-of select="address" />
,
<xsl:value-of select="address2" />
</td>
</tr>
- <tr>
<td bgcolor="beige" align="left">City</td>
- <td bgcolor="beige">
<xsl:value-of select="city" />
</td>
</tr>
- <tr>
<td bgcolor="beige" align="left">State</td>
- <td bgcolor="beige">
<xsl:value-of select="state" />
</td>
</tr>
- <tr>
<td bgcolor="beige" align="left">Country</td>
- <td bgcolor="beige">
<xsl:value-of select="country" />
</td>
</tr>
- <tr>
<td bgcolor="beige" align="left">Date of Birth</td>
- <td bgcolor="beige">
<xsl:value-of select="dob" />
</td>
</tr>
- <tr>
<td />
<td />
</tr>
</xsl:for-each>
</table>
</font>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

# Transformation APIs

The key APIs, AONSTransformer and AONSTransformerFactory are summarized below. For additional information, see individual bladelet descriptions in the *AON Development Studio Guide*.

# AONSTransformer

AONSTransformer defines a transformer object in AON to perform transformation services. The interface takes XML data as input (SAXSource or DOMSource) and puts the transformed result in the target result object.

AONSTransformer is implemented in the AON transformation service that provides XSLT based transformation in AON. An AON transformer object is created by AONSTransformerFactory. For a more details, see the AONSTransformer, page 5-12.

# AONSTransformerFactory

AONSTransformationFactory defines a factory for an AON transformer object. Each type of transformation implements this interface to provide a specific transformer object factory. Two extensions are possible to Transformation Service in AON. For a more details, see the AONSTransformerFactory, page 5-13.

CHAPTER 7

# Schema Validation

The AON provides two forms of schema validations services: conventional (Java-coded) and optimized (C-coded). Both services provide message schema validation. This chapter introduces these services in the following sections:



For more information, see the *AON Development Studio Guide* and *AON Administration and Installation Guide*.

## Schema Validation Prerequisites

Before incorporating schema validation into a message Policy Execution Plans (PEP), you must:

- Use AON Development Studio (ADS) windows to create one or more schema packages

  A schema package includes the package name, version, vendor name, description, package properties, and one or more schema files, XML schema (.xsd), or Document Type Definition (.dtd) files.

- Use the AON Management Console (AMC) to upload and register the new schema package.

For directions, see the *AON Installation and Administration Guide* and the *AON Development Studio Guide*. Also, see Conventional and Optimized Schema Validations, page 7-1.

## Conventional and Optimized Schema Validations

AON provides two ways to validate schemas: the conventional or optimized processing. Both services use Xerces and Tarari validating parsers. These processes are described in the following sections:



For more information, see Schema Validation.

# Conventional Schema Validation

AON provides conventional validation for messages handled by normal AON processes. This process involves schema validation.The role of the Validate bladelet in conventional schema validation is discussed in Validate Bladelet, page 7-2. For more information, see Packaging Schema Validations, page 7-3.

## Validate Bladelet

You include the Validate bladelet (supplied with AON) in message PEPs to enable validation. Generally, AON can validate incoming XML messages to verify their conformity to a particular schema or DTD.

AON uses the Apache Xerces and Tarari validating parsers to validate XML schemas and DTDs. It loads the provisioned schemas and DTD files into cache at bootstrap time and uses the cached grammars for runtime validation. Because of this, grammar files do not have to be parsed each time for every incoming message that is to be validated.

If the XML messages that require validation have an embedded grammar (XSD or DTD) declaration, AON will expect to find that schema or DTD in its cache of preparsed grammars. If not found, AON will not be able to validate the message. AON will not fetch DTDs or Schema files from external references, if indicated in the message.

Moreover, AON can also validate an XML message that does not have a grammar declaration. For example, if the message has undergone transformation in AON, and the message needs to conform to a particular endpoint schema, then AON Validate Content bladelet can be used to validate the message. In this case, the Bladelet needs to be configured with the appropriate Schema Policy reference, indicating the Schema file that needs to be used.

The ADS windows can be used to configure the following parameters for Schema Validation.

- Validate Schemas and/or DTDs – AON can be configured to validate only those XML messages that have a schema reference, or only those messages that have a DTD declaration, or both types of messages.
- Validate Properties – The Validate bladelet > Validate Properties > Advanced Input Parameters is used to set the advanced options listed below.
  - Limit Number of Node Occurrences–Disable (default) or Enable.
  - Limit Number of Entity Expansions–Disable (default) or Enable.
  - Action for Messages With No Grammar Available–Disallow Message (default) or Validate with Default Schema.

    When this action is allowed, AON uses the default schema to validate XML messages with no available grammar.
  - Full Schema Constraint Checking–Checked (default) or unchecked.

    When checked, this validation process reviews the schema grammar for additional errors that are time-consuming or memory intensive. Currently, the following elements of schema validation belong to Full Schema Validation.

    - particle unique attribution constraint checking and

    - particle derivation restriction checking

For more information, see the description of the ContentValidation interface on ContentValidation, page 5-17 and the Validate bladelet in the *AON Development Studio Guide*.

## Optimized Schema Validation

The Optimized schema validation service is provided through the optimized Validate bladelet and optimized schema validation PEP. The optimized PEP (including the optimized Validate bladelet) is used to set input parameters and constraints for messages that should be validated against a schema.

This service validates incoming XML messages to verify that they conform to a particular schema. The schemas are packaged in ADS and uploaded and provisioned to AON from AMC. If validation fails, the message is rejected or dropped and an error is returned to the client.

In summary, optimized schema validation:

- Validates XML messages for conformance with the schema.
- Check to be certain that the XML message is properly formed.
- Supports the W3C XML schema validation.
- Supports namespace-based schema validation.
- Supports schema package updates.
- Does not retrieve external schemas.
- Uses Tarari validating parser.

For more information, see the description of the Validate bladelet in the *AON Development Studio User Guide*. For additional information, see Developing a Custom Bladelet for Fastpath Processing, page 4-1, and Packaging Schema Validations, page 7-3.

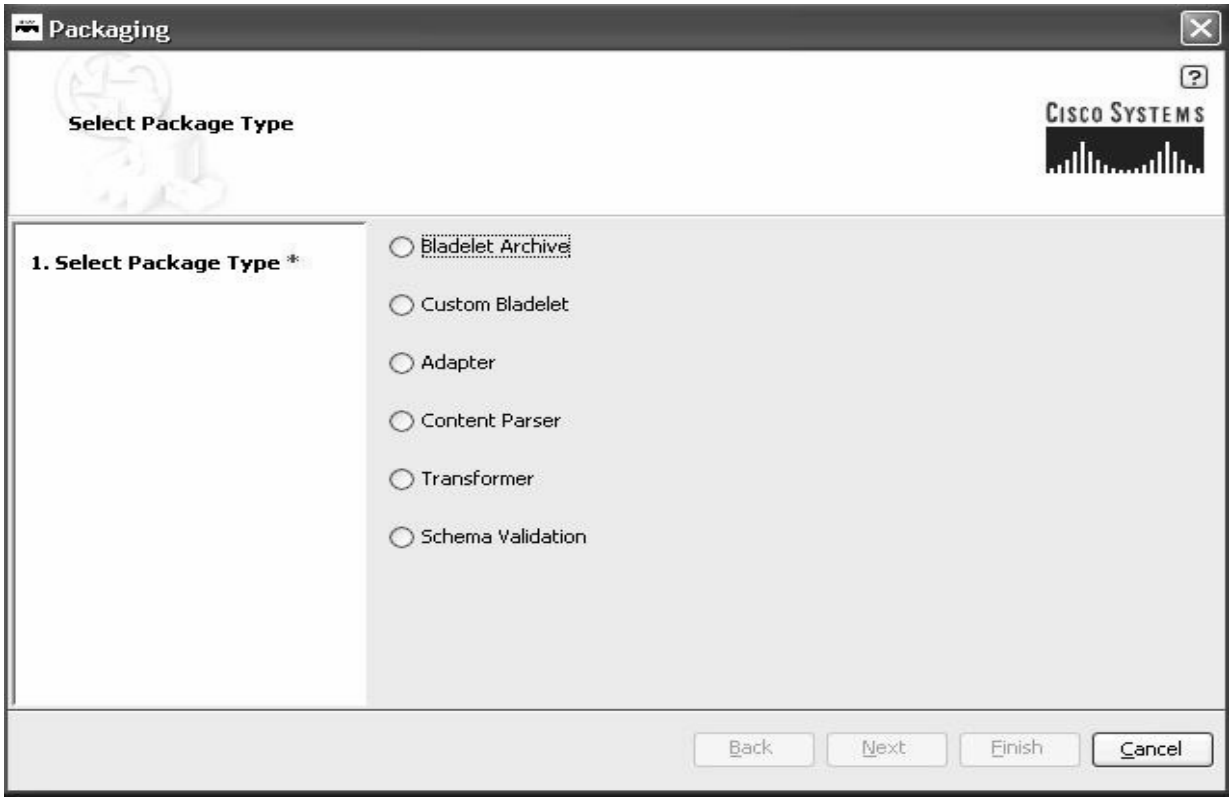
# Packaging Schema Validations

You use the same ADS windows to package schema validations for either conventional or optimized message processing. Later, you use AMC windows to upload the package to AON. Follow the steps listed below.

**Step 1** Using the ADS, select **Tools > Packaging**.

The AON Packaging window appears. For example:

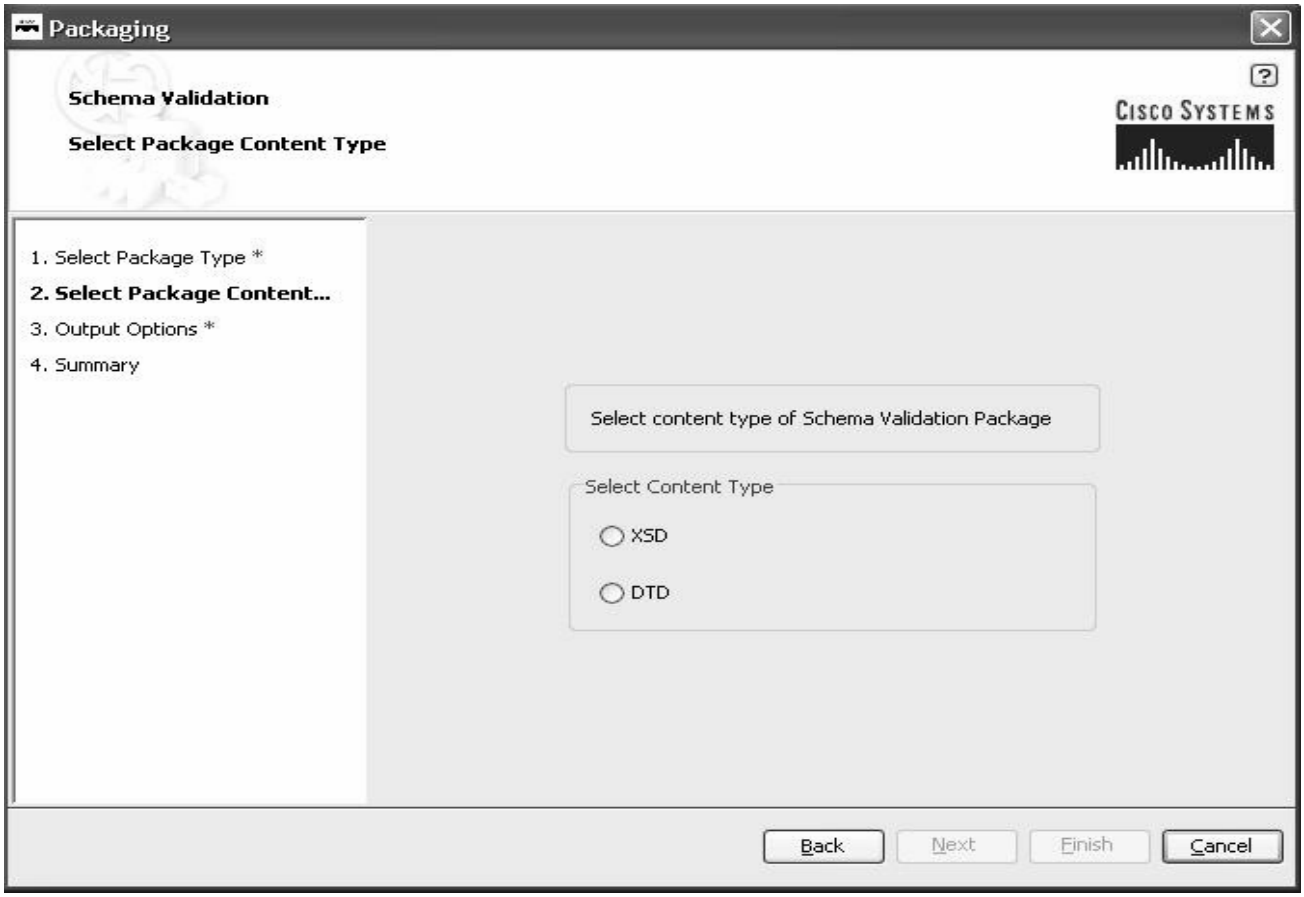
*Figure 7-1 AON Packaging window*




**Step 2** Select **Schema Validation** and click **Next**.

The Select Package Content Type screen appears.

*Figure 7-2 Package Content Type Selection*




**Step 3** Select the Content Type and click **Next**.

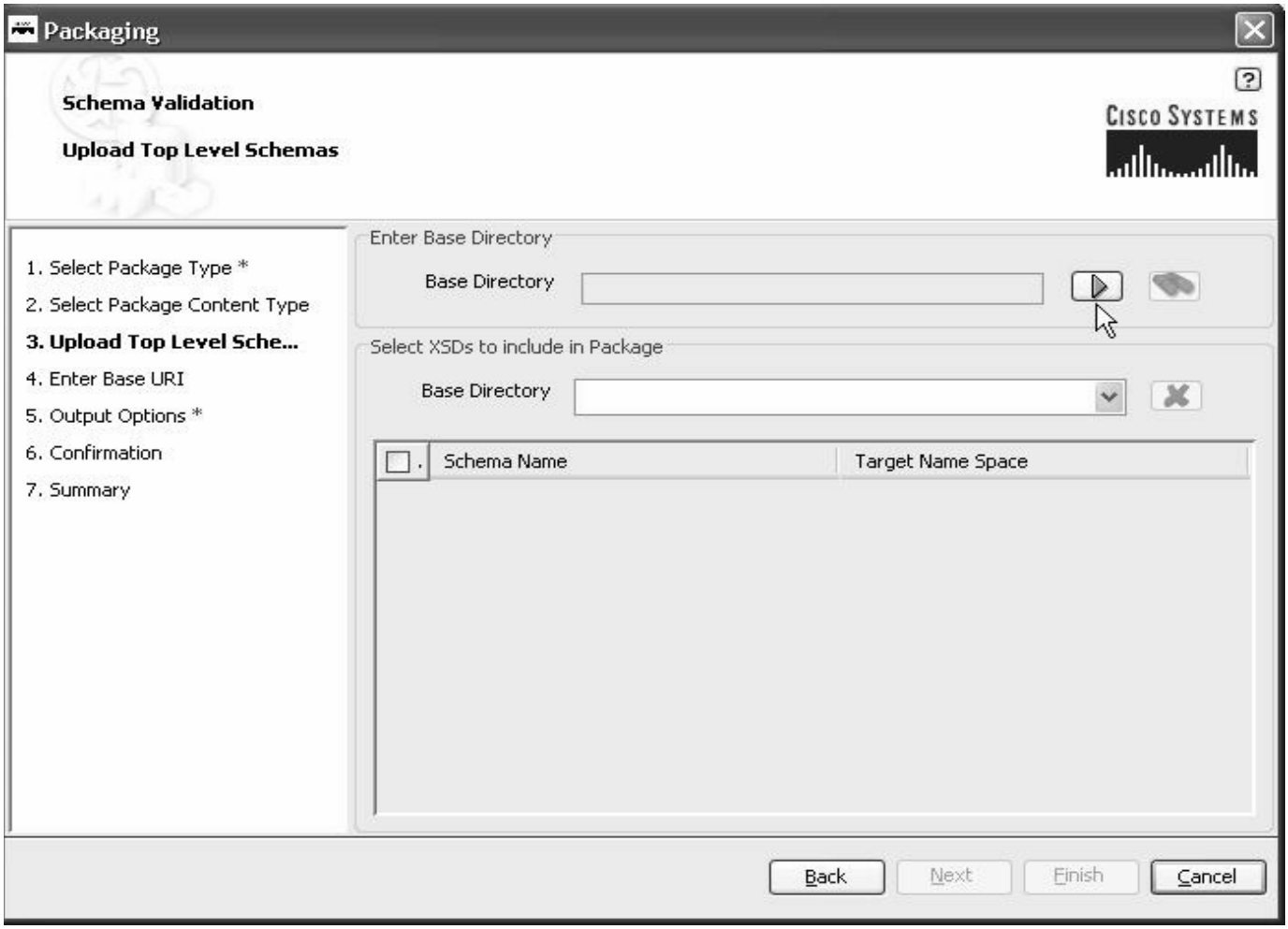
For conventional message processing, select XSD or DTD. For optimized processing, select XSD.

To learn about optimized message processing, see Schema Validation.

**Step 4** Complete the schema validation packaging process.

- If you select XSD, the Upload Top Level Schema screen appears:

*Figure 7-3 Upload the Top Level Schemas*



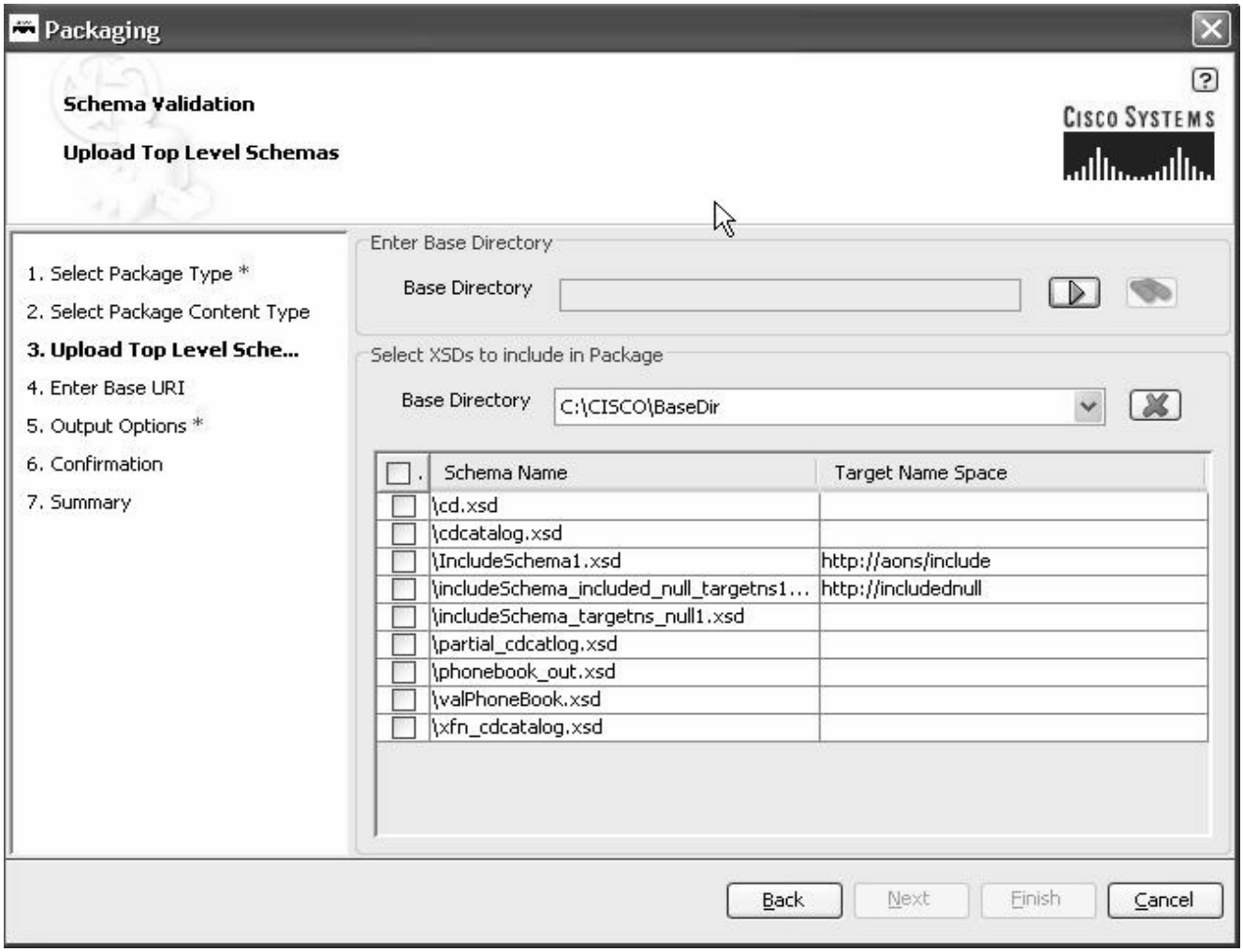

– Using the right-arrow button, select the base directory.

  If you press the binoculars icon (to the far right of the Base Directory field), the system will identify any files that are missing in your base directory. For example:

*Figure 7-4 Missing Schema Components Message*


– Click the binocular icon (to the right) to populate the Schema Name list. For example:

*Figure 7-5 Populating the Base Directory Schema Name List*

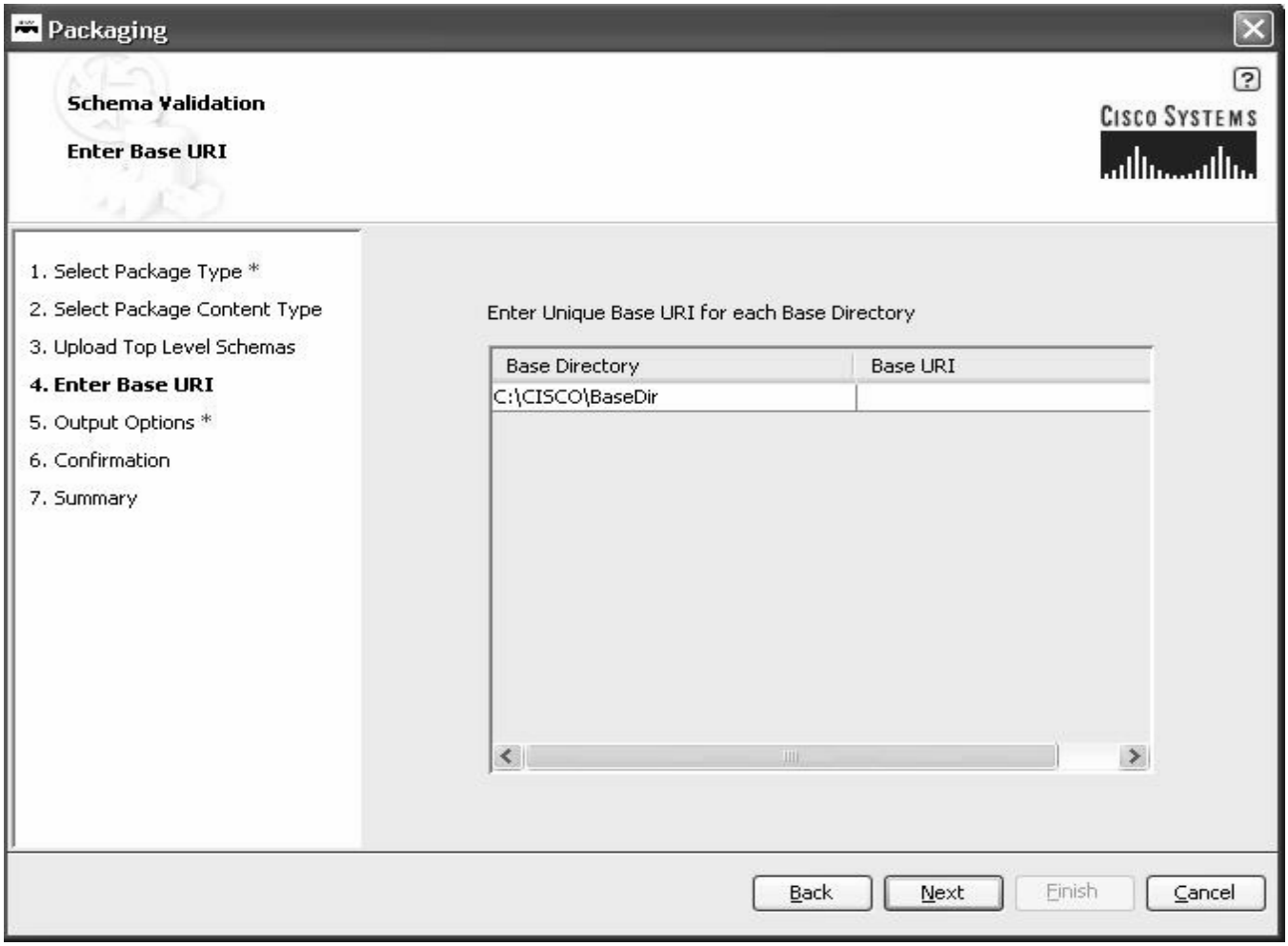



In this example, BaseDir was selected as the Base Directory.

– Check the schemas to be included in the package and click **Next**.

The Schema Validation - Enter Base URI screen appears. For example:

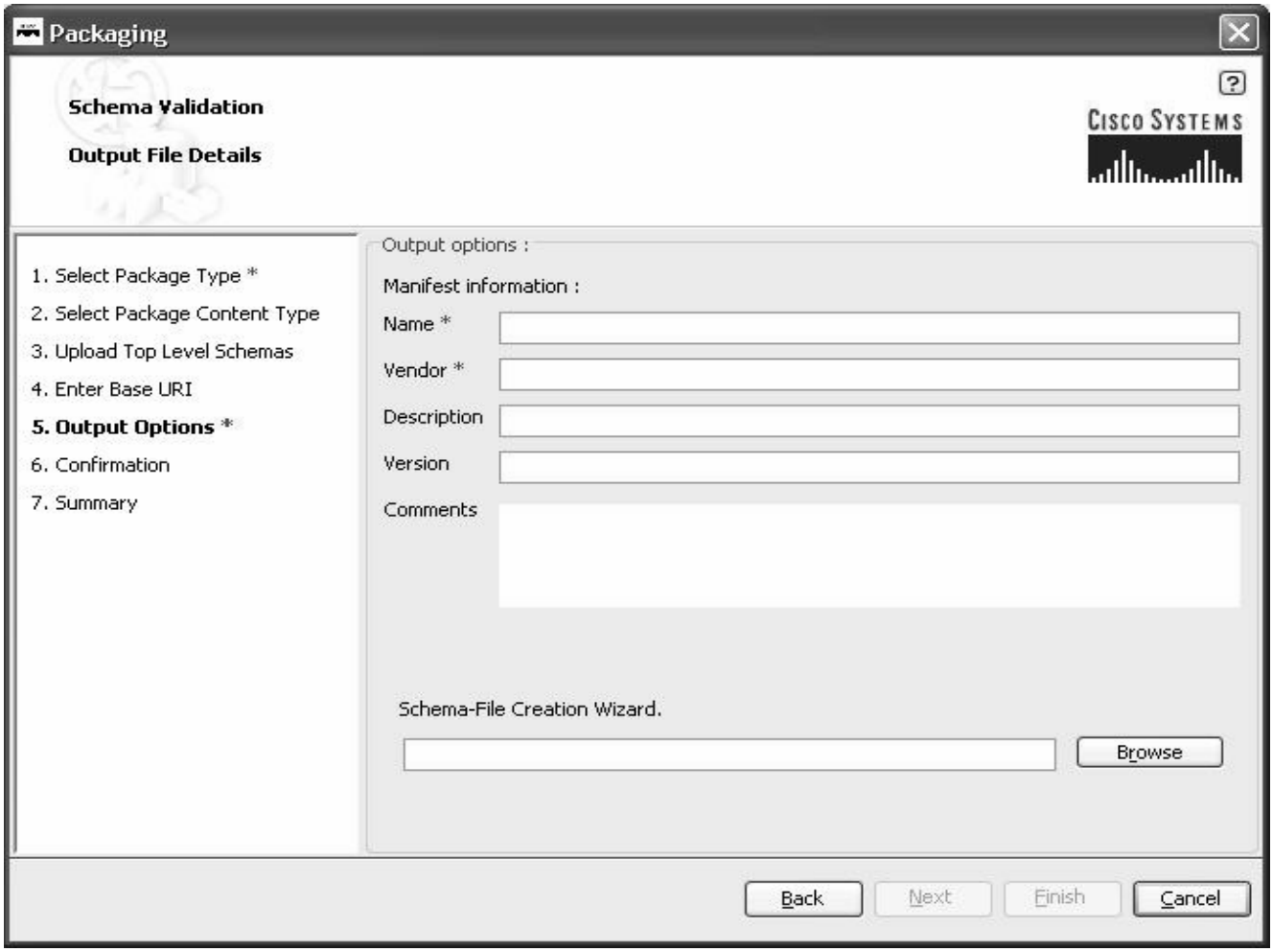
*Figure 7-6 Entering Base URI*




– Fill in the Base URI and click **Next**.

The Schema Validation - Output File Details screen appears. For example:

*Figure 7-7 Output File Details window*




– Fill in the fields.

  The screen fields are described below.

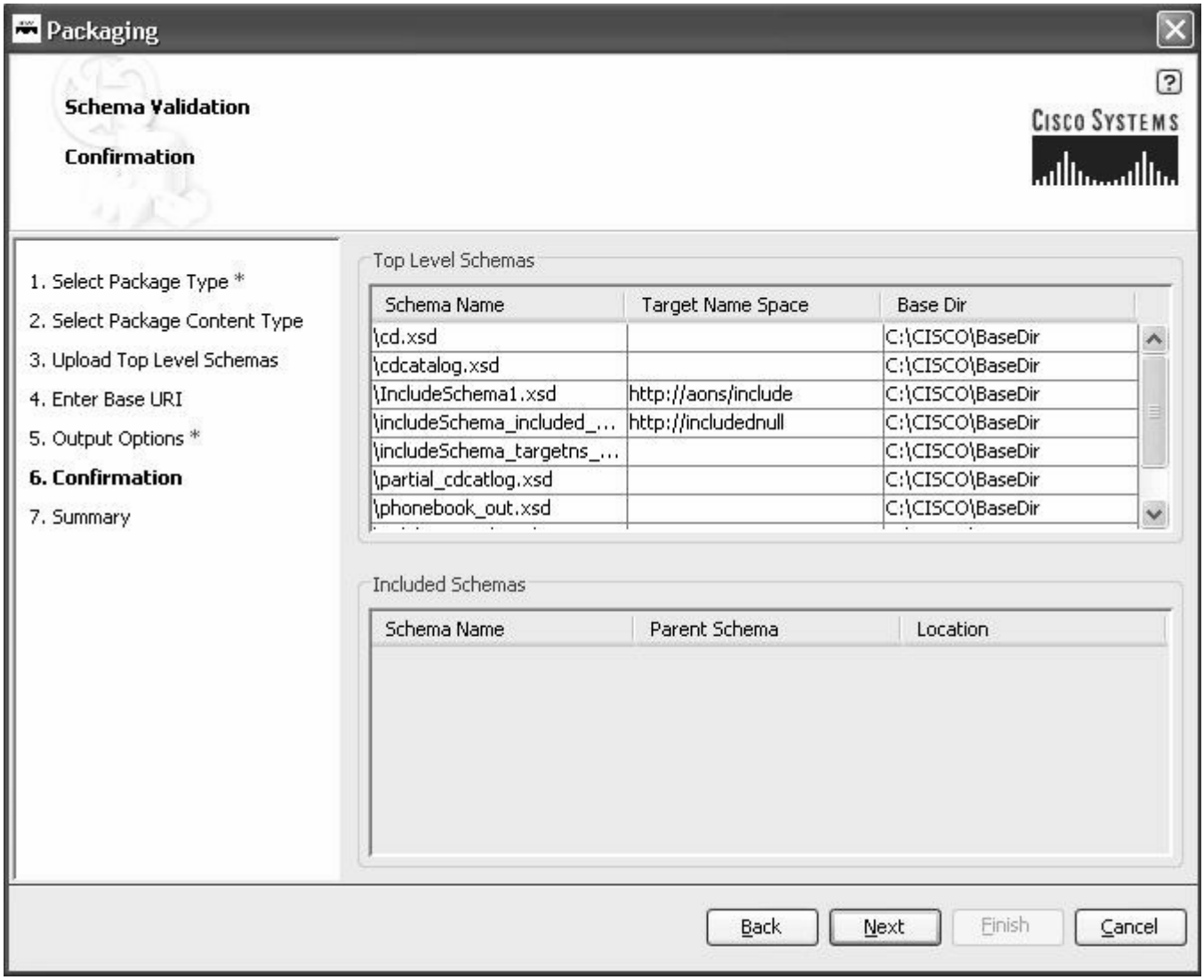
| Field | Definition |
|---|---|
| Name | Package name. Mandatory. |
| Vendor | Vendor. For example, Cisco Systems, Inc. Mandatory. |
| Description | (Optional) Package description. |
| Version | (Optional) Package version number. |
| Comments | (Optional) Comments about the schema validation package. |

– By default, the Schema-File Creation Wizard field fills in with a location for the new .sar file. You may use the Browser button to find a different location for the .sar file.

– Click **Next**.

  A Confirmation screen appears. For example:

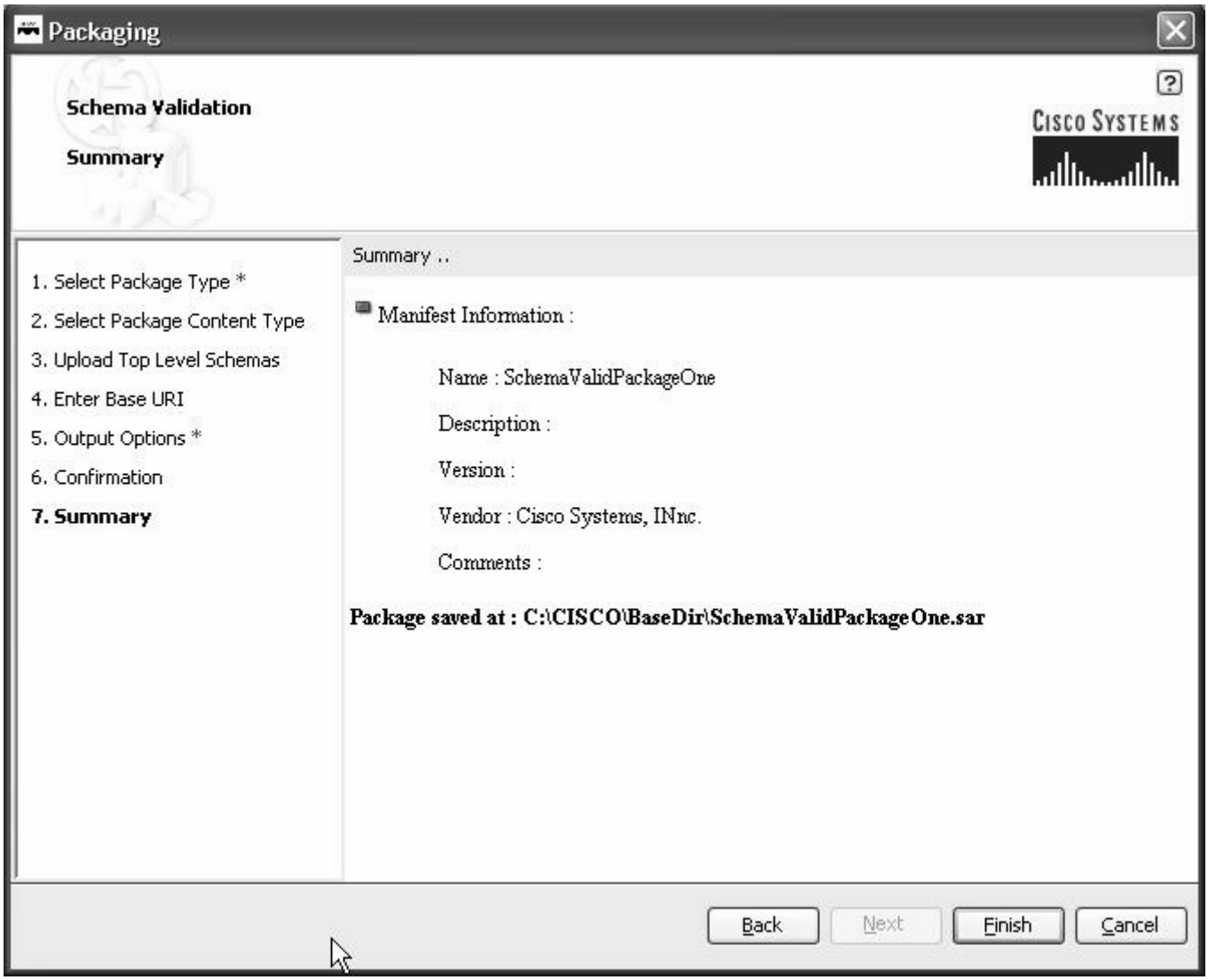
*Figure 7-8 Schema Validation Confirmation window*




– Click **Next**.

A summary screen appears. For example:

*Figure 7-9 Package Summary window*

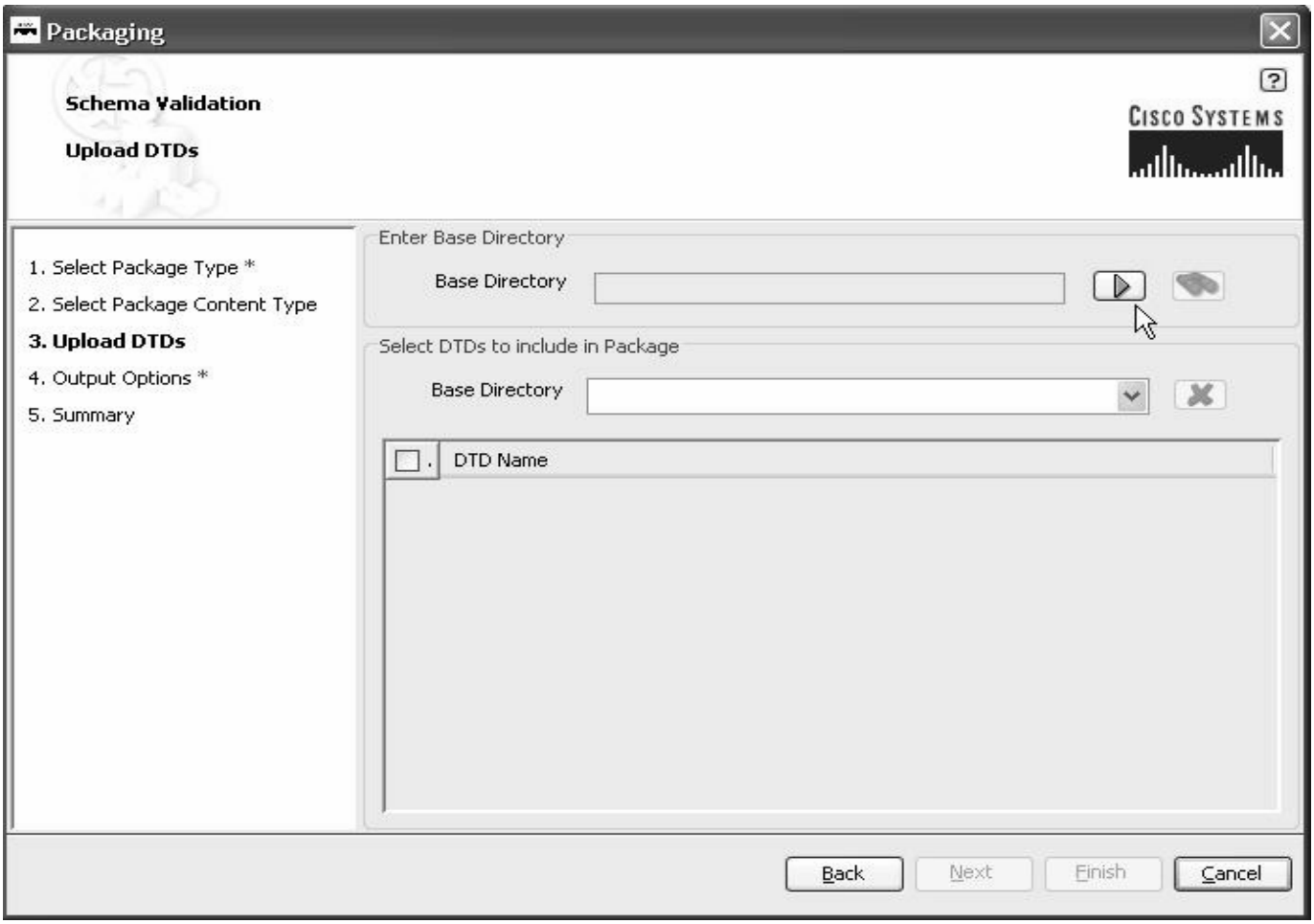



  - Click **Finish.**

    A popup dialog indicates success.
  - Click **OK.**

    The ADS screen appears.
- If you select DTD, the Upload DTDs screen appears:

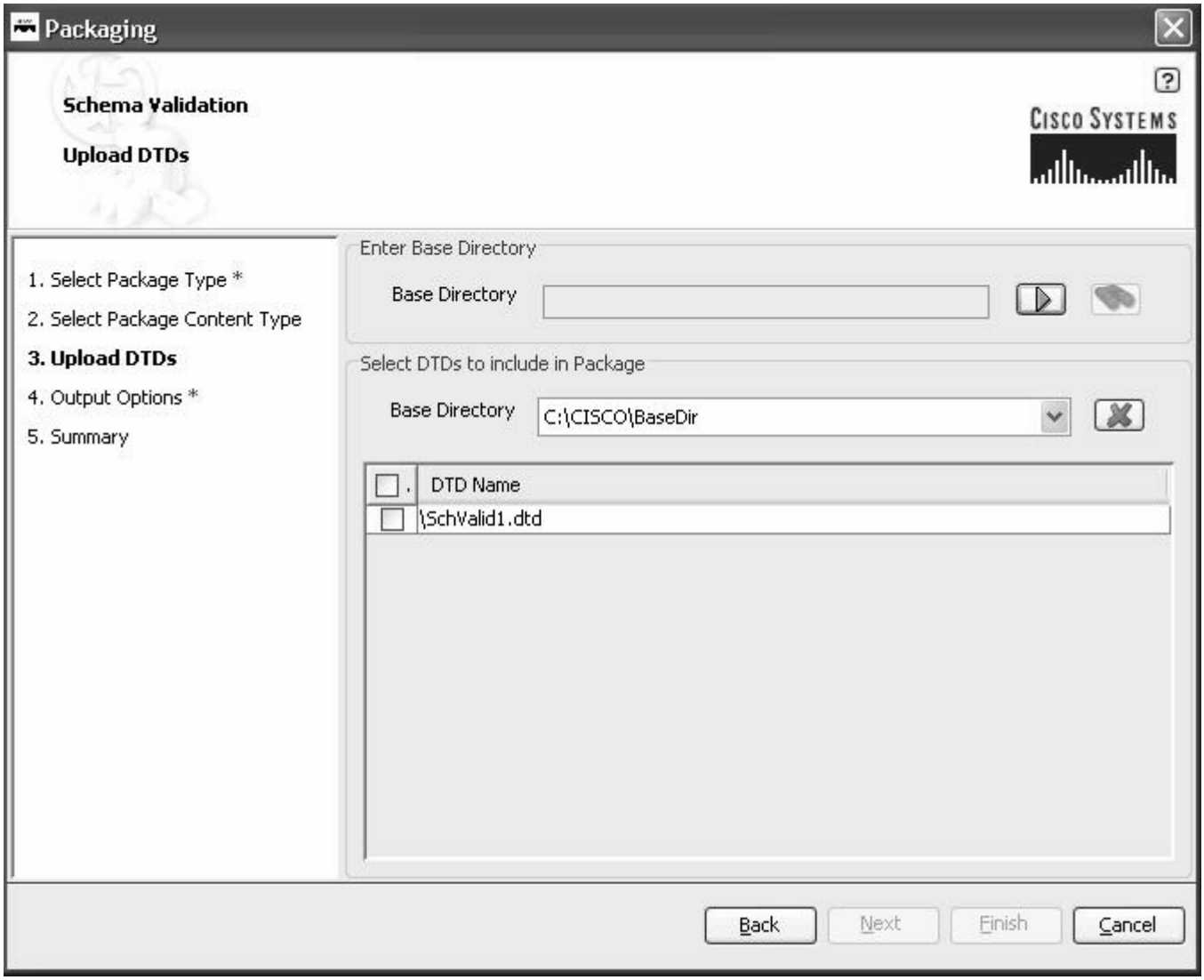
*Figure 7-10 Upload DTDs window*




- Using the right-arrow button, select the base directory.
- Click the binocular icon.

  The screen changes and the .dtd is listed under DTD Name. For example:

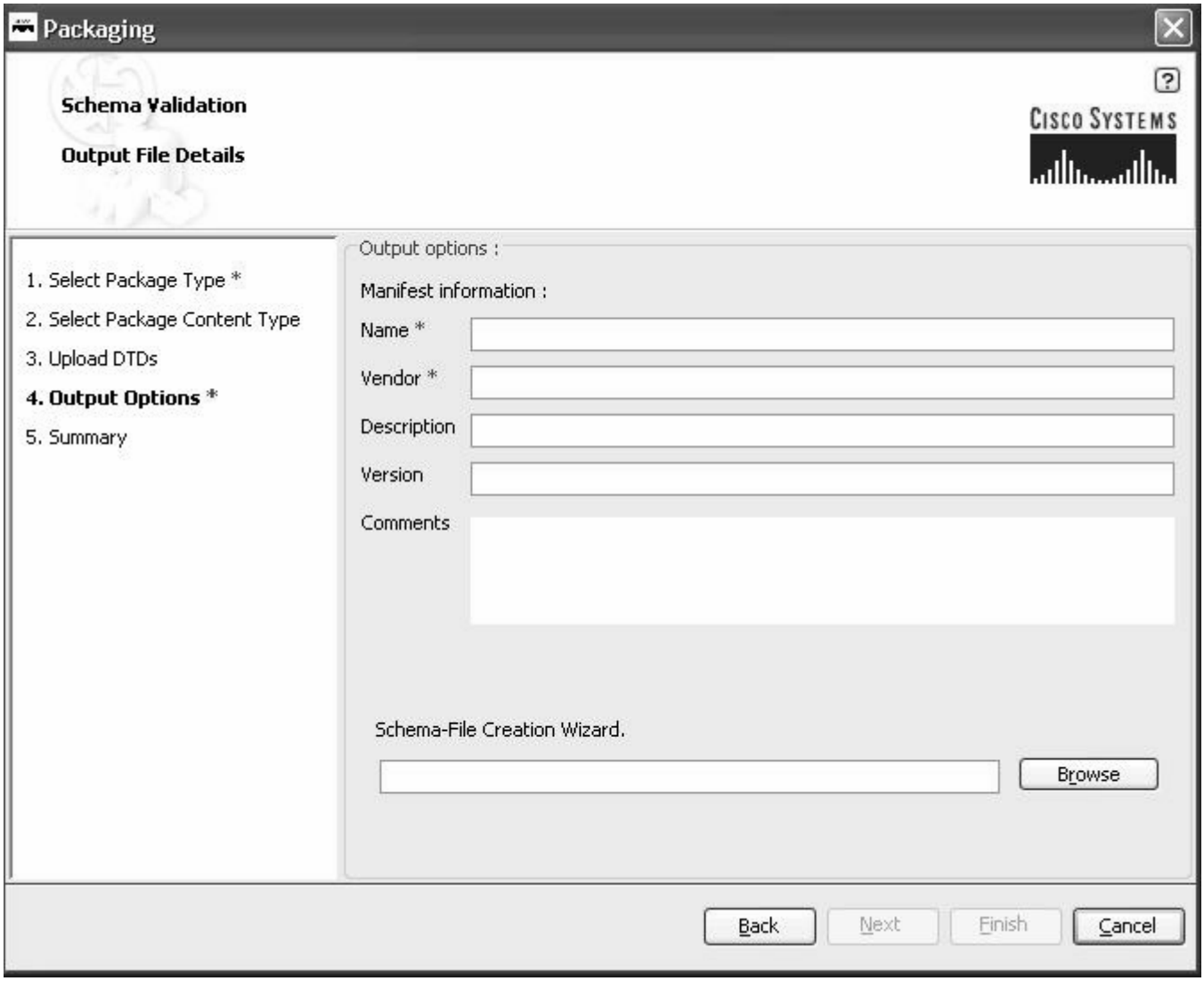
*Figure 7-11 DTDs Selected window*




– Check the .dtds to be included in the package and click **Next**.

An Output File Details screen appears. For example:

*Figure 7-12 Output File Details window*



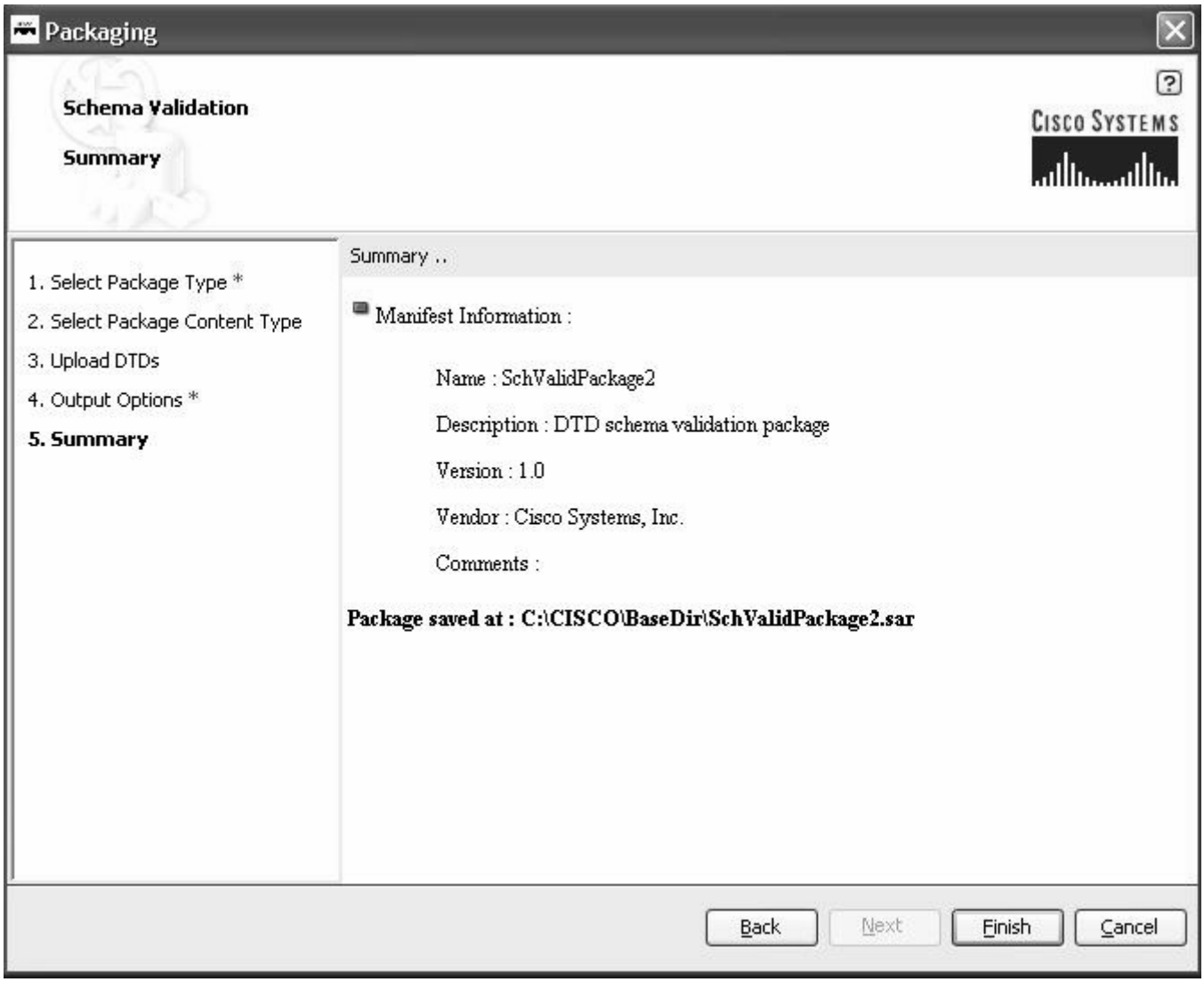

The screen fields are described below.

| Field | Definition |
|---|---|
| Name | Package name. Mandatory. |
| Vendor | Vendor. For example, Cisco Systems, Inc. Mandatory. |
| Description | (Optional) Package description. |
| Version | (Optional) Package version number. |
| Comments | (Optional) Comments about the schema validation package. |

- By default, the Schema-File Creation Wizard field fills in with a location for the new .sar file. You may use the Browser button to find a different location for the .sar file.
- Click **Next**.

A Confirmation screen appears. For example:

*Figure 7-13 Schema Validation Confirmation window*




– Click **Finish.**

  A popup dialog indicates package creation success.

– Click **OK.**

  The ADS screen appears.

To learn how packages are uploaded to the AON Management Console (AMC) and incorporated into AON, see the *AON Administration and Installation Guide*.

APPENDIX A

# AONSCommon Specification

AONSCommon packages are used in conjunction with Custom Bladelet, Custom Adapter, External Service, XSLT Transformation and Schema Validation software development kits (SDKs). The AON Common (aonscommon) file includes the following groups of associated classes (identified as packages):



## Exception Package

The AONSCommon Exception package (**com.cisco.aons.exception**) includes a class and a set of exceptions. These elements are discussed in the following sections:



### Class

The exception package includes one class: ExceptionType.

### ExceptionType

Extended from java.lang.Object, this class (com.cisco.aons.exception.ExceptionType) is used to return the exception type. It includes the fields and method summarized below.

| Field | Description |
|---|---|
| Application | public static final ExceptionType APPLICATION<br>Application type. |
| Recoverable | public static final ExceptionType RECOVERABLE<br>Recoverable type. |
| System | public static final ExceptionType SYSTEM<br>System type. |

| Method | Description |
|---|---|
| getType() | public abstract int getType()<br>Returns the type of exception: application, recoverable, or system types. |

## Exceptions

The following AONSCommon exceptions are used:

- AONSException, page A-3
- AONSRuntimeException, page A-6
- ExtServiceException, page A-7
- InitializationException, page A-9
- NamedExtenstionException, page A-8
- NoSuchVariable, page A-10

These exceptions are summarized in the following sections.

## AONSException

The is the root AON exception (com.cisco.aons.exception.AONSException). It includes the methods summarized below.

| Method | Description |
|---|---|
| create | public static final AONSException create (java.lang.String pErrorCode)<br>Parameters:<br>pErrorCode—String<br>Returns: An AONSException based on the error code. |
| create | public static final AONSException create(java.lang.String pErrorCode, Exception pEnum)<br>Parameters:<br>pErrorCode—String<br>pEnum—ExceptionType<br>Returns an AONSException based on the error code and type |
| getArgs | public final java.lang.Object[] **getArgs()**<br>Gets object arguments. |
| getErrorCode | public int getErrorCode()<br>Gets the error code. |
| getErrorResourceKey | public final java.lang.String getErrorResourceKey()<br>Returns an error key used to lookup aonsErrors.properties for the error message. |
| getErrorResourceKey | public static final java.lang.String getErrorResourceKey(java.lang.String pPrefix, int pErrorCode)<br>Parameters:<br>pPrefix—gives the module of the exception<br>pErrorCode—error code of the exception<br>Throws: java.util.MissingResourceException<br>Returns an error key consisting of the parameters that are used to lookup aonsErrors.properties. |
| getExceptionEnum | public final ExceptionType getExceptionEnum()<br>Gets the exception number. |
| getPrefix | protected java.lang.String getPrefix()<br>Returns: String containing a three letter acronym that identifies the component or module that threw the exception. It defaults to a system acronym. |

| Method | Description |
|---|---|
| getReplyMessage | public java.lang.Object getReplyMessage()<br>Returns the reply message set by the throwing class.<br>Returns the reply message object it is set by the throwing class. Otherwise, returns null. |
| setReplyMessage | public void setReplyMessage(java.lang.Object pMsg)<br>Parameters:<br>pMsg—Reply message object<br>Sets the reply message to pMsg. |
| getResourceBundle() | public static final java.util.ResourceBundle getResourceBundle(java.util.Locale pLocale)<br>throws java.util.MissingResourceException<br>Parameters:<br>pLocale<br>Gets a resource bundle. |
| getResourceBundle() | public static final java.util.ResourceBundle getResourceBundle(java.util.Locale pLocale,<br>java.lang.ClassLoader pClassLoader)<br>throws java.util.MissingResourceException<br>Gets a resource bundle. |
| getResourceMessage() | public final java.lang.String getResourceMessage()<br>throws java.util.MissingResourceException<br>Returns the resource tokens (name-value pairs) associated with the error number or an exception message if an error number is not specified. |
| getResourceMessage() | public final java.lang.String getResourceMessage(java.lang.ClassLoader pClassLoader)<br>throws java.util.MissingResourceException<br>Parameters:<br>pClassLoader—ClassLoader<br>Throws MissingResourceException if no error number is specified.<br>Returns (string) the resource tokens (name-value pairs) associated with the error number. |

| Method | Description |
|---|---|
| getResourceMessage() | public final java.lang.String getResourceMessage(java.lang.String pQualifier)<br>throws java.util.MissingResourceException<br>Parameters:<br>pQualifier<br>pQualifier—Further qualifies the resource bundle<br>Returns (string) the resource tokens (name-value pairs) associated with the error number or an exception message if an error number is not specified. |
| getResourceMessage | public final java.lang.String getResourceMessage(java.lang.String pQualifier,<br>java.lang.ClassLoader pClassLoader,<br>java.lang.String pResourceKey)<br>throws java.util.MissingResourceException<br>Parameters:<br>pQualifier—String<br>pClassLoader—ClassLoader<br>Returns: String containing the resource tokens (name-value pairs) associated with the error number or exception message if there is no error number specified. |
| getResourceMessage | public static final java.lang.String getResourceMessage(java.lang.String pPrefix,<br>int pErrorCode)<br>throws java.util.MissingResourceException<br>Parameters:<br>pErrorCode |
| getResourceMessage | public static final java.lang.String getResourceMessage(java.lang.String pPrefix,<br>int pErrorCode,<br>java.lang.ClassLoader pClassLoader)<br>throws java.util.MissingResourceException |

| Method | Description |
|---|---|
| getResourceMessage | public static final java.lang.String getResourceMessage(java.lang.String pPrefix,<br>int pErrorCode,<br>java.util.Locale pLocale)<br>throws java.util.MissingResourceException<br>Parameters:<br>pErrorCode<br>pLocale |
| getResourceMessage | public static final java.lang.String getResourceMessage(java.lang.String pPrefix,<br>int pErrorCode,<br>java.util.Locale pLocale,<br>java.lang.ClassLoader pClassLoader)<br>throws java.util.MissingResourceException<br>Parameters:<br>pErrorCode<br>pLocale |
| getResourceMessage() | public final java.lang.String getResourceMessage(java.lang.String pQualifier,<br>java.lang.ClassLoader pClassLoader)<br>throws java.util.MissingResourceException<br>Parameters:<br>pQualifier—String<br>pClassLoader—ClassLoader<br>Returns (string) the resource tokens (name-value pairs) associated with the error number or an exception message if an error number is not specified. |

## AONSRuntimeException

Extending Java class RuntimeException, this exception is used to handle AON runtime exceptions. AONSRuntimeException inherits methods from java.lang and has a set of constructors. These components are listed below.

### Inherited Methods

AONSRuntimeException inherits the following methods from class.java.lang.Throwable:

- fillInStackTrace

- getCause
- getLocalizedMessage
- getMessage
- getStackTrace
- initCause
- printStackTrace
- printStackTrace
- printStackTrace
- setStackTrace
- toString

AONSRuntimeException also inherits the following methods from class.java.lang.Object:

- clone
- equals
- finalize
- getClass
- hashCode
- notify
- notifyAll
- wait (three expressions)

**Constructors**

- public AONSRuntimeException()
- public AONSRuntimeException(java.lang.String message)
- public AONSRuntimeException(java.lang.String message, java.lang.Throwable cause)
- public AONSRuntimeException(java.lang.Throwable cause)

## ExtServiceException

This exception indicates conditions that service clients are expected to catch. ExtServiceException inherits methods from java.lang and has a set of constructors. These components are listed below.

**Inherited Methods**

It inherits the following methods from class.java.lang.Throwable.

- fillInStackTrace
- getCause
- getLocalizedMessage
- getMessage
- getStackTrace
- initCause
- printStackTrace (three expressions)
- setStackTrace

- toString

ExtServiceException also inherits the following methods from class.java.lang.Object:

- clone
- equals
- finalize
- getClass
- hashCode
- notify
- notifyAll
- wait (three expressions)

**Constructors**

- ExtServiceException(java.lang.Exception e)—Creates an ExtServiceException by wrapping an exception.
- ExtServiceException(java.lang.String message)—Creates an ExtServiceException given an input message.
- ExtServiceException(java.lang.String message, java.lang.Exception e)—Creates an ExtServiceException by wrapping an exception.

### NamedExtenstionException

Extending AONSException, this class is used to handle custom bladelet exceptions that occur during Policy Execution Plans (PEP) execution. The constructors and methods of this class are summarized below.

**Constructors**

- NamedExtensionException(java.lang.Exception exception)
- NamedExtensionException(java.lang.Exception exception, boolean pRecoverable, java.lang.String pExId)
- NamedExtensionException(int pErrorCode)
- NamedExtensionException(int pErrorCode, boolean pRecoverable, java.lang.String pExId)
- NamedExtensionException(int pErrorCode, java.lang.Throwable pEx)
- NamedExtensionException(int pErrorCode, java.lang.Throwable pEx, boolean pRecoverable, java.lang.String pExId)
- NamedExtensionException(java.lang.String msg)
- NamedExtensionException(java.lang.String msg, boolean pRecoverable, java.lang.String pExId)
- NamedExtensionException(java.lang.String msg, java.lang.Exception e)
- NamedExtensionException(java.lang.String msg, java.lang.Exception e, boolean pRecoverable, java.lang.String pExId)
- NamedExtensionException(java.lang.String msg, int pErrorCode)
- NamedExtensionException(java.lang.Throwable pEx)
- NamedExtensionException(java.lang.Throwable pEx, boolean pRecoverable, java.lang.String pExId)

**Methods**

| Method | Description |
|---|---|
| getExceptionID | public java.lang.String getExceptionID()<br>Gets the exception ID. |
| isRecoverable | public boolean isRecoverable()<br>Indicates whether the exception is recoverable or not. |
| setExceptionID | public void setExceptionID(java.lang.String pExId)<br>Sets the exception ID. |
| setRecoverable | public void setRecoverable(boolean pRecover)<br>Specifies that the exception is recoverable. |

**Inherited Methods**

NamedExtensionException inherits the following methods from class com.cisco.aons.exeption.AONSException.

- create (two expressions)
- getArgs
- getErrorCode
- getErrorResourceKey (two expressions)
- getExceptionEnum
- getPrefix
- getReplyMessage (two expressions)
- getResourceBundle (five expressions)
- getResourceMessage (multiple expressions)

For descriptions of these methods, see AONSException, page A-3.

## InitializationException

Extending AONSException, this class is used to handle exceptions that occur during AON initialization. InitializationException inherits methods from java.lang and has a set of constructors. These components are summarized below.

**Inherited Methods**

InitializationException inherits the following methods from class com.cisco.aons.exeption.AONSException.

- create (two expressions)
- getArgs
- getErrorCode
- getErrorResourceKey (two expressions)
- getExceptionEnum
- getPrefix
- getReplyMessage (two expressions)

- getResourceBundle (five expressions)

  For descriptions of these methods, see AONSException, page A-3.

InitializationException inherits the following method from class.java.lang.Throwable:

- fillInStackTrace
- getCause
- getLocalizedMessage
- getMessage
- getStackTrace
- initCause
- printStackTrace
- printStackTrace
- printStackTrace
- setStackTrace
- toString

It also inherits the following methods from class.java.lang.Ojbect:

- clone
- equals
- finalize
- getClass
- hashCode
- notify
- notifyAll
- wait (three expressions)

**Constructors**

- public InitializationException(java.lang.Exception e)
- public InitializationException(java.lang.String msg)
- public InitializationException(java.lang.String msg, java.lang.Exception e)

### NoSuchVariable

This exception is used to handle calls to non-existent policy execution plan (PEP) variables. NoSuchVariable inherits methods from class com.cisco.aons.exception.AONSException and others java.lang. The methods and constructors are listed below.

**Inherited Methods**

- create (two expressions)
- getArgs
- getErrorCode
- getErrorResourceKey (two expressions)
- getExceptionEnum

- getPrefix
- getReplyMessage (two expressions)
- getResourceBundle (two expressions)
- getResourceMessage (nine expressions)

For descriptions of these methods, see AONSException, page A-3.

NoSuchVariable also inherits the following methods from class java.lang.Throwable:

- fillInStackTrace
- getCause
- getLocalizedMessage
- getMessage
- getStackTrace
- initCause
- printStackTrace (three expressions)
- setStackTrace
- toString

It also inherits the following methods from class java.lang.Object:

- clone
- equals
- finalize
- getClass
- hashCode
- notify
- notifyAll
- wait (three expressions)

**Constructors**

- public NoSuchVariable(int pErrorCode)
- public NoSuchVariable(int pErrorCode, ExceptionType pEnum)
- public NoSuchVariable(java.lang.String pMessage, int ErrorCode)
- public NoSuchVariable(java.lang.String pMessage, int pErrorCode, ExceptionType pEnum)
- public NoSuchVariable(int pErrorCode, java.lang.Object[] pArgs)
- public NoSuchVariable(int pErrorCode, ExceptionType pEnum, java.lang.Object[] pArgs)
- public NoSuchVariable(int pErrorCode, java.lang.Object[] pArgs, java.lang.Throwable pEx)
- public NoSuchVariable(int pErrorCode, ExceptionType pEnum, java.lang.Object[] pArgs, java.lang.Throwable pEx)
- public NoSuchVariable(java.lang.String pMessage, int pErrorCode, java.lang.Object[] pArgs, java.lang.Throwable pEx)
- public NoSuchVariable(java.lang.String pMessage, int pErrorCode, ExceptionType pEnum, java.lang.Object[] pArgs, java.lang.Throwable pEx)

# External Services Package

The External Services Package (com.cisco.aons.aonscommon.com.cisco.aons.extservice) contains the following interfaces:

- AONSTransformer
- AONSTransformerFactory
- Authentication
- CacheService
- Compression
- ContentLookup
- ContentValidation
- Encryption
- ExtService
- ExtServiceContext
- ExtServiceProfile
- MessageLog
- MIME
- NoSuchVariable
- ServiceFactory
- Signature (also known as “Digital Signature”)
- Transform

These packages are described separately in Chapter 4, “AON Optimization - Fastpath”.

# PEP Package

The AON PEP package (**com.cisco.aons.pep**) contains one interface class: PEPData. For more information, see Policy Execution Plans, page 2-43.

## PEPData

The PEPData interface generates all meta information available about the policy execution plan (PEP).It includes the methods and fields summarized below.

**Methods**

| Method | Description |
|---|---|
| getId() | public java.lang.String **getId()**<br>Returns the Id (string) associated with this PEP. |
| getName() | public java.lang.String getName()<br>Returns the PEP name as a string. |

| Method | Description |
|---|---|
| getTime() | public long getTime()<br>Returns the PEP time associated with the PEP time. |
| getVariable() | public java.lang.Object getVariable(java.lang.String pName)<br>throws NoSuchVariable<br>Parameters:<br>pName—Input name.<br>Returns the value associated with the Variable. |
| getMsgTypeName() | public java.lang.String getMsgTypeName()<br>Returns the message classifier type. |
| setVariable | public java.lang.Object setVariable(java.lang.String pName,<br>java.lang.Object pValue)<br>throws NoSuchVariable<br>Sets the Variable value. |
| setVariable | public java.lang.Object setVariable(java.lang.String pName,<br>java.lang.Object pValue,<br>boolean pCreate)<br>throws NoSuchVariable<br>Parameters:<br>pName—input name<br>pValue—input object value<br>pCreate—Creates the variable if it does not exist. Primarily used in List and Map if the variable at the index or key does not exist.<br>Sets the Variable value. |

**Fields**

| Field | Description |
|---|---|
| REQUEST_MESSAGE | public static final java.lang.String REQUEST_MESSAGE |
| RESPONSE_MESSAGE | public static final java.lang.String RESPONSE_MESSAGE |

# Log Package

The AONSCommon Log package (**aonscommon.src.com.cisco.aons.log**) includes one class for logging operations: Log.

For a general description of AON logging, see the AON Installation and Administration Guide.

## Log

The Log class is used with AON message logging. It includes the methods summarized below.

| Method | Description |
|---|---|
| debug | public void debug(java.lang.Object message) |
| debug | public void debug(java.lang.Object message, java.lang.Throwable t) |
| error | public void error(java.lang.Object message) |
| error | public void error(java.lang.Object message, java.lang.Throwable t) |
| fatal | public void fatal(java.lang.Object message) |
| fatal | public void fatal(java.lang.Object message, java.lang.Throwable |
| info | public void info(java.lang.Object message) |
| info | public void info(java.lang.Object message, java.lang.Throwable t) |
| isDebugEnabled() | public boolean isDebugEnabled() |
| isErrorEnabled() | public boolean isErrorEnabled() |
| isFatalEnabled() | public boolean isFatalEnabled() |
| isInfoEnabled() | public boolean isInfoEnabled() |
| isWarnEnabled() | public boolean isWarnEnabled() |
| warn | public void warn(java.lang.Object message) |
| warn | public void warn(java.lang.Object message, java.lang.Throwable t) |

# Message Package

The AON Common Message package (**aonscommon.src.com.cisco.aons.message**) includes the following elements:

- Interfaces, page A-14
- Classes, page A-38
- Exceptions, page A-40

## Interfaces

The message package includes the interfaces listed below.

- IAONSMessage, page A-15
- ICloseable, page A-16

These interfaces are summarized in the following sections.

### IAONSMessage

Extending ICloseable, this interface is a canonical container for representing messages. It includes the methods summarized below.

| Method | Description |
|---|---|
| getContent() | public IContent getContent()<br>Returns the message content in a canonical container, which can be any subtype of IContent.<br>For more information, see IMapContent, IMIMEContent, INullContent, ISOAPContent, IStreamContent, and IXMLContent. |
| getMessageBuilder() | public IMessageBuilder getMessageBuilder()<br>Returns IMessageBuilder, the message builder associated with this AON message.It can be used to create a new AON message. |
| getMessageContext() | public IMessageContextgetMessageContext()<br>Returns the context that contains message metadata. |
| getMessageContextID() | public java.lang.String getMessageContextId()<br>Returns the context ID for this message. |

| Method | Description |
|---|---|
| getMessageHeaders() | public IMessageHeadersgetMessageHeaders()<br>Returns the message headers of a message. |
| getMessageId() | public java.lang.String getMessageId()<br>Returns the message ID associated with this message. |
| getMessageTimestamp() | public long getMessageTimestamp()<br>Returns the long date and time when the message arrived in milliseconds. |
| getMessageType() | public int getMessageType()<br>Returns the message type (int). Various bits in the int define attributes including request/reply/ack, app/control message, sync/async message, and so forth. For more information, see IMessageConstants. |
| getSendingNode() | public URI getSendingNode()<br>Returns the AON cloud node that sent the message.<br>Returns the URI of the AON cloud node (entry or exit node) that sent this message. This node generated the MessageId. The URI will be in the form aonp://111.122.111.11:433. |
| getSessionId() | public java.lang.String getSessionId()<br>Returns the ID (string) for the session associated with this message. A session is defined by a connection. |
| getSSLCert() | public java.security.cert.Certificate getSSLCert()<br>Returns the SSL certificate. |
| isApplicationReply() | public boolean isApplicationReply()<br>Returns “true” (boolean) if the message is an application reply message, responding to a request message. |
| IsApplicationRequest() | public boolean isApplicationRequest()<br>Returns “true” (boolean) if the message is an application request message. A client application sends a request message via an AON entry node. |
| setSSLCert | public void setSSLCert(java.security.cert.Certificate cert)<br>Sets the ConnectionConext to obtain the SSL certificate. For additional information, see the description of the Identity bladelet in the AON ADS Bladelet Reference. |

## ICloseable

This interface is implemented by classes whose instances are to be tracked in PEP execution and closed at the end of the PEP.

All objects that are placed in the PEP context are garbage collected by the normal cleanup process. This interface does not have to be implemented for every type of object that can be placed in the context.

# Message Package

The AON Common Message package (**aonscommon.src.com.cisco.aons.message**) includes the following elements:

- Interfaces, page A-14
- Classes, page A-38
- Exceptions, page A-40

## Interfaces

The message package includes the interfaces listed below.

- IAONSMessage, page A-15
- ICloseable, page A-16
- IContent, page A-17t
- IContentAttachment, page A-19
- IContentVisitable, page A-19
- IContentVisitor, page A-20
- IContextSerializable, page A-21
- IDeliveryContext, page A-22
- IEncodingConstants, page A-22
- IMapContent, page A-22
- IMessageBuilder, page A-25r
- IMessageConstants, page A-29
- IMessageContext, page A-29
- IMessageDeliveryContext, page A-30
- IMessageHeaders, page A-31
- IMIMEContent, page A-34
- IMsgAttachment, page A-36
- INullContent, page A-36
- IRNContent, page A-36
- ISOAPContent, page A-37t
- IStreamContent, page A-37
- IXMLContent, page A-37

These interfaces are summarized in the following sections.

| Method | Description |
|---|---|
| getMatchingHeaders | public java.util.Iterator getMatchingHeaders(java.lang.String hdrName)<br>Parameters:<br>hdrName—Name of header to match<br>Returns an iterator over a collection of headers that match the given header. |
| getParent() | public IContent getParent()<br>Returns the parent of the current content or null. |
| IsInputStreamAvailable() | public boolean isInputStreamAvailable()<br>Returns true if the input stream is available, otherwise false. |
| removeHeader | public void removeHeader(java.lang.String name)<br>Parameters:<br>name—Name of the header to be removed<br>Removes headers with the given name. |
| setContentAttachment | public void setContentAttachment (IContentAttachment pAttachment)<br>Parameters:<br>pAttachment—IContentAttachment<br>Sets the attachment with the content. |
| setHeader | public void setHeader(java.lang.String name,<br>java.lang.String value)<br>Parameters:<br>name—Header name<br>value—Header value<br>Replaces the given header with a new value or adds a header. |

| Field | Description |
|---|---|
| MAP_CONTENT | public static final int MAP_CONTENT<br>Map content. |
| MIME_CONTENT | public static final int MIME_CONTENT<br>MIME content. |
| NULL_CONTENT | public static final int NULL_CONTENT<br>Null content. |
| SOADP_CONTENT | public static final int SOAP_CONTENT<br>SOAP content. |

| Field | Description |
|---|---|
| STREAM_CONTENT | public static final int STREAM_CONTENT<br>Stream content. |
| XML_CONTENT | public static final int XML_CONTENT<br>XML content. |

### IContentAttachment

This is content attachment interface. It the single method summarized below.

| Method | Description |
|---|---|
| close() | public void close()<br>Releases any resources held by the attachment. |

For more information, see "IContent" section on page A-17.

### IContentVisitable

This interface accepts a content visitor. It is implemented by each subtype of the IContent interface. A visitor can visit any subtype of IContent using the double-dispatching mechanism. It has the following subinterfaces: IContent, IMapContent, IMIMEContentt, INullContent, ISOAPContent, IStreamContent, and IXMLContent. IContentVisitable has a single method, summarized below.

| Method | Description |
|---|---|
| accept Visitor | public void accept Visitor(IContentVisitor pVisitor)<br>throws AONSException<br>Parameters:<br>pVisitor—IContentVisitor the content visitor<br>Accepts the content visitors. Throws AONSException if it does not process the call. |

### IContentVisitor

The IContentVisitor interface visits IContent subtypes. It is implemented by AbstractContentVisitor. IContentVisitor includes the methods summarized below.

| Method | Description |
|---|---|
| visitMapContent | public void visitMapContent(IMapContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IMapContent, the map content<br>Visits a map content. Throws AONSException if unable to visit a map content. |
| visit MIME Content | public void visitMIMEContent(IMIMEContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IMIMEContent the MIME content<br>Visits a MIME content. Throws AONSException if unable to visit a MIME content. |
| visitNullContent | public void visitNullContent(INullContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—INullContent, the null content<br>Visits a null content. Throws AONSException if unable to visit a null content. |
| visitSOAPContent | public void visitSOAPContent(ISOAPContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IMapContent, the SOAP content<br>Visits a SOAP content. Throws AONSException if unable to visit a SOAP content. |

| Method | Description |
|---|---|
| visitStreamContent | public void visitStreamContent(IStreamContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IStreamContent, the stream content<br>Visits a stream content. Throws AONSException if unable to visit the stream content. |
| visitXMLContent | public void visitXMLContent(IXMLContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IXMLContent, the XML content<br>Visits an XML content. Throws AONSException if unable to visit an XML content. |

## IContextSerializable

Extending java.io.Serializable, this is a generic interface for serialization of AON message context. IContextSerializable has the following subinterfaces: IDeliveryContext and IMessageContext. The IContextSerializable methods are summarized below.

| Method | Description |
|---|---|
| deserialize | public void serialize(java.io.OutputStream pOS)<br>throws java.io.IOException<br>Parameters:<br>pOS—OutputStream<br>Serializes the message context to the output stream |
| serialize | public void serialize(java.io.OutputStream pOS)<br>throws java.io.IOException<br>Parameters:<br>pIS—InputStream<br>Serializes the message context from the input stream |

### IDeliveryContext

The IDeliveryContext interface holds the contextual information of an AONS message It has one subinterface:IMessageContext. The IDeliveryContext has the methods summarized below.

| Method | Description |
|---|---|
| getEndpointMessageId | public java.lang.String getEndpointMessageId()<br>Returns the endpoint message ID (if one exists) for this message. |
| getMessageContextId | public java.lang.String getMessageContextId()<br>Returns the message ID context for this message. |
| getMessageId | public java.lang.String getMessageId()<br>Returns the message ID associated with this message. |

### IEncodingConstants

The IEncodingConstants interface is used to handle standard encoded constants where the content is chunked, content-encoded, transfer-encoded, or gzipped. It has the fields summarized below.

| Field | Description |
|---|---|
| CHUNKED | public static final int CHUNKED<br>Indicates that the content is chunked. |
| CONTENT_ENCODING | public static final int CONTENT_ENCODING<br>Indicates content encoding. |
| GZIP | public static final int GZIP<br>Indicates that the content is compressed. |
| TRANSFER_ENCODING | public static final int TRANSFER_ENCODING<br>Indicates transfer encoding. |

For more information, see IMessageConstants, IMapContent, IMIMEContent, INullContent, ISOAPContent, IStreamContent, and IXMLContent

### IMapContent

The IMapContent interface maps message content, representing data as name-value pairs. It inherits the following fields from IContent: MAP_CONTENT, MIME_CONTENT, NULL_CONTENT, SOAP_CONTENT, STREAM_CONTENT, and XML_CONTENT. It also inherits the following methods from IContent: addHeader, getAllHeaders, getContentType, getInputStream, getMatchingHeaders, getParent, isInputStreamAvailable, removeHeader, setContentAttachement, and setHeader. IMapContent also inherits one method form IContentVisitable: acceptVisitor.

IMapContent has the methods summarized below.

| Method | Description |
|---|---|
| getParameter | public java.lang.String getParameter(java.lang.String name)<br>Parameters:<br>name—String, the name.<br>Returns the value of a request parameter as a string, or null if the parameter does not exist. Request parameters are extra information sent with the request. For HTTP protocol, parameters are contained in the query string or posted form data.<br>You should only use this method when you are certain that the parameter has a single value. If the parameter may have more than one value, use getParameterValues(java.lang.String).<br>If you use this method with a multivalued parameter, the returned value is equal to the first value in the array returned by getParameterValues.<br>If the parameter data was sent in the request body, such as an HTTP POST request, using getInputStream() to directly read the body may interfere with the execution of this method. |
| getParameterMap | public java.util.Map getParameterMap()<br>Returns a java.util.Map of the parameters of this request. Request parameters are extra information sent with the request. For HTTP protocol, parameters are contained in the query string or posted form data. |
| getParameterNames | public java.util.Enumeration getParameterNames()<br>Returns an enumeration of string objects containing the names of parameters contained in this request. If the request has no parameters, this method returns an empty enumeration. |
| getParameterValues( | public java.lang.String[] getParameterValues(java.lang.String name)<br>Parameters:<br>name—String<br>Returns an array of strings containing all of the values that the given request parameter has, or null if the parameter does not exist. |

| Method | Description |
|---|---|
| setParameter | public void setParameter(java.lang.String name,<br>java.lang.String value)<br>Parameters:<br>name—String<br>value—String<br>Adds a single name-value pair to the map content. |
| setParameter | public void setParameter(java.lang.String name,<br>java.lang.String[] values)<br>Parameters:<br>name—String<br>values—String[]<br>Adds a single name multivalue pair to the map content. |

## IMessageBuilder

The IMessageBuilder interface is used to build AON messages. It includes the methods and fields summarized below.

| Method | Description |
|---|---|
| copyAONSMessage | public IAONSMessage copyAONSMessage(IAONSMessage pMessage)<br>throws MessageBuildException<br>Parameters:<br>pMessage—IAONSMessage<br>Clones the content of the original AON message and creates a new one. The content is referenced, not copied. |
| createAONSMessage | public IAONSMessage createAONSMessage(int mMsgType,<br>IMessageContext pContext,<br>IMessageHeaders pHeaders,<br>IContent pContent,<br>java.lang.String pMsgContextId)<br>Parameters:<br>mMsgType—int indicating the type of message. APP_REQUEST_MESSAGE—if it is a request message APP_REPLY_MESSAG—if it is a response message<br>pContext—IMessageContext<br>pHeaders—IMessageHeaders<br>pContent—IContent<br>pMsgContextId—String the message context id. Used for correlation.<br>Returns IAONSMessage, creating a new AON message using supplied values. For more information, see IMessageConstants. |
| createAONSMessage | public IAONSMessage createAONSMessage(int mMsgType,<br>IMessageContext pContext,<br>IMessageHeaders pHeaders,<br>IContent pContent)<br>Parameters:<br>mMsgType—int indicating the type of message.<br>APP_REQUEST_MESSAGE—if it is a request message APP_REPLY_MESSAG—if it is a response message<br>pContext—IMessageContext<br>pHeaders—IMessageHeaders<br>pContent - IContent<br>Returns IAONSMessage, creating a new AON message using the supplied values. For more information, see IMessageConstants. |

| Method | Description |
|---|---|
| createAONSMessage | public IAONSMessage createAONSMessage(int mMsgType,<br>IMessageContext pContext,<br>IMessageHeaders pHeaders,<br>java.lang.String pMsgContextId)<br>Parameters:<br>mMsgType—int indicating the type of message.<br>APP_REQUEST_MESSAGE—if it is a request message<br>APP_REPLY_MESSAG—if it is a response message<br>pContext—IMessageContext<br>pHeaders—IMessageHeaders<br>pMsgContextId—String<br>Returns IAONSMessage, creating a new AON message using the supplied values. For more information, see IMessageConstants. |
| createAONSMessage | public IAONSMessage createAONSMessage(int mMsgType,<br>IMessageContext pContext,<br>IMessageHeaders pHeaders)<br>Parameters:<br>mMsgType—int indicating the type of message.<br>APP_REQUEST_MESSAGE—if it is a request message<br>APP_REPLY_MESSAG—if it is a response message<br>pContext—IMessageContext<br>pHeaders—IMessageHeaders<br>Returns IAONSMessage, creating a new AON message using supplied values. For more information, see IMessageConstants. |
| createDeliveryContext | public IDeliveryContextcreateDeliveryContext()<br>throws MessageBuildException<br>Creates an empty AON message delivery context. |
| createErrorAONSMessage | public IAONSMessage createErrorAONSMessage(IAONSMessage pMessage,<br>AONSException pException)<br>throws MessageBuildException<br>Parameters:<br>pMessage—IAONSMesspException -<br>pException—AONSException<br>Returns IAONSMessage, creating an error message. The method copies appropriate context and header information from the input AON message. |

| Method | Description |
|---|---|
| createMapContent | public IMapContent createMapContent()<br>Returns IMapContent, creating map content. |
| createMapContent | publicIMapContent createMapContent(java.lang.String pContentType)<br>Parameters:<br>pContentType—String<br>Creates map content of a specified type. |
| createMessageContext | publicIMessageContext createMessageContext()<br>Returns IMessageContext, creating a new message context. |
| createMessageContext | public IMessageContext createMessageContext(int pRedirect)<br>Parameters:<br>pRedirect—int<br>Returns IMessageContext, creating a new message context with a redirect directive. |
| createMessageHeaders | public IMessageHeaders createMessageHeaders()<br>Returns IMessageHeaders, creating a new message header. |
| createMIMEContent | publicIMIMEContent createMIMEContent()<br>Returns IMIMEContent, creating MIME content. |
| createMIMEContent | public IMIMEContent createMIMEContent(java.lang.String pContentType)<br>Parameters:<br>pContentType—String<br>Returns IMIMEContent, creating MIME content of a specified type. |
| createNullContent | public INullContent createNullContent()<br>Returns INullContent, creating null content. |
| createReplyAONSMessage | public IAONSMessage createReplyAONSMessage(IAONSMessage pMessage,<br>IContent pContent)<br>throws MessageBuildException<br>Parameters:<br>pMessage—original message content<br>pContent—new content<br>Returns IAONSMessage, creating an AON message from supplied content. It copies appropriate content and header information. |

| Method | Description |
|---|---|
| createReplyAONSMessage | public IAONSMessage createReplyAONSMessage(IAONSMessage pMessage,<br>IMessageHeaders pHeaders,<br>IContent pContent)<br>throws MessageBuildException<br>Parameters:<br>pMessage—original message content<br>pHeaders—IMessageHeaders, message headers of the returned AON message<br>pContent—new content<br>Returns IAONSMessage, creating a new reply AON message using contextual information from the input AON message. Throws MessageBuildException if unable to construct a new AON message. |
| createSOAPContent | public ISOAPContent createSOAPContent()<br>Returns ISOAPContent, creating SOAP content. |
| createSOAPContent | public ISOAPContent createSOAPContent(java.lang.String pContentType)<br>Parameters:<br>pContentType—String<br>Returns ISOPContent, creating SOAP content of a specified type. |
| createStreamContent | public IStreamContent createStreamContent()<br>Returns IStreamContent, creating stream content. |
| createStreamContent | public IStreamContent createStreamContent(java.lang.String pContentType)<br>Parameters:<br>pContentType—String<br>Returns IStreamContent, creating stream content of a specified type. |
| createXMLContent | public IXMLContent createXMLContent()<br>Returns IXML, creating XML content. |

| Method | Description |
|---|---|
| createXMLContent | public IXMLContent createXMLContent(java.lang.String pContentType)<br>Parameters:<br>pContentType—String<br>Returns IXMLContent, creating XML content of a specified type. |
| replaceContent | public void replaceContent(IAONSMessage pMessage,<br>IContent pContent)<br>throws MessageBuildException<br>Parameters:<br>pMessage—IAONSMessage<br>pContent—IContent<br>Replaces the original content with new content. |

| Field | Description |
|---|---|
| S_ALL | public static final int S_ALL |
| S_CONTENT | public static final int S_CONTENT |
| S_ CONTEXT | public static final int S_CONTEXT |
| S_HEADER | public static final int S_HEADER |
| S_REPLY_MESSAGE | public static final int S_REPLY_MESSAGE |
| S_REQUEST_MESSAGE | public static final int S_REQUEST_MESSAGE |

### IMessageConstants

This interface contains fields defining a variety of message constants (remote_host, session_id, HTTP_query_string, control message, error message, and so on).

### IMessageContext

This interface maintains the contextual information of a message. It includes the methods summarized below.

| Method | Description |
|---|---|
| getDestination() | Returns the message destination. |
| getDestinationHost() | Returns the destination host IP for the message. |
| getDestinationPort() | Returns the destination host port for the message. |
| getDestinationProtocol() | Returns the destination protocol. |
| getReason() | Returns the message reason. |
| getRedirect() | Returns the redirect indicator, or zero if it is not set. |
| getSessionId | Returns the session ID. |

| Method | Description |
|---|---|
| getSourceHost() | Returns the source host IP address for the message. |
| getSourcePort() | Returns the source host port for the message. |
| getSourceProtocol() | Returns the source protocol |
| getSourceURI() | Returns the source URI. |
| getStatus | Returns the status code. |
| getTransportSourceHost() | Returns the transport source IP for the message. |
| getTransportSourcePort() | Returns the transport source port for the message. |
| isProxyStyleRequest() | Returns “true” if is a proxy style request, otherwise “false.” |
| setDestination(URI pDesURL) | Sets the destination URI. |
| setDestinationProtocol(int pProtocol) | Sets the destination protocol. |
| setReason(java.lang.String pReason) | Sets the reason. |
| setSourceProtocol(int pProtocol) | Sets the source protocol. |
| setStatus(int pStatus) | Sets the status code. |

### IMessageDeliveryContext

This interface holds contextual information of an AONS message The methods are summarized below.

| Method | Description |
|---|---|
| clone | public java.lang.Object clone()<br>throws<br>java.lang.CloneNotSupportedException<br>All sub-types implement this method. |
| count | public int count()<br>Returns the count of the delivery context headers. |
| fromMap | public void fromMap(java.util.Map pMap)<br>Sets the delivery context using a map. |
| getContextType | public int getContextType()<br>Returns the context type. |
| initialize | public void initialize(java.lang.String pKey,<br>int pIndex)<br>Parameters:<br>pKey - String<br>pIndex - int<br>Initializes the context with type. |

| Method | Description |
|---|---|
| initialize | public void initialize(int pIndex)<br>Parameters:<br>pIndex - int<br>Initializes the context with name and type. |
| toMap | public java.util.Map toMap()<br>Returns a map representation of the delivery context. |

### IMessageHeaders

Extending Cloneable, this interface is a container for message headers. It includes the methods summarized below.

| Method | Description |
|---|---|
| add | public void add(java.lang.String name, java.lang.String value)<br>Parameters:<br>name - the name of the field<br>value - the value of the field.<br>Adds or sets a header field. If the field can have multiple values, adds multiple headers of the same name. |
| addDataField | public void addDateField(java.lang.String name,<br>java.util.Date date)<br>Parameters:<br>name - the field name<br>date - the field date value<br>Adds the value of a date field. |
| addDataField | public void addDateField(java.lang.String name,<br>long date)<br>Parameters:<br>name - the field name<br>date - the field date value<br>Adds the value of a date field. |
| clear | public void clear()<br>Removes all headers. |
| clone | <br>Clones a message header. |
| containsKey | public boolean containsKey(java.lang.String name)<br>Parameters:<br>name - the field name<br>Checks for the existence of a header field |

| Method | Description |
|---|---|
| ger | public java.lang.String get(java.lang.String name)<br>Parameters:<br>name - the case-insensitive field name<br>Returns the value of the header or null if not found. For multiple fields of the same name, only the first is returned. |
| getDateField | public long getDateField(java.lang.String name)<br>Parameters:<br>name - String<br>Returns the value of a date field, or "–1" if no date field is found. |
| getFieldNames | public java.util.Enumeration getFieldNames()<br>Gets an enumeration of header mNames.Returns an enumeration of strings representing the header mNames for this request. |
| getIntField | public int getIntField(java.lang.String name)<br>throws java.lang.NumberFormatException<br>Parameters:<br>name - String<br>Gets a header as an integer value.GReturns the value of an integer field or -1 if not found. The case of the field name is ignored. |
| getLongField | public long getLongField(java.lang.String name)<br>throws java.lang.NumberFormatException<br>Parameters:<br>name - String<br>Gets a header as a long value. Returns the value of an long field or -1 if not found. The field name case is ignored. |
| iterator | public java.util.Iterator iterator()<br>Returns an iterator for field name value pairs. |
| put | public void put(java.lang.String name,<br>java.util.List list)<br>Parameters:<br>name - the name of the field<br>list - the List value of the field. If null the field is cleared.<br>Sets a field. |
| put | public java.lang.String put(java.lang.String name,<br>java.lang.String value)<br>Parameters:<br>name - the name of the field<br>value - the value of the field. If null the field is cleared.<br>Sets a field. |

| Method | Description |
|---|---|
| putDateField | public void putDateField(java.lang.String name,<br>java.util.Date date)<br>Parameters:<br>name - the field name<br>date - the field date value<br>Sets the value of a date field. |
| putDateField | public void putDateField(java.lang.String name,<br>long date)<br>Parameters:<br>name - the field name<br>date - the field date value<br>Sets the value of a date field. |
| putIntField | public void putIntField(java.lang.String name,<br>int value)<br>Parameters:<br>name - the field name<br>value - the field integer value<br>Sets the value of an integer field. |
| putLongField | public void putLongField(java.lang.String name,<br>long value)<br>Parameters:<br>name - the field name<br>value - the field long value<br>Sets the value of a long field. |
| clone | public java.lang.Object **clone**()<br>throws java.lang.CloneNotSupportedException<br><br>Returns a clone object. |

| Method | Description |
|---|---|
| getValue | public java.util.Enumeration getValues(java.lang.String name,<br>java.lang.String separators)<br><br>Parameters:<br>name - the case-insensitive field name<br>separators - String of separators.<br>Returns an enumeration, multiple field values with separator.s The multiple values can be represented as separate headers of the same name or by a single header using the separator(s), or a combination of both. Separators may be quoted. |
| getValues | public java.util.Enumeration getValues(java.lang.String name)<br>Parameters:<br>name - the case-insensitive field name<br>Gets multiple field values. |

### IMIMEContent

Extended from IContent, this interface includes methods to handle MIME content. It includes the methods summarized below.

| Method | Description |
|---|---|
| addContent | public void addContent(IContent content)<br>Parameters:<br>content—content to add<br>Adds content to the end of a multipart MIME message. |
| addContentByIndex | public void addContentByIndex(IContent content,<br>int idx)<br>Parameters:<br>content—content to add<br>idx—index at which to add, other parts pushed back<br>Adds content at a specified position. |
| addEnvelop | public void addEnvelope(int type)<br>Parameters:<br>type—type constant, e.g. IMIMEContent.ENV_RN11<br>Add an envelope around the MIME content |

| Method | Description |
|---|---|
| getContentByHeader | public IContent getContentByHeader(java.lang.String[] names, java.lang.String[] values)<br>Parameters:<br>names—array of header names<br>values—corresponding array of values to match<br>Gets content that matches all given names and values. |
| getContentByHeader | public IContent getContentByHeader(java.lang.String name, java.lang.String value)<br>Parameters:<br>name—header name<br>value—header value<br>Gets content with the given name and value in the header or null. |
| getContentById | public IContent getContentByID(java.lang.String id)<br>Parameters:<br>id—content ID to select<br>Returns the IContent (part) corresponding to the ID. |
| getContentByIndex | public IContent getContentByIndex(int idx)<br>Parameters:<br>idx—part number to select<br>Returns the IContent corresponding to the part number. |
| getCount | public int getCount()<br>Returns the number of parts (count) of a multipart message. |
| getEnvType | public int getEnvType()<br>Returns envelope type (default IMIMEContent.ENV_NONE) |
| getIndex | public int getIndex(IContent content)<br>Parameters:<br>content—content to match<br>Returns the index of a given content using the Content-ID as the key or -1 if not found |
| removeContent | public void removeContent(int idx)<br>Parameters:<br>idx—index of part to remove<br>Removes content with the same Content-ID as the specified content. |

| Method | Description |
|---|---|
| removeContent | public void removeContent (IContent content)<br>Parameters:<br>content—content with the Content-ID to remove<br>Removes content at a specified index. The parts behind the point are pulled forward. |
| stripEnvelope | public void stripEnvelope()<br>Strips envelope from MIME content. Any information in the envelope is lost |

### IMsgAttachment

This interface is used to handle message attachments. It includes the methods summarized below.

| Method | Description |
|---|---|
| close | public void close()<br>Closes the message attachment. |
| getContext | public java.lang.Object getContext()<br>Gets the attachment context. |

### INullContent

Extended from IContent and IContentVisitable, this interface represents null content. It inherits the following methods from IContent. addHeader, getAllHeaders, getContentAttachment, getContentType, getInputStream, getMatchingHeaders, getParent, isInputStreamAvailable, removeHeader, setContentAttachment, and setHeader. It also inherits the acceptVisitor from IContentVisitable.

### IRNContent

This interface is used to get the signature associated with an envelope and get and set the version. It includes the methods summarized below.

| Method | Description |
|---|---|
| getSignature | public java.io.InputStream getSignature()<br>Returns the signature currently present in the envelope. |
| getVersion | public int getVersion()<br>Returns the version ID in the envelope header. Where, the upper 16 bits = major and lower 16 bits = minor. |
| setVersion | public void setVersion(int version)<br>Sets the version ID in the envelope header. Where, the upper 16 bits = major and lower 16 bits = minor. |

### ISOAPContent

Extended from IXMLContent, this interface represents SOAP XML content. It includes the method summarized below.

| Method | Description |
|---|---|
| getSOAPMessage() | public SOAPMessage getSOAPMessage()<br>throws MessageParseException<br>Returns SOAP messages. |

### IStreamContent

Extended from IContent, this interface represents stream content. It has the method summarized below.

| Method | Description |
|---|---|
| getOutputStream() | public java.io.OutputStream getOutputStream()<br>throws java.io.IOException<br>Returns an output stream to write content. |

### IXMLContent

Extended from IContent, this interface creates a container representing an XML document. It includes the methods summarized below.

| Method | Description |
|---|---|
| copyMsgBuffer | public void copyMsgBuffer(IRawStreamBuffer fromBuffer,<br>java.nio.ByteBuffer toBuffer)<br>throws AONSException<br>Copies a message buffer. |
| getAsDocument | public org.w3c.dom.Document getAsDocument()<br>throws MessageParseException<br>Returns a W3C document. |
| getMsgAttachment | public IMsgAttachment getMsgAttachment(java.lang.String name)<br>Returns an AON message attachment. |
| getRawStreamBuffer | public IRawStreamBuffer getRawStreamBuffer()<br>throws AONSException<br>Gets a raw stream buffer. |

| Method | Description |
|---|---|
| queryDocument | public org.w3c.dom.NodeList queryDocument(java.lang.String pXpath)<br>throws MessageParseException<br>Parameters:<br>pXpath—String<br>Evaluates the XPath expression on the document and returns a node list. |
| removeMsgAttachment | public void **removeMsgAttachment**(java.lang.String name)<br>Removes an AON message attachment. |
| setMsgAttachment | public void **setMsgAttachment**(java.lang.String name,<br>IMsgAttachment attachment)<br>Sets an AON message attachment. |
| writeDocument | public void writeDocument(org.w3c.dom.Document pDocument)<br>throws MessageParseException<br>Parameters:<br>pDocument—document<br>Saves the XML document into this container. |

## Classes

The AONSCommon message package includes the classes listed below.

- AbstractContentVisitor, page A-39
- AbstractMessageDeliveryContext, page A-40
- MessageBuilderRegistry, page A-40

These classes are summarized in the following sections.

### AbstractContentVisitor

This class implements the content visitor, defined by IContentVisitor. It includes the methods summarized below.

| Method | Description |
|---|---|
| visitMapContent | public void visitMapContent(IMapContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IMapContent the map content<br>Visits a map content. |
| visitMIMEContent | public void visitMIMEContent(IMIMEContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IMIMEContent the MIME content<br>Visits MIME content. |
| visitNullContent | public void visitNullContent(INullContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—INullContent the null content<br>Visits null content. |
| visitSOAPContent | public void visitSOAPContent (ISOAPContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IMapContent the SOAP content<br>Visits SOAP content. |
| visitStreamContent | public void visitStreamContent(IStreamContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IStreamContent the stream content<br>Visits stream content. |
| visitXMLContent | public void visitXMLContent (IXMLContent pContent)<br>throws AONSException<br>Parameters:<br>pContent—IXMLContent the XML content<br>Visits XML content. |

### AbstractMessageDeliveryContext

This class is an abstract implementation of IMessageDeliveryContext It has the methods summarized below.

| Method | Description |
|---|---|
| clone | public java.lang.Object clone()<br>throws java.lang.CloneNotSupportedException<br>Clones this context. |
| count | public int count()<br>Returns number of name tokens in this context. |
| getContextHeader | public final java.lang.String getContextHeader()<br>Returns the context name. |
| getContextType | public final int getContextType()<br>Returns the context type. |
| initialize | public final void initialize(int pCtxType)<br>Parameters:<br>pCtxType—int<br>Initializes the message context with the context type. |
| initialize | public final void initialize(java.lang.String pHeader,<br>int pCtxType)<br>Parameters:<br>pHeader—String<br>pCtxType—int<br>Initializes the message context with context name and type. |

### MessageBuilderRegistry

This class is used to build a message registry. It includes the methods summarized below.

| Method | Description |
|---|---|
| getInstance | public static final MessageBuilderRegistry getInstance()<br>Returns the single instance of the message builder registry. |
| getMessageBuilder | public IMessageBuilder getMessageBuilder(java.lang.String pProtocol)<br>Returns the protocol specific message builder. |
| getMessageBuilder | public IMessageBuilder getMessageBuilder()<br>Returns the default message builder. |

## Exceptions

The Message package uses the exceptions listed below.

- MessageBuildException, page A-41

- MessageException, page A-41
- MessageParseException, page A-41
- MessageWriteException, page A-41

These exceptions are summarized in the following sections.

### MessageBuildException

This class is used to build message exceptions. It inherits the following methods from AONSException: create (two versions), getArgs, getErrorCode, getErrorResourceKey (two versions), getExceptionEnum, getPrefix, getReplyMessage, getResourceBundle (two versions), getResourcemessage (nine versions), and setReplyMessage.

MessageBuildException also inherits the following methods from java.lang.Throwable: fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, setStackTrace, and toString. And, it inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, and wait (three versions).

### MessageException

Extending AONSEception, the MessageException is the top level message exception class. It inherits the following methods from AONSException: create (two versions), getArgs, getErrorCode, getErrorResourceKey (two versions), getExceptionEnum, getPrefix, getReplyMessage, getResourceBundle (two versions), getResourcemessage (nine versions), and setReplyMessage.

MessageException also inherits the following methods from java.lang.Throwable: fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, setStackTrace, and toString. And, it inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, and wait (three versions).

### MessageParseException

Extended from AONSException, this class is used to parse message exceptions. It inherits the following methods from AONSException: create (two versions), getArgs, getErrorCode, getErrorResourceKey (two versions), getExceptionEnum, getPrefix, getReplyMessage, getResourceBundle (two versions), getResourcemessage (nine versions), and setReplyMessage.

MessageParseException also inherits the following methods from java.lang.Throwable: fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, setStackTrace, and toString. And, it inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, and wait (three versions).

### MessageWriteException

Extended from AONSException, this is the message write exception class. It inherits the following methods from AONSException: create (two versions), getArgs, getErrorCode, getErrorResourceKey (two versions), getExceptionEnum, getPrefix, getReplyMessage, getResourceBundle (two versions), getResourcemessage (nine versions), and setReplyMessage.

MessageWriteException also inherits the following methods from java.lang.Throwable: fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, setStackTrace, and toString. And, it inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, and wait (three versions).

# Net Package

The AON Common net package (**com.cisco.aons.net**) includes the elements discussed in the following sections:Classes, page A-38.

For more information about AON, see the AON Installation and Administration Guide.

## Classes

The net package includes the classes discussed below.

### DateCache

This class defines the Date Format Cache. It computes string representations of dates and caches the result so that subsequent requests within the same minute will be fast. It only handles format strings that contain either “ss” or “ss.SSS.” The time zone of the date may be included as an ID with the “zzz” format string or as an offset with the “ZZZ” format string. If consecutive calls are often different, this class may be a little slower than a normal DateFormat. DateCache includes the methods summarized below.

| Method | Description |
|---|---|
| format | public java.lang.String format(java.util.Date inDate)<br>Formats a date according to a stored formatter. |
| format | public java.lang.String format(long inDate)<br>Formats a date according to a stored formatter. |
| format | public void format(long inDate,<br>java.lang.StringBuffer buffer)<br>Parameters:<br>inDate - Date the format<br>buffer - StringBuffer<br>Formats to a string buffer. |
| getFormat | public java.text.SimpleDateFormat getFormat()<br>Gets the format. |
| getFormatString | public java.lang.String getFormatString()<br>Gets a format string. |
| getTimeZone | public java.util.TimeZone getTimeZone()<br>Returns the time zone. |
| setTimeZone | public void setTimeZone(java.util.TimeZone tz)<br>Parameters:<br>tz—TimeZone<br>Sets the time zone. |
| setTimeZoneID | public void setTimeZoneID(java.lang.String timeZoneId)<br>Sets the time zone ID. |

## LazyList

This List helper class is sued to avoid unnecessary list creation. If a method needs to create a list to return, but it will be empty or contain a single item, this class is used to avoid additional object creations based on Collections.EMPTY_LIST or Collections.singletonList. LazyList is used the following way:

```
LazyList lazylist =null; while(loopCondition) { Object item = getItem(); if
(item.isToBeAdded()) lazylist = LazyList.add(lazylist,item); } return
LazyList.getList(lazylist);
```

The initial LazyList is a default sized ArrayList. This class has the methods summarized below.

| Method | Description |
|---|---|
| add | public static LazyList add(LazyList list,<br>java.lang.Object item)<br>Parameters:<br>list—The list to add to or null if none yet created.<br>item—The item to add.<br>Adds an item to a LazyList. |
| add | public static LazyList add(LazyList list,<br>java.util.Collection collection)<br>Parameters:<br>list—The list to add to or null if none yet created.<br>Adds an item to a LazyList. |
| add | public static LazyList add(LazyList list,<br>int initialSize,<br>java.lang.Object item)<br>Parameters:<br>list—The list to add to or null if none yet created.<br>initialSize—A size to use when creating the real list<br>item—The item to add.<br>Adds an item to a LazyList. |
| clone | public static LazyList clone(LazyList list) |
| clone | public java.lang.Object clone() |
| get | public static java.lang.Object get(LazyList list,<br>int i)<br>Parameters:<br>list—A LazyList returned from LazyList.add(Object) or null<br>Returns an item from the list. |
| get | public java.lang.Object get(int i) |

| Method | Description |
|---|---|
| getList | public static java.util.List getList(LazyList list)<br>Parameters:<br>list—A LazyList returned from LazyList.add(Object).<br>Returns real List of added items, which may be an EMPTY_LIST or a SingletonList. |
| getList | public static java.util.List getList (LazyList list,<br>boolean nullForEmpty)<br>Parameters:<br>list—A LazyList returned from LazyList.add(Object) or null<br>nullForEmpty—If true, null is returned instead of an empty list.<br>Returns real List of added items, which may be an EMPTY_LIST or a SingletonList. |
| iterator | public java.util.Iterator iterator() |
| listIterator | public java.util.ListIterator listIterator() |
| listiterator | public java.util.ListIterator listIterator(int i) |
| remove | public static LazyList remove (LazyList list,<br>java.lang.Object o)<br>Remove a LazyList. |
| size | public static int size(LazyList list)<br>Parameters:<br>list—A LazyList returned from LazyList.add(Object) or null<br>Returns the size of a LazyList. |
| size | public int size()<br>Returns the size of a LazyList. |
| toString | public java.lang.String toString() |
| toString | public static java.lang.String toString (LazyListlist) |
| toStringArray | public static java.lang.String[] toStringArray(LazyList list) |

## MultiMap

This class defines a multi-valued Map. The Map specializes the HashMap and provides methods that operate on multi valued items. It is implemented as a map of LazyList values MultiMap has the methods summarized below.

| Method | Description |
|---|---|
| add | public void add(java.lang.Object name,<br>java.lang.Object value)<br>Parameters:<br>name—entry key.<br>value—entry value.<br>Adds value to a multi valued entry. If the entry is single valued, it is converted to the first value of a multi valued entry. |
| addValues | public void addValues(java.lang.Object name,<br>java.util.List values)<br>Parameters:<br>name—entry key.<br>Adds values to a multi valued entry. If the entry is single valued, it is converted to the first value of a multi valued entry. |
| addValues | public void addValues(java.lang.Object name,<br>java.lang.String[] values)<br>Parameters:<br>name—entry key.<br>Adds values to a multi valued entry. If the entry is single valued, it is converted to the first value of a multi valued entry. |
| clone | public java.lang.Object clone() |
| get | public java.lang.Object get(java.lang.Object name) |
| getString | public java.lang.String getString(java.lang.Object name)<br>Parameters:<br>name—entry key.<br>Get a value as a string. Single valued items are converted to a String with the toString() Object method. Multi valued entries are converted to a comma separated List. The methods does not quote commas within values. |

| Method | Description |
|---|---|
| getValue | public java.lang.Object getValue(java.lang.Object name,<br>int i)<br>Parameters:<br>name—entry key.<br>i—index of the element to get.<br>Returns an unmodifiable list of values. If the value is not a multivalue, then index 0 retrieves the value or null. |
| getValues | public java.util.List getValues(java.lang.Object name)<br>Parameters:<br>name—entry key.<br>Gets multiple values. Single valued entries are converted to singleton lists. |
| put | public java.lang.Object put(java.lang.Object name,<br>java.lang.Object value)<br>Parameters:<br>name—entry key.<br>value—entry value.<br>Returns the previous value or a null. Puts an entry into the map. |
| putAll | public void putAll(java.util.Map m)<br>Parameters:<br>m—Map<br>Puts all contents of the map. |
| putValues | public java.lang.Object putValues(java.lang.Object name,<br>java.util.List values)<br>Parameters:<br>name—entry key.<br>Puts a multi valued entry. |
| putValues | public java.lang.Object putValues(java.lang.Object name,<br>java.lang.String[] values)<br>Parameters:<br>name—entry key.<br>Puts a multi valued entry. |

| Method | Description |
|---|---|
| removeValue | public boolean removeValue(java.lang.Object name,<br>java.lang.Object value)<br>Parameters:<br>name—entry key.<br>value—entry value.<br>Returns “true” if the value is removed. Remove a values. |
| toStringArrayMap | public java.util.Map toStringArrayMap()<br>Returns a map of string arrays. |

### StringMap

Extending java.util.AbstractMap, this is a map like class of strings. This string map is optimized for mapping small sets of strings where the most frequently accessed strings are put to the map first. In addition, it can look up entries by substring or sections of char and byte arrays. This can prevent many string objects from being created solely for map lookup purposes. This map is not synchronized. StingMap has the methods summarized below.

| Method | Description |
|---|---|
| clear | public void clear() |
| containsKey | public boolean containsKey(java.lang.Object key) |
| entrySet | public java.util.Set entrySet() |
| get | public java.lang.Object get(java.lang.Object key) |
| get | public java.lang.Object get(java.lang.String key) |
| getEntry | public java.util.Map.Entry getEntry(byte[] key,<br>int offset,<br>int length)<br>Parameters:<br>key—byte array containing the key. A simple ASCII byte to char mapping is used. offset—offset of the key within the array.<br>length—length of the key<br>Returns the Map.Entry for the key or null if the key is not in the map. |
| getEntry | public java.util.Map.Entry getEntry(java.nio.CharBuffer key,<br>int length)<br>Parameters:<br>key—char array containing the key<br>length—length of the key<br>Gets a map entry by char array key. |

| Method | Description |
|---|---|
| getEntry | public java.util.Map.Entry getEntry(java.lang.CharSequence key,<br>int pStart,<br>int pEnd)<br>Parameters:<br>key—char array containing the key<br>Gets a map entry by char array key. |
| getEntry | public java.util.Map.Entry getEntry(java.lang.CharSequence key)<br>Parameters:<br>key—char array containing the key<br>Gets a map entry by char array key. |
| getEntry | public java.util.Map.Entry getEntry(char[] key,<br>int offset,<br>int length)<br>Parameters:<br>key—char array containing the key<br>offset—offset of the key within the String.<br>length—length of the key<br>Returns the Map.Entry for the key or null if the key is not in the map. |
| getEntry | public java.util.Map.Entry getEntry(java.lang.String key,<br>int offset,<br>int length)<br>Parameters:<br>key—String containing the key<br>offset—Offset of the key within the String.<br>length—length of the key<br>Returns the map entry for the key or null if the key is not in the map. |
| getWidth | public int getWidth() |
| isEmpty | public boolean isEmpty() |
| isIgnoreCase | public boolean isIgnoreCase() |
| put | public java.lang.Object put(java.lang.Object key,<br>java.lang.Object value) |
| put | public java.lang.Object put(java.lang.String key,<br>java.lang.Object value) |
| readExternal | public void readExternal(java.io.ObjectInput in)<br>throws java.io.IOException,<br>java.lang.ClassNotFoundException |

| Method | Description |
|---|---|
| remove | public java.lang.Object remove(java.lang.Object key) |
| remove | public java.lang.Object remove(java.lang.String key) |
| setIgnoreCase | public void setIgnoreCase(boolean ic)<br>Parameters:<br>ic—If true, the map is case insensitive for keys.<br>Sets the ignoreCase attribute. |
| setWidth | public void setWidth(int width)<br>Parameters:<br>width—Width of hash tables, larger values are faster but use more memory.<br>Sets the hash width. |
| size | public int size() |
| writeExternal | public void writeExternal(java.io.ObjectOutput out)<br>throws java.io.IOException |

### StringUtil

Extending java.lang.Object, this is the class of fast string utilities. These string utilities provide both convenience methods and performance improvements over most standard library versions. The main aim of the optimizations is to avoid object creation unless absolutely required. StringUtil has the methods summarized below.

| Method | Description |
|---|---|
| append | public static void append(java.lang.StringBuffer buf,<br>java.lang.String s,<br>int offset,<br>int length)<br>Parameters:<br>buf - StringBuffer to append to<br>s—string to append from<br>offset—offset of the substring<br>length—length of the substring<br>Appends substring to StringBuffer |
| append | public static void append(java.lang.StringBuffer buf,<br>byte b,<br>int base) |
| asciiToLowerCase | public static java.lang.String asciiToLowerCase(java.lang.String s)<br>Parameters:<br>s—string to convert<br>Performs a fast lower case conversion.Only works on ascii, not on unicode |

| Method | Description |
|---|---|
| endsWithIgnoreCase | public static boolean endsWithIgnoreCase(java.lang.String s,<br>java.lang.String w) |
| equals | public static boolean equals(java.lang.String s,<br>char[] buf,<br>int offset,<br>int length) |
| indexFrom | public static int indexFrom(java.lang.String s,<br>java.lang.String chars)<br>Returns the next index of a character from the chars string. |
| nonNull | public static java.lang.String nonNull(java.lang.String s)<br>Parameters:<br>s—string<br>Returns a non null string. |
| replace | public static java.lang.String replace(java.lang.String s,<br>java.lang.String sub,<br>java.lang.String with)<br>Replaces substrings within string. |
| startWithIgnoreCase | public static boolean startsWithIgnoreCase(java.lang.String s,<br>java.lang.String w) |
| unquote | public static java.lang.String unquote(java.lang.String s)<br>Removes single or double quotes. |

## TypeUtil

Defines type utilities. Provides various static utility methods for manipulating types and their string representations. TypeUtil has the methods summarized below.

| Method | Description |
|---|---|
| convertHexDigit | public static byte convertHexDigit(byte b)<br>Parameters:<br>b—ascii encoded character 0-9 a-f A-F<br>Returns the byte value of the character 0-16. |
| fromHexString | public static byte[] fromHexString(java.lang.String s) |
| fromName | public static java.lang.Class fromName(java.lang.String name)<br>Parameters:<br>name—class or type name.<br>Returns the class from a canonical name for a type. |

| Method | Description |
|---|---|
| newInteger | public static java.lang.Integer newInteger(int i)<br>Converts an int to an integer using cache. |
| parseBytes | public static byte[] parseBytes(java.lang.String s,<br>int base) |
| parseInt | public static int parseInt(java.lang.CharSequence s,<br>int start,<br>int end,<br>int base)<br>throws java.lang.NumberFormatException |
| parseInt | public static int parseInt(java.lang.CharSequence s,<br>int base)<br>throws java.lang.NumberFormatException<br>Parameters:<br>Parses an int from a substring. Negative numbers are not handled. |
| toHexString | public static java.lang.String toHexString(byte[] b) |
| toHextSring | public static java.lang.String toHexString(byte[] b,<br>int offset,<br>int length) |
| toName | public static java.lang.String toName(java.lang.Class type)<br>Parameters:<br>type—class, which may be a primitive TYPE field.<br>Returns the canonical name for a type. |
| toSring | public static java.lang.String toString(int i)<br>Converts an int to a string using a cache. |
| toString | public static java.lang.String toString(byte[] bytes,<br>int base) |

| Method | Description |
|---|---|
| valueOf | public static java.lang.Object valueOf(java.lang.Class type, java.lang.String value)<br>Parameters:<br>type—class of the instance, which may be a primitive TYPE field.<br>value—value as a string.<br>Converts a string value to an instance. |
| valueOf | public static java.lang.Object valueOf(java.lang.String type, java.lang.String value)<br>Parameters:<br>type—classname or type (for example, int)<br>value—value as a string.<br>Converts a string value to an instance. |

## URI

This is the URI holder class. It includes methods that assist with encoding and decoding or HTTP URIs. It assists with query string formatting. It includes the methods summarized below.

| Method | Description |
|---|---|
| addPaths | public static java.lang.String addPaths(java.lang.String p1, java.lang.String p2)<br>Parameters:<br>p1—URI path segment<br>p2—URI path segment<br>Adds two URI path segments. |
| canonicalPath | public static java.lang.String canonicalPath(java.lang.String path)<br>Converts a path to a canonical form. All instances of "//", "." and ".." are factored out. Null is returned if the path tries to. above it's root. |
| clearParameters | public void clearParameters()<br>Clears the URI mParameters. |
| clone | public java.lang.Object clone()<br>Returns a URI clone. |
| decodePath | public static java.lang.String decodePath(java.lang.String path)<br>Parameters:<br>path—path the encode<br>Decodes a URI path. |

| Method | Description |
|---|---|
| emcodePath | public static java.lang.String encodePath(java.lang.String path)<br>Parameters:<br>path—path the encode<br>Encodes a URI path. This is the same encoding offered by URLEncoder, except that the '/' character is not encoded. |
| encodePath | public static java.lang.StringBuffer encodePath(java.lang.StringBuffer buf,<br>java.lang.String path)<br>Parameters:<br>path—path the encode<br>buf—StringBuffer to encode path into (or null)<br>Returns the StringBuffer or null if no substitutions required. Encodes a URI path. |
| encodeString | public static java.lang.StringBuffer encodeString(java.lang.StringBuffer buf,<br>java.lang.String path,<br>java.lang.String encode)<br>Parameters:<br>path—path the encode<br>buf—StringBuffer to encode path into (or null)<br>encode—string of characters to encode.<br>Returns the StringBuffer or null if no substitutions required. Encodes a URI path. |
| equals | public boolean equals(java.lang.Object pObj) |
| get | public java.lang.String get(java.lang.String name)<br>Gets the named value |
| getEncodedPath | public java.lang.String getEncodedPath()<br>Gets the encoded URI path. |
| getHost | public java.lang.String getHost()<br>Gets the URI host. |
| getParameterNames | public java.util.Set getParameterNames()<br>Gets URI query mParameters names. |
| getParameters | public MultiMap getParameters()<br>Gets URI query mParameters. |
| getPath | public java.lang.String getPath()<br>Gets the URI path. |
| getPort | public int getPort()<br>Gets the URI port. |

| Method | Description |
|---|---|
| getQuery | public java.lang.String getQuery()<br>Gets the URI query string. |
| getScheme | public java.lang.String getScheme()<br>Gets the URI scheme. |
| getUnmodifiableParameters | public java.util.Map getUnmodifiableParameters()<br>Gets the URI query mParameters. |
| getValues | public java.util.List getValues(java.lang.String name)<br>Parameters:<br>name—parameter name<br>Returns named multiple values. |
| hashCode | public int hashCode() |
| hasScheme | public static boolean hasScheme(java.lang.String uri)<br>Parameters:<br>uri—URI<br>Returns true if the URI has a scheme. |
| isAbsolute | public boolean isAbsolute()<br>Checks to determine whether the URI is absolute. |
| parentPath | public static java.lang.String parentPath(java.lang.String p)<br>Returns the parent path. Treats a URI like a directory path and return the parent directory. |
| put | public void put(java.lang.String encoded)<br>Parameters:<br>encoded—HTTP encoded string of mParameters: e.g. “a=1&b=2”<br>Adds a dictionary to the URI query mParameters. |
| put | public java.lang.Object put(java.lang.Object name,<br>java.lang.Object value)<br>Parameters:<br>name—name of value<br>value—value, which may be a multi valued list or String array.<br>Adds a name value pair to the URI query mParameters. |
| put | public void put(java.util.Map values)<br>Adds encoded mParameters. |
| putParametersTo | public void putParametersTo(MultiMap map)<br>Adds the URI query mParameters to a MultiMap. |
| remove | public void remove(java.lang.String name)<br>Removes a named value. |

| Method | Description |
|---|---|
| stripPath | public static java.lang.String stripPath(java.lang.String path)<br>Strips parameters from a path. |
| toString | public java.lang.String toString()<br>Returns path up to any semicolon parameters. |
| toURL | public java.net.URL toURL()<br>throws java.net.MalformedURLException<br>Returns the URL. |

### UriEncoded

This class handles coding of MIME "x-www-form-urlencoded". UniEncoded handles the encoding and decoding for either the query string of a URL or the content of a POST HTTP request. It has the methods summarized below.

| Method | Description |
|---|---|
| clone | public java.lang.Object clone()<br>This method overrides clone in class MultiMap |
| decode | public void decode(java.lang.String query) |
| decode | public void decode(java.lang.String query,<br>java.lang.String charset) |
| decodeString | public static java.lang.String decodeString(java.lang.String encoded)<br>Returns a decoded string with % encoding. |
| decodeString | public static java.lang.String decodeString(java.lang.String encoded,<br>java.lang.String charset)<br>Returns a decoded string with% encoding. |
| decodeString | public static java.lang.String decodeString(java.lang.String encoded,<br>int offset,<br>int length,<br>java.lang.String charset)<br>Returns decoded parameters to map. |
| decodeTo | public static void decodeTo(java.lang.String content,<br>MultiMap map)<br>Parameters:<br>content—string containing the encoded parameters<br>Returns decoded parameters to map. |

| Method | Description |
|---|---|
| decodeTo | public static void decodeTo(java.lang.String content,<br>MultiMap map,<br>java.lang.String charset)<br>Parameters:<br>data—byte[] containing the encoded parameters |
| decodeTo | public static void decodeTo(byte[] data,<br>MultiMap map,<br>java.lang.String charset)<br>Parameters:<br>data—ByteBuffer[] containing the encoded parameters<br>Returns decoded parameters to a map. |
| encode | public java.lang.String encode()<br>Encodes hashtable with % encoding. |
| encode | public java.lang.String encode(java.lang.String charset)<br>Encodes hashtable with % encoding. |
| encode | public java.lang.String encode(java.lang.String charset,<br>boolean equalsForNullValue)<br>Parameters:<br>equalsForNullValue—if True, then an '=' is always used, even for parameters without a value.For example, "blah?a=&b=&c=".<br>Encodes hashtable with % encoding. |
| encodeString | public static java.lang.String encodeString(java.lang.String string)<br>Performs URL encoding. |
| encodeString | public static java.lang.String encodeString(java.lang.String string,<br>java.lang.String charset)<br>Performs URL encoding. |

# Utilities Package

The AON Common utilities package (**com.cisco.aons.util**) includes the interfaces summarized below.



## Interfaces

The Utilities package has one interface:

### DomainReader

This interface is used by clients to get domain information. It includes the methods summarized below.

| Method | Description |
|---|---|
| getAttrNames() | Returns all the attribute names defined in the set. |
| getDomains() | Returns all domains having the given profile. |
| getSet(java.lang.String setName) | Returns all the information defined in the set. |
| getSetsForDomain() | Returns all sets defined in the domain. |
| getValues(java.lang.String setName, java.lang.String attrName) | Returns all attribute values for a given attribute. |
| setDomain(java.lang.String domainName) | Sets the domain for access. |
| setProfile(java.lang.String pType, java.lang.String pName) | Sets the profile for the DomainReader. |

## Classes

The Utilities package has one class:

### DomainException

Extended from the Exception, this class defines exceptions used by the DomainReader. DomainException inherits the following methods from java.lang.Throwable: fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, setStackTrace, and toString. It also inherits the following methods from java.lang.Object: clone, equals, finalize, getClass, hashCode, notify, notifyAll, and wait (three versions).

# XPath Engine Package

This package includes interfaces for XPath engine processing. Together, they define the processor and a buffer. These elements are described in the following section: Interfaces, page A-57.

For more information about AON, see the AON Installation and Administration Guide.

## Interfaces

The XPath engine packages are summarized below.

### IRawStreamBuffer

This interface defines a raw stream buffer. It includes the methods summarized below.

| Method | Description |
|---|---|
| close() | Closes the buffer. |
| getByteBuffer() | Returns the raw stream byte buffers |
| getStreamSize() | Returns the raw stream length. |

### IXPathProcessor

This interface defines an XPath processor. It includes the methods summarized below.

| Method | Description |
|---|---|
| checkCount | public boolean checkCount(IXMLContent xpathResult,<br>int groupID) |
| checkIndexInfo | public MetaData checkIndexInfo(java.lang.String flowName,<br>long flowTime) |
| compileXPathQueries | public boolean compileXPathQueries(java.lang.String[] exps,<br>int grpId) |
| classifyXPathExp | public boolean classifyXPathExp(java.lang.String exp,<br>int groupId) |
| compileXPathQueries | public boolean compileXPathQueries(java.util.ArrayList exps,<br>int grpId) |
| createGrpID | public int createGrpID(java.lang.String flowName,<br>java.lang.Object flowDef) |
| getGrpID | public int getGrpID(java.lang.String flowName) |
| lookup | public IFindResult lookup(IXMLContent content,<br>int groupID,<br>int xpathExpressionID)<br>throws XPathEngineException |
| lookup | public java.util.HashMap lookup(IXMLContent content,<br>int groupID,<br>java.util.HashMap Values) |
| process | boolean process(IXMLContent input) |


# AON Data Types

AON-specific and Java-based data types are used repeatedly in AON operations. This appendix introduces the file that contains data type definitions and provides summary descriptions of AON-specific data types. These elements are described in the following sections:

- DataTypes File, page B-1
- Data Types, page B-1

For more information about AON and the AON Management Console, see the AON Installation and Administration Guide. For a description of the AON Development Studio, see the AON Development Studio Guide.

## DataTypes File

The AON DataType.xml file contains definitions for all data types used in AON including AON-specific and Java-standard data types. Data Types, page B-1 provides descriptions for each AON-specific data type. This appendix does not describe the Java-standard data types that are used in AON operations. For descriptions of these data types (byte, short, int, float, long, double, string, object, boolean, iterator, and map) see the publicly-available current set of Java documentation.

## Data Types

This section describes the following AON-specific data types and associated attributes:

- AONSubject, page B-2
- AONSubjectListIterator, page B-2
- Content, page B-3
- ContentListIterator, page B-3
- FindContentListIterator, page B-3
- FindResult, page B-4
- FindContentListIterator, page B-3
- FindResultMapIterator, page B-4
- FindResultMapListIterator, page B-5
- Message, page B-5

# AONSubject

AONSubject (class com.cisco.aons.security.identity.AONSubjectSurrogate), a security input flow variable type, contains identity information about an AON message including username, password, and certificate. AONSubject attributes are summarized below.

| Attribute name | Type | Description |
|---|---|---|
| authenticated | boolean | Indicates (yes/no) whether or not an AON subject is authenticated. Read-only access. |
| authorized | boolean | Indicates (yes/no) whether or not an AON subject is authorized. Read-only access. |
| verified | boolean | Indicates (yes/no) whether or not an AON subject is verified. Read-only access. |
| trusted | boolean | Indicates (yes/no) whether or not an AON subject is trusted. Read-only access. |
| subjectID | string | String containing the subject ID. Read-only access. |
| LDAPGroups | list | List of LDAP groups. Read-only access. |

**Note** Most AON data type attributes are automatically generated (for example, by bladelet action) without any user input through ADS or AMC windows. These attributes are designated by "Read-only access" in the summary tables of this section.

# AONSubjectListIterator

AONSubjectListIterator (class com.cisco.aons.security.identity.AONSubjectListIterator), a security input flow variable, is an iterator for a list of AONSubject objects. The AONSubjectListIterator attributes are summarized below.

| Attribute name | Type | Description |
|---|---|---|
| first | AONSubject | First item in a subject list. Read-only access. |
| last | AONSubject | Last item in a subject list. Read-only access. |
| element At | AONSubject | Element At in a subject list. Read-only access. |

# Content

Associated with the IContext interface, the Content type (class com.cisco.aons.message.IContent) is represents the AON content. This type of object is created by the CreateContent bladelet and is used by the CreateMessage bladelet.

The Document type has an attribute that returns the message content. For example, REQUEST_MESSAGE.content() returns the content of the incoming message. Content attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| iterator | ContentListIterator | Iterator for the content list. Read-only access. |
| numParts | int | If the underlying content is a MIME content, this attribute returns the number of parts in the content. The number of parts of a MIME content in the incoming message is given by REQUEST_MESSAGE.content().numParts().<br>Read-only access. |

# ContentListIterator

The ContentListIterator (class com.cisco.aons.message.MEContentListIterator) contains a list of content values that can be accessed one at a time. For more information, see "Content" section on page B-3. ContenListIterator attributes are summarized below.

| Attribute name | Type | Description |
|---|---|---|
| first | Content | First value in the list. Read-only access. |
| last | Content | Last value in the list. Read-only access. |
| elementAt | Content | Element At in the content list. Takes an index (int) argument. Read-only access. |

# Document

Document (class org.w3c.dom.Document) represents a DOM Document. If the content is in XML, this type of object can be extracted from the Content object. You can not create an object of this type through direct input. For more information, see "Content" section on page B-3.

# FindContentListIterator

FindContentListIterator (class com.cisco.aons.xpathEngine.FindContentListIterator) is an iterator for content list search result. The list is the XPath / Regex result from Find bladelet evaluation. For more information, see "Content" section on page B-3 and the description of Find in the "AON Bladelets" section of the *AON Development Studio Guide*.

FindContentListIterator attributes are summarized below.

| Attribute name | Type | Description |
|---|---|---|
| first | FindResult | First item in the list. Read-only access. |
| last | FindResult | Last item in the list. Read-only access. |
| elementAt | FindResult | Element At in the list. Read-only access. |

# FindResult

Associated with the IFindResult interface, FindResult (class com.cisco.aons.xpathEngine.IFindResult) is a collection of search results for one xpath. It is generated by the Find bladelet as a XPath / Regex evaluation for each xpath / regex contained in the xpath/regex group.

FindResult attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| value | string | For single node results, this attribute returns the string value of the node. For a list of nodes, it returns the string value for the first node. Read-only access. |
| nodeValue | string | Returns the string value for the ith node. Read-only access. |
| size | int | Returns the size of the result set.<br>Read-only access. |

# FindResultMapIterator

FindResultMapIterator (class com.cisco.aons.xpathEngine.FindResultMapIterator) is an iterator for a map of xpath/regex search results. The key for the map is the name of the input xpath. The value of the map is the FindResult corresponding to the xpath. The Find bladelet creates this type of object.

FindResultMapIterator attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| first | FindContentListIterator | The first item. Read-only access. |
| last | FindContentListIterator | The last item. Read-only access. |
| elementAt | FindContentListIterator | Element At item. Read-only access. |

# FindResultMapListIterator

FindReulstMapListIterator (class com.cisco.aons.security.identity.SecContext) is an iterator for a map list of xpath/regex search results. This is a list of all the results of all XPath / Regex evaluations for all contents. FindResultMapListIterator attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| first | FindResultMapIterator | First item in the list. Read-only access. |
| last | FindResultMapIterator | Last item in the list. Read-only access. |
| elementAt | FindResultMapIterator | Element At item in the list. Read-only access. |

# Message

Associated with the IAONSMessage interface, the Message type (class com.cisco.aons.message.IAONSMessage) data type represents the AON message. The flow variable REQUEST_MESSAGE associated with this data type is available in the request-action and represents the incoming message. The flow variable REQUEST_MESSAGE is available in the response-action and represents the outgoing message. The CreateMessage bladelet can create an object of this type in the flow. Message attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| messageId | string | Returns the message ID.The ID of an incoming message is given by: REQUEST_MESSAGE.messageId(). Read-only access. |
| timeStamp | long | Time that the message was created. The timestamp of the incoming message is given by REQUEST_MESSAGE.timeStamp(). Read-only access. |
| srcIP | string | IP address of the message source. The source IP of the incoming message is given by REQUEST_MESSAGE.srcIP(). Read-only access. |
| srcPort | int | Port number of the message source. The source port of the incoming message is given by REQUEST_MESSAGE.srcPort(). Read-only access. |
| destIP | string | IP address of the message destination. The destination IP of the incoming message is given by REQUEST_MESSAGE.destIP(). Read-only access. |
| destPort | int | Pport number of the message destination. The destination port of the incoming message is given by REQUEST_MESSAGE.destPort(). Read-only access. |
| destProtocol | string | Message protocol at the destination. The protocol name of the incoming message is given by REQUEST_MESSAGE.destProtocol(). Read-only access. |

| Attribute name | Type | Description |
|---|---|---|
| header | object | Value of the message header. The User-Agent header of the incoming message is given by REQUEST_MESSAGE.header(User-Agent). Read-only access. |
| content | Content | AON content of the message. The content of the incoming message is given by REQUEST_MESSAGE.content(). Read-only access. |
| URI | string | Destination URI of the message. The URI of the incoming message is given by REQUEST_MESSAGE.URI(). Accessed via obtainUIR. |

# MessageTypeInfo

MessageTypeInfo produces the message type-related read-only attributes summarized below.

| Attribute name | Type | Description |
|---|---|---|
| name | string | Message type name |
| protocol | string | Message type protocol |
| uniPattern | string | URI pattern for the message type |

# PEPMetaData

PEPMetaData datatype produces the PEP-related read-only attributes summarized below.

For additional information, see Policy Execution Plans, page 2-43.

| Attribute name | Type | Description |
|---|---|---|
| interactionStyle | string | Values: REQUEST_RESPONSE or REQUEST_ONLY. |
| location | string | Values: REQUEST, RESPONSE, or LOCATION |
| name | string | PEP name |

# PlatformInfo

PlatformInfo datatype produces the platform-related read-only attributes summarized below.

| Attribute name | Type | Description |
|---|---|---|
| accelerator | boolean | Hardware accelerator |
| aonsVersion | string | AON version |
| internalDataIpAddr | string | Data IP address |
| internalMgmtIpAddr | string | Management IP address |

# SearchResult

SearchResult (class com.cisco.aons.message.MESearchResult), a core input flow variable, maps a search specifier to a list of content. In each case, the search specifier is determined by a previously specified search criteria. You use the search specifier to locate the corresponding result for a particular search criteria. SearchResult attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| elementAt | ContentListIterator | ElementAt in the list. Gets a string containing the key. Read-only access. |

# SearchResultListIterator

SearchResultListIterator (class com.cisco.aons.message.MESearchResultListIterator, a core input flow variable, is an iterator over a list of SearchResult objects. SearchResultListIterator attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| first | SearchResult | First search result. Read-only access. |
| last | SearchResult | Last search result. Read-only access. |
| elementAt | SearchResult | Element At in the search result. Read-only access. |

# SecurityContext

SecurityContext (class com.cisco.aons.security.identity.SecContext) stores subject and credential information for certain messages or content. SecurityContent attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| wssUsernameTokens | AONSubjectListIterator | List iterator for wssUsername token data. Read-only access. |
| wssUsernameTokensAuthenticated | AONSubjectListIterator | List iterator for wssUsername token authentication data. Read-only access. |
| wssUserNameTokensAuthorized | AONSubjectListIterator | List iterator for wssUsername token authorization data. Read-only access. |
| httpBasicAuths | AONSubjectListIterator | List iterator for basic HTTP authorization. Read-only access. |
| httpBasicAuthsAuthenticated | AONSubjectListIterator | List iterator for authenticated HTTP basic authorizations. Read-only access. |
| httpBasicAuthsAuthorized | AONSubjectListIterator | List iterator for authorized HTTP basic authentications. Read-only access. |

| Attribute name | Type | Description |
|---|---|---|
| wssSPNEGOTokens | AONSubjectListIterator | List iterator for wssSPNego tokens. Read-only access. |
| wssSPNEGOTokensAuthenticated | AONSubjectListIterator | List iterator for authenticated wssSPNego tokens. Read-only access. |
| httpNegAuths | AONSubjectListIterator | List iterator for authorized HTTP Negs. Read-only access. |
| httpNegAuthsAuthenticated | AONSubjectListIterator | List iterator for authenticated HTTP Neg authorizations. Read-only access. |
| wssX509CertTokens | AONSubjectListIterator | List iterator for verified qssX509 certification tokens. Read-only access. |
| wssX509CertTokensVerified | AONSubjectListIterator | List iterator for verified wssX509 certification tokens. Read-only access. |
| wssX509CertTokensTrusted | AONSubjectListIterator | List iterator for trusted wssX500 certification tokens. Read-only access. |
| wssX509CertPathTokens | AONSubjectListIterator | List iterator for wssX509 certification path tokens. Read-only access. |
| wssX509CertPathTokensVerified | AONSubjectListIterator | List iterator for verified wssX509 certification path tokens. Read-only access. |
| SAMLAssertions | AONSubjectListIterator | List iterator for SAML assertions. Read-only access. |
| SAMLAssertionsVerfieid | AONSubjectListIterator | List iterator for verified SAML assertions. Read-only access. |
| SSLPeerCerts | AONSubjectListIterator | List iterator for SSL peer certifications. Read-only access. |
| SSLPeerCertsVerfieid | AONSubjectListIterator | List iterator for verified SSL peer certifications. Read-only access. |
| SSLPeerCertsTrusted | AONSubjectListIterator | List iterator for trusted SSL peer certifications. Read-only access. |

## SecurityContextListIterator

SecurityContextListIterator (class com.cisco.security.identity.SecContextListIterator), a security input flow variable, is an iterator for a list of SecurityContext objects. SecurityContextListIterator attributes are summarized in the following table.

| Attribute name | Type | Description |
|---|---|---|
| first | SecurityContext | First item in the list. Read-only access. |
| last | SecurityContext | Last item in the list. Read-only access. |
| elementAt | SecurityContext | Element At in the list. Read-only access. |

## SystemInfo

SystemInfo datatype produces the system-related read-only attributes summarized below.

| Attribute name | Type | Description |
|---|---|---|
| upTime | long | AON uptime in milliseconds. |
| totalBufferSize | long | Total buffer size configured in AON. |
| percentInUse | double | Percentage of buffer that is in use. |

# AON Packaging

AON extensibility enables users to create custom bladelets, adapters, content parsers, transforms, and schema validations. ADS Packaging Wizard windows guide you through packaging process. Using the wizard, you create extensions that can be:

- Validated by AMC to ensure package integrity
- Understood by AON when a message is received.

**Caution** Although it is possible to create extensions manually (for example, developing a new jar file), you should use the packaging wizard. This approach ensures that your packages can be correctly uploaded into AMC or run on AON.

Creating a packages involves collecting all the data needed for that type of AON extension point into a single file. ADS validates the contents of each package and generates the appropriate manifest () that allows these packages to be loaded into AMC and deployed to the AON runtime engine. In particular, it is important to understand the definition of each package and their relationship to each other (if applicable):

- Bladelet Archive: This package contains the metadata for a bladelet that describes how it is to be configured in ADS. For more information, see Using the AON Bladelet Schema. The archive has a .bar extension and contains the following items:
  - Bladelet Info File: This is an XML file that completely describes a bladelet and all its parameters.
  - Icons: Image files used to represent the bladelet in the palette as well in the flow.

  - Validation classes: The validation class that the bladelet designer creates to ensure that the bladelet is completely and properly configured. ADS invokes the bladelet designer specified class each time the user enters a value for a parameter, providing real time feedback to the user about the state of the bladelet in the flow.
  - Validation rules: While the validation class is used to implement context sensitive and complex validation logic for a bladelet, some of the more simple validations (like data format, required versus optional etc.) are automatically performed by ADS using a declarative XML syntax. The validation rules for each parameter can be specified in this file and will be enforced by ADS when the bladelet is used in a flow.
  - Resource Bundle: The resource bundle contains the mapping between the resource keys used in the bladelet info file and their visual representation. These resource bundles provide complete i18n support for parameter names, error messages, and tool tips.

- Custom Bladelet Archive: This package represent the complete custom bladelet and consists of the bladelet archive (created using the bladelet archive packaging wizard step) along with the runtime components - Java/C code, libraries, supporting files, etc. This file which has a .scar extension is uploaded to the AMC and downloaded to each ADS when it synchronizes with that ADS. When an ADS synchronizes with the AMC, the bar file that is inside this package is extracted and sent to the ADS. The runtime code and any supporting files are not sent to the ADS since the bar file is sufficient to completely describe the bladelet. When the scar file is uploaded to the AMC, a global deployment request is created and the custom bladelet is deployed to all the nodes when the administrator processes these deployment requests. For more information, see Packaging Custom Bladelets.
- Adapter Package: This package contains the classes and metadata associated with an adapter or adapter extension. The metadata is specified in an adapter info XML file and the data can be shared libraries, Java classes, native code and supporting files. For more information, see Packaging Adapters.
- Content Parser and Transform Packages: A Content Parser plug-in extension is used to parse input data and convert it to an equivalent XML format on which AON XSLT Based Transformation can be applied. For more information, see Creating a Content Parser Extension.
- Schema Validation Package: The Content Validation bladelet is included in message PEPs to enable validation. AON can validate incoming XML messages to verify their conformity to a particular schema or DTD. Your message PEP can be designed to reject or drop any message that does not conform to the given schema. For more information, see Schema Validation.


## A



## B



## C

## D



## E

## F



## G



## H



## I

## J



## L



## M



## N

## O



## P



## R



## S



## T

## U



## V



## W



## X