Cisco AON Programming Guide

AON Release 1.1
October 2005

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

Introduction

This release of the Application-Oriented Network (AON) system includes sets of specialized application programming interfaces (APIs) and associated classes. This software is packaged as several sets of APIs and software development kits (SDKs). This guide describes the APIs and explains how to implement them.

This chapter introduces AON programmable features in the following sections:

Overall, 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, “External Services”
- Chapter 5, “Transformation”
- Chapter 6, “Schema Validation”
- Appendix A, “AONSCommon Specification”
- AON Data Types

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

AON Extensibility

Cisco provides an extensive set of built-in bladelets and adapters with AON. If they are not enough to meet your company’s operating requirements, you can extend AON. You may want to handle messages in a different way or be able to convert a certain customer’s message output to a more useful format. You can address these and many other unique message-handling challenges by creating new custom bladelets or adapters.

- Custom Bladelets

  You can use the Custom Bladelet SDK to develop new bladelets that extend the capabilities of AON. After they are incorporated into AON, the new bladelets function the same way as other bladelets. Generally, bladelets are functional software components that are variously grouped together as “policy execution plans” (PEPs) and imported into AON to provide combinations of message-processing services. AON also enables you to sandbox custom bladelets to protect AON

and custom bladelet-related security permissions. You can also enhance new bladelets so that an exception encountered during a PEP execution does not stop processing. For details, see Chapter 2, "Custom Bladelets".

- Custom Adapters

  An AON installation processes a variety of network traffic that may include custom protocols and proprietary message types. AON includes a set of built-in adapters for converting frequently-used message protocols to meet receiving-end requirements. Using the Custom Adapter SDK, you can create new adapters to handle custom protocols and message types.

  AON also enables you to meet the requirements of the IBM Webshere 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. For details, see Chapter 3, "Custom Adapters".

- 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 4, "External Services".

- Transformation

  XSLT Transformation (Transform and Content Parser) enables AON to transform a message or part of a message to fit the requirements at the sending end, receiving end, or both. This feature can be used to transform a XML message to HTML, non-XML to XML, and so forth. For details, see Chapter 5, "Transformation".

- Schema Validation

  AON can validate incoming XML messages to verify that they adhere to a specific schema or DTD. For details, see Chapter 6, "Schema Validation".

- AON Common Specification

  A large 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 AON Data Types.

AON Programming Requirements

This guide focuses on the issues and techniques that are used to implement the following AON programming requirements:

- Core API
  - Must have a well defined execution and exception handling model.
  - Must conform to bladelet SDK specifications (for example, extending AbstractCustomBladelet).
  - 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-ino.xml file (mentioned above). For example, see Using the AON Bladelet Schema, page 2-12 and Bladelet Info File, page 2-13.

Where to Go Next

For more information about AON, see the following documents:

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

Custom Bladelets

The Application-Oriented Network (AON) system includes a Custom Bladelet software development kit (SDK). You can use this set of application programming interfaces (APIs) and associated classes to create custom bladelets for AON. This feature is discussed in the sections listed below.

For additional information, see the *AON Administration and Installation Guide* and *AON Development Studio Guide*. See Chapter 4, "External Services" for more about extending custom bladelets.

Designing Custom Bladelets

AON bladelets are highly configurable. They can be executed in an environment that includes message-based policies, previously uploaded to the AON node. A custom bladelet designer should determine how a new bladelet will function and how it's operations may be affected at runtime by AON runtime-enforced policies.

Designers should take into account the installed policies of an AON node, possibly including message PEP policies. They should also decide which properties of the bladelet will be configurable; for example, the key size and algorithm of a bladelet that digitally signs documents.

Bladelets are incorporated into Policy Execution Plans (PEPs). Using the AON Development Studio (ADS), you create 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 Bladelet Reference section of the *AON Development Studio*.

Bladelet Development Summary

Briefly, to develop a custom bladelet, you must:

1. Extend bladelet classes and implement bladelet interfaces to achieve desired functionality
2. Implement the bladelet_info.xml file. This file defines the GUI appearance of the custom bladelet in the AON Developer Studio (ADS).

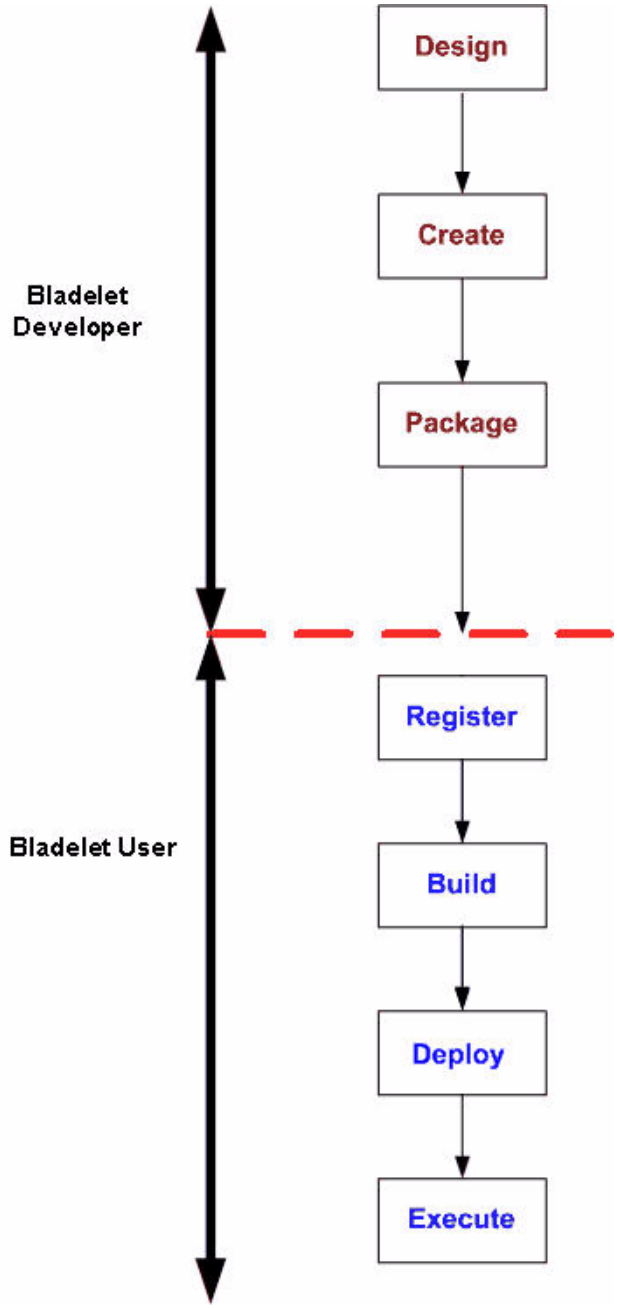
Custom bladelet developers extend bladelet classes and implement bladelet interfaces using the AON application programming interfaces (APIs) such as PolicyManager and message, to accomplish specific tasks within a policy execution plan (PEP).

For detailed steps, see Creating the Custom Bladelet, page 2-10.

Development Life Cycle

The development life cycle of a custom bladelet is represented in Figure 2-1. As this figure indicates, the overall development to execution life cycle of a custom bladelet. As this figure indicates, the developer creates and packages the new bladelet using the AON Design Studio (ADS). Later, the customer uses the AON Management Console (AMC) to upload the new bladelet and run it during AON operations.

Figure 2-1 AON Custom Bladelet Development Life Cycle



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

Bladelet Developer

- Design—The developer determines what functions the custom bladelet is to provide, how it is to be configured, what external libraries will be required, 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 "AbstractCustomBladelet" section on page 2-29.
- 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.

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 the "Designing the Custom Bladelet" section on page 2-4 and the "Creating the Custom Bladelet" section on page 2-10. For a complete description of all user activities, see the *AON Development Studio Guide* and *AON Management Console Guide*.

Designing the Custom Bladelet

When you develop a new custom bladelet, it must meet certain requirements and include key elements of the Custom Bladelet API. These considerations are discussed in the following sections:

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

Design Requirements

Your custom bladelet must conform to the general requirements listed in the following sections:

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—Custom bladelet should have a shorter, 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" section on page 2-12.
- Custom Bladelet API
  - Extend AbstractCustomBladelet

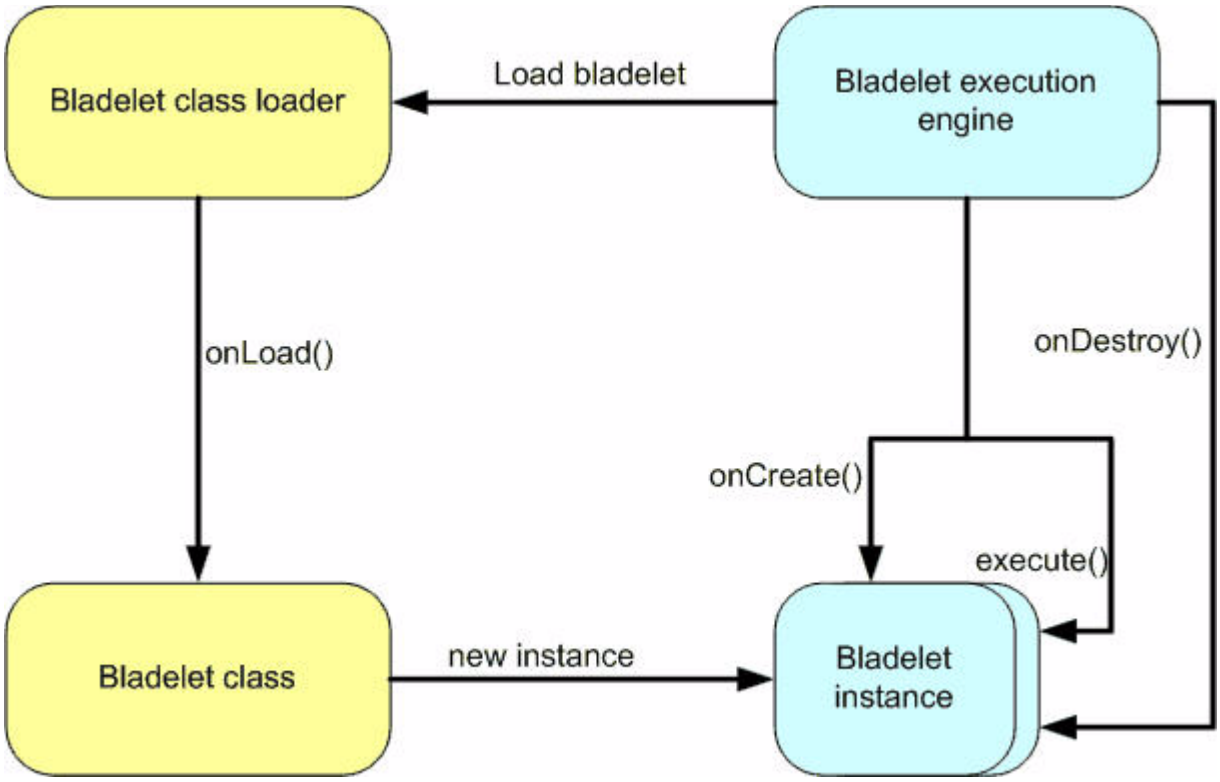
  or
  - Implement ICustomBladelet interface

For details, see the "Creating the Custom Bladelet" section on page 2-10.

Custom Bladelet Model

Figure 2-2 represents custom bladelet activity (state transitions).

Figure 2-2 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 PEP 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 Administration and Installation 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" section on page 2-25 and the"EmailBladelet" section on page 2-27. 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.

Sandbox

The AON sandbox feature enables customers to open up various permissions by default and restrict customizable permissions for custom bladelets. At the same time, sandboxing protects AON by restricting certain permissions which cannot be customized.

AON parses the internal AON policy file (cannot be user updated) which has default restrictions. At the custom bladelet package deployment time, AON parses the sandbox policy file which is defined in the package. Custom bladelet package level sandbox policy cannot override any restrictions imposed by the Global policy. At deployment time, AONS parses the policy file 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.

Java Based

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 Administration and Installation 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>
```

Custom Bladelet API Summary

The AON Custom Bladelet SDK includes Java interfaces and classes that can be used to build custom bladelets in Java. The kit also includes utility libraries for XML parsing, XPath, and other services. These components are identified below.

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

The Custom Bladelet SDK includes the following items:

- 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 PEP details, set output path and log messages.
- 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, remember to pass the message as a parameter and call 8-per and override the onCreate method.

For detailed descriptions of these components, see the "Custom Bladelet API Specification" section on page 2-28.

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 Java archive files (.jar) in the classpath.

Or change the ant script to include the SDK package in the classpath.

Note Ant is a scripting tool that is used to compile and run Java programs.

Creating the Custom Bladelet

After installing the SDK, you can create the custom bladelet. Follow the steps outlined below.

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" section on page 2-12, "Custom Bladelet Samples" section on page 2-25, and the "Custom Bladelet API Specification" section on page 2-28.

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 the *AON Administration and Installation Guide*.

Step 3 **Package the files**.

Using the AON Development Studio (ADS), package the custom bladelet files as a scar file. For general packaging steps, see "Packaging the Custom Adapter."

Note For brevity, these packaging steps are not repeated here.

Step 4 **Register the custom bladelet**

This action uploads the custom bladelet package to the AMC. For directions, see the registration step in "Packaging the Custom Adapter."

Step 5 **Deploy the custom bladelet.**

For directions see "Packaging the Custom Adapter." For additional information, see the *AON Development Studio Guide*.

Adding Exception Recovery to Custom Bladelets

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.

   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-2.

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

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" section on page 2-13.

- 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" section on page 2-19.

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 2-1 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.

Table 2-1 Bladelet Info File Attributes

Attribute	Description
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/AON/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 2-2 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.
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 2-3 on page 2-18, 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 2-3 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 2-3Bladelet Properties Screen, page 2-18.
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.
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 2-3 on page 2-18 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 2-4 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 2-5 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.
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 2-6 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.

Table 2-6 Parameter Attributes

Attribute	Value	Description
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.
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 2-7 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.

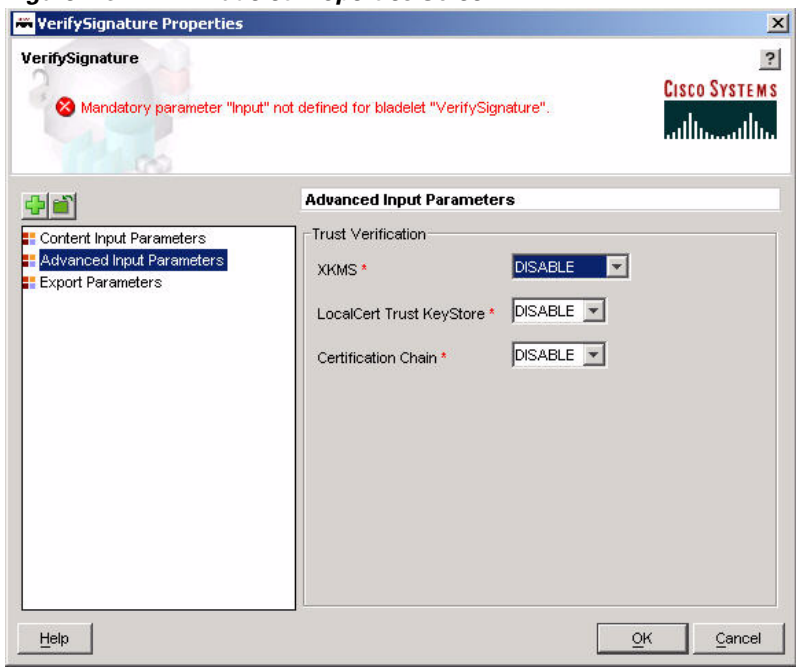
Column-Info Element

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 2-3 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 2-3 Bladelet Properties Screen



The following section of the bladelet-info.xml file (Verify bladelet) includes the configuration group and configuration subgroup sections that define items in Figure 2-3. For the entire code sample, see the "Verify Bladelet Info XML" section on page 2-40.

```
- <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 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>
```

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.

Custom Bladelet Samples

The following sections provide samples:

- RnetBladelet1, page 2-25
- EmailBladelet, page 2-27

Each sample conforms to the standard described in the "Using the AON Bladelet Schema" section on page 2-12. 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.IAONMessage;
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.AONException;
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 AONException {
      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 AONException(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 AONException(exception);
        }
        catch(NumberFormatException exception) {
            throw new AONException(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.IAONMessage;
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.AONException;
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 AONException {
      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 AONException(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 AONException(exception);
    }
    catch(MessagingException exception) {
        throw new AONException(exception);
    }
} // if sendemail is true
```

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 2-29**

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

- **CustomBladelet, page 2-33**

  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 2-35**

  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.

Code

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

import com.cisco.AON.message.IAONMessage;
import com.cisco.AON.log.Log;
import com.cisco.AON.exception.AONException;
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 IAONMessage msg;
    protected CustomBladeletContext context;
    protected DomainReader domainReader;
    public static Log logger;

    public void onLoad() {
    }

    public void onCreate(IAONMessage msg) {
        this.msg = msg;
    }

    /**
     * All the extending classes are expected to implement this method
     */
    public abstract void execute() throws AONException;

    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 Context for the bladelet.
domainReader	protected DomainReader domainReader Domain reader for the bladelet.
logger	public static Log logger Logger for the bladelet.
msg	protected IAONMessage msg 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 execute() throws AONException Specified by execute in interface CustomBladelet. All extending classes must implement this method. It performs the required business logic.
getContext	public CustomBladeletContext getContext() Specified by getContext in interface CustomBladelet. Gets the context from the custom bladelet.
setContext	public void setContext(CustomBladeletContext context) Parameters: context—CustomBladeletContext Specified by setContext in interface CustomBladelet. Sets the context for the custom bladelet.
onCreate	public void onCreate(IAONMessage msg) Specified by onCreate in interface CustomBladelet. Custom bladelets implement this method to perform creation activities.

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

Methods	Description
getLogger	public Log getLogger() Specified by getLogger in interface CustomBladelet. Returns the log object to record information and error messages.
setLogger	public void setLogger(Log logger) Specified by setLogger in CustomBladelet. 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

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

import com.cisco.AON.message.IAONMessage;
import com.cisco.AON.log.Log;
import com.cisco.AON.exception.AONException;
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(IAONMessage ctx);

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

    /**
     * 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() throws AONException Custom bladelet classes implement this method to perform business logic operations.
getContext	public CustomBladeletContext getContext() Returns the CustomBaldeletContext, getting context from the custom bladelet.

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

```
package com.cisco.AON.sdk.custombladelet;
import java.util.Map;
import com.cisco.AON.exception.AONException;
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 AONException
     */
    public void setGlobalContextData(String name,Object value) throws AONException;

    /**
     * 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 AONException
     */
    public void setContextData(String name,Object value) throws AONException;

    /**
     * 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() Returns the bladelet name.
getContextData	public java.lang.Object getContextData(java.lang.String name) Parameters: `name`—input parameter as String Returns the output object containing data associated with an input parameter of custom bladelet.
setContextData	public void setContextData(java.lang.String name, java.lang.Object value) throws AONException Parameters: `name`—output parameter as String Sets output parameters of custom bladelet.

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

Methods	Description
logMessage	logMessage public void logMessage(boolean bSync, java.lang.String messageLogPolicy, short logLevel, java.lang.String customId, java.util.Map expressions, java.util.Map variables) throws ExtServiceException Parameters: `bSync` (true)—Log records will be written to data source synchronously. (false)—Log records will be written to data source asynchronously. messageLogPolicy—Name of message log policy to use for logging. Message Log plogLevel 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. customId—Arbitrary string to identify this log entry. e.g. key to other 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. 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. Writes log records to a data source.
setOutputPath	setOutputPath public void setOutputPath(java.lang.String pOutputLabel) Parameters: pOutputLabel—label to be set Sets the output path (to determine the next step) for the bladelet.

Samples

The sample shown below is a typical bladelet-info.xml files.

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/AON/bladelet/v1/26i_verifysignature.png" />
<document-icon href="com/cisco/AON/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>
```

Custom Adapters

AON can process a variety of network traffic including industry standard and custom protocols. AON has a set of built-in adapters that may address all of your operating needs.

If you need additional adapters, you can create them with the AON Custom Adapter Software Development Kit (SDK). This chapter provides an adapter overview and explains how to use the SDK.

See Chapter 4, “External Services” 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.

Overview

Adapters, like bladelets, are configured to execute on a particular AON node. Adapter designers specify configuration parameters that are set by network administrators (typically in AMC), and read in by the adapter at runtime. This read in process enforces the policy determining how the adapter will connect 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. They provide the services listed below:

Inbound adapter performs:

1. TCP termination
2. protocol termination

3. encapsulates the application message into an AON-specific protocol header for processing by the AON system.

Outgoing adapter:

1. removes the AON-specific message from the incoming encapsulation
2. re-encapsulates the message into the outbound protocol.

This section provides background information about custom adapters and related AON processes.

AON Adapter Requirements

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 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 SDK Platform

The Custom Adapter SDK currently supports adapter development in Java.

Custom Adapter Interaction Models

The AON Custom Adapter SDK supports two interaction models:

- One-way message

  In this case, the message sender does not expect a 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 to receive a response from the message recipient.

Custom Adapter Integration Models

Depending on your requirements, you may develop an embedded or standalone adapter. These models are described below.

Embedded Adapter

When you want the adapter to use all AON resources, you should develop an embedded adapter. This type of adapter is tightly integrated with the AON framework. It is designed to open a socket and read a stream of data off it.

The adapter framework controls the message input and output through network connections, data buffers, and thread management. The Custom Adapter SDK provides a unique set of APIs for writing the embedded adapter. An embedded adapter is well-suited to handle stream-oriented protocols (HTTP, SMTP, FTP, and so on).

The embedded adapter must maintain its internal state. The framework drives the adapter's state transitions through a series of call backs to the adapter's receive and send handlers.

A callback programming model is used 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.

Standalone Adapter

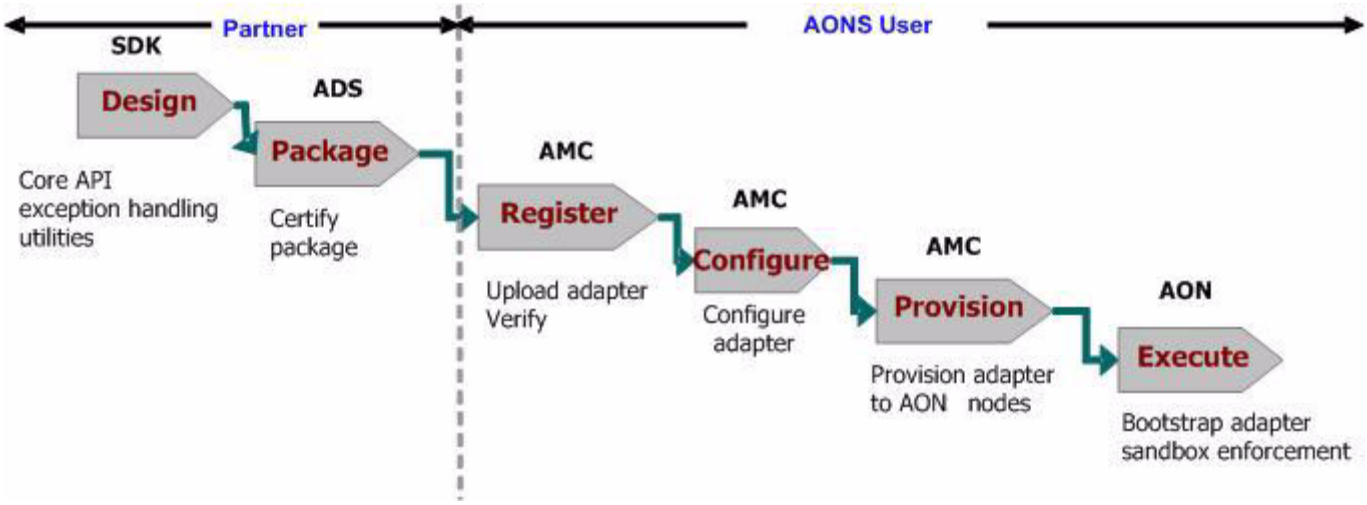
When you want to use the adapter decoupled from the rest of AON, you should develop a standalone adapter. A standalone adapter uses few or none of the resources provided by the adapter framework. The framework makes no assumptions about reading the message in or writing it out. When the standalone adapter 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.

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:

Custom Adapter Developer

- Develop—The developer writes the adapter using the Custom Adapter SDK APIs 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 3-43.

  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.

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 the Custom Adapter

This section explains how to develop the custom adapter in two sections:

- Developing an Embedded Adapter, page 3-6
- Developing Standalone Adapters, page 3-21

Tip Read the appropriate development procedure (for example, "Developing a Standalone Adapter") before you see the next section "Registering the Custom Adapter."

See also the *AON Management Console Guide*.

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 Management Console Guide*.

Using the AdapterListenerDomain

After registering your custom embedded or standalone adapter, you use the AMC to specify the communication parameters (a global policy) for the new adapter. For the steps, see the *AON Management Console Guide*.

Developing an Embedded Adapter

This section identifies the requirements and structural components of the embedded adapter.

Custom Adapter Names and Versions

The custom adapter must be uniquely identified by:

- 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.

Adapter Code Components

You write the custom 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.
- 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.

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

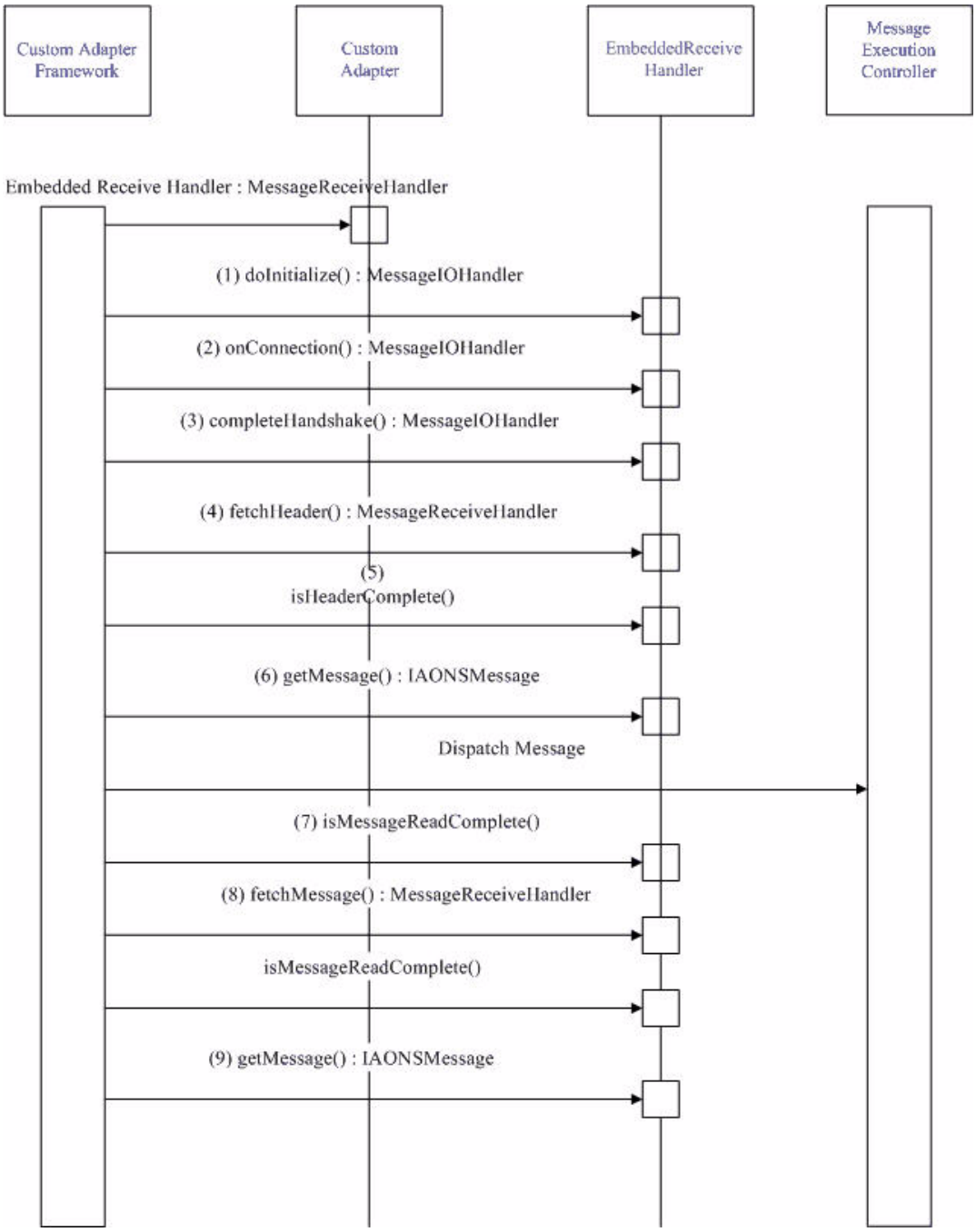
- IContentDecoder—Required for message decoding.
- IContentEncoder—Required for message encoding.

For detailed lists of the interface packages that are included in the custom adapter, see Custom Adapter API Specification, page 3-80.

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 dispatching to AON to completing the response.

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 IAONMessage 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.

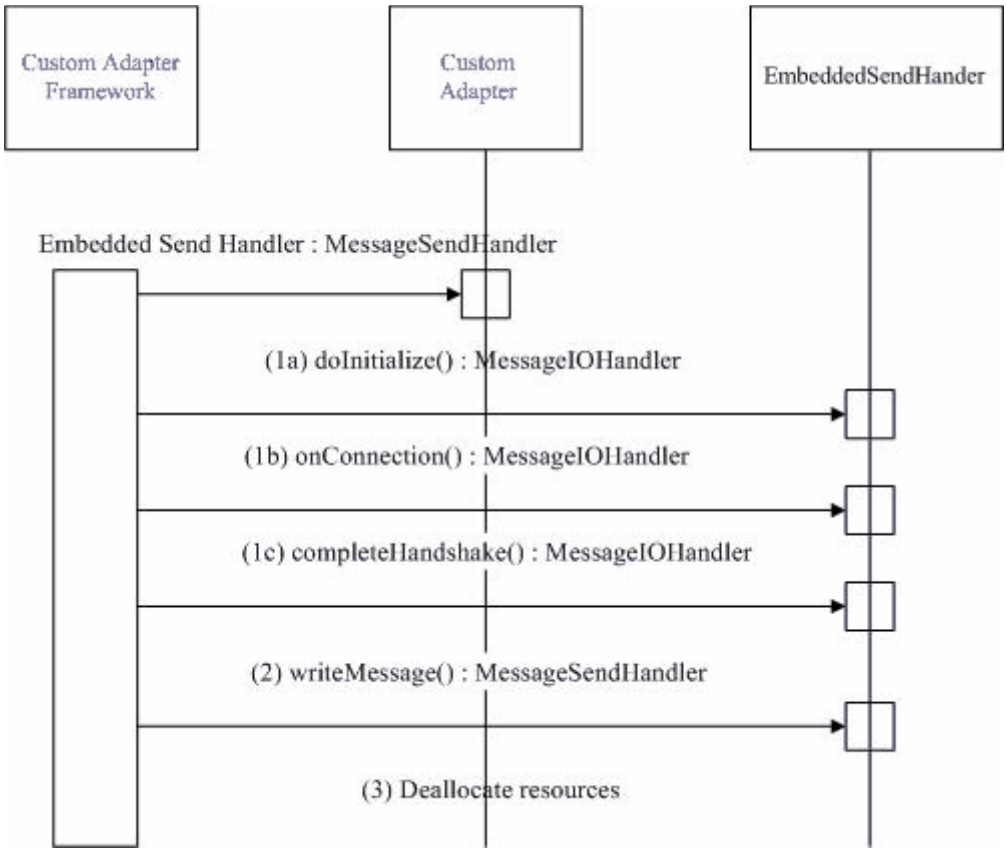
   If the dispatched message is a Request message, the framework classifies it according to the user-defined message type. The message type may have an associated Policy Execution Plan (PEP). AON executes the Request part of the PEP, if one is associated.

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 3-3below.

Figure 3-3 Embedded Adapter Send Handler



As this figure indicates, the Embedded Send Handler code executes in the following sequence: The adapter code uses the MessageSendHandler class to receive the AON message. This class extends MessageSendHandler (package com.cisco.aons.adapter).

1. Starting up, the code executes:
   - doInitialize()
   - onConnection()
   - completedHandshake()
2. 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.
3. Deallocates (closes) obtained resources.
   - 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

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.IAONMessage;
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.AONException;
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 IAONMessage 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 IAONMessage
     */
    public IAONMessage 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
                            AONException
                        {
                            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.IAONMessage;
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 IAONMessage 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 Standalone Adapters

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

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

Adapter Names and Versions

The adapter must be uniquely identified by:

- 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. As a minimum, your code must include these 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

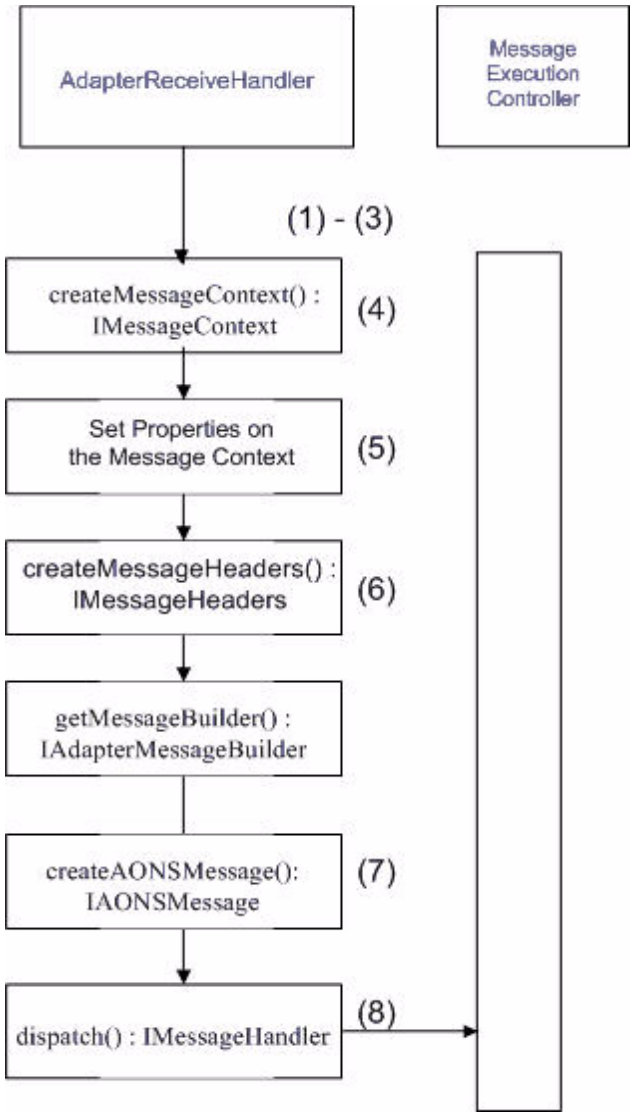
  Provide an encoder class that can encode an application message.

For an example, see the "Custom Adapter API Specification" section on page 3-80.

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 IAONMessage 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);
```

8. 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.

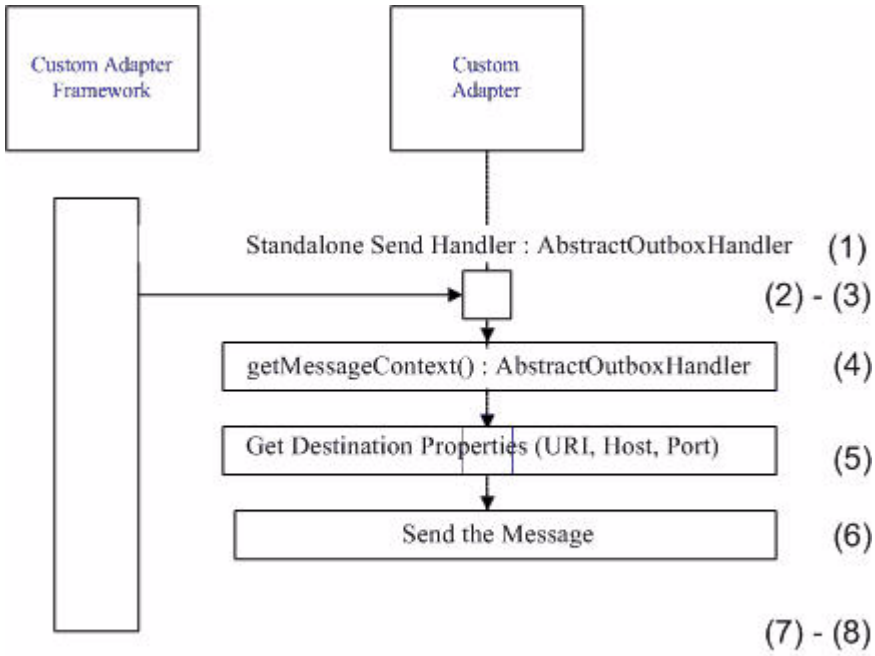
The following sample code sets attachments:

```
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

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.IAONMessage;
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.AONException;
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 IAONMessage 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 (AONException e)
    {
        mLogger.fatal(e.getMessage(), e);
    }
}

/**
 * readMessage
 *
 * @return IAONMessage
 */
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

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. This first step in the MQ Adapter development process is summarized below.

Note The same ADS and AMC tasks are performed to package and upload custom adapters, custom bladelets, content parsers, transformers, and schema validations. Although this an adapter procedure, you will use similar screens to package and upload other AON components.

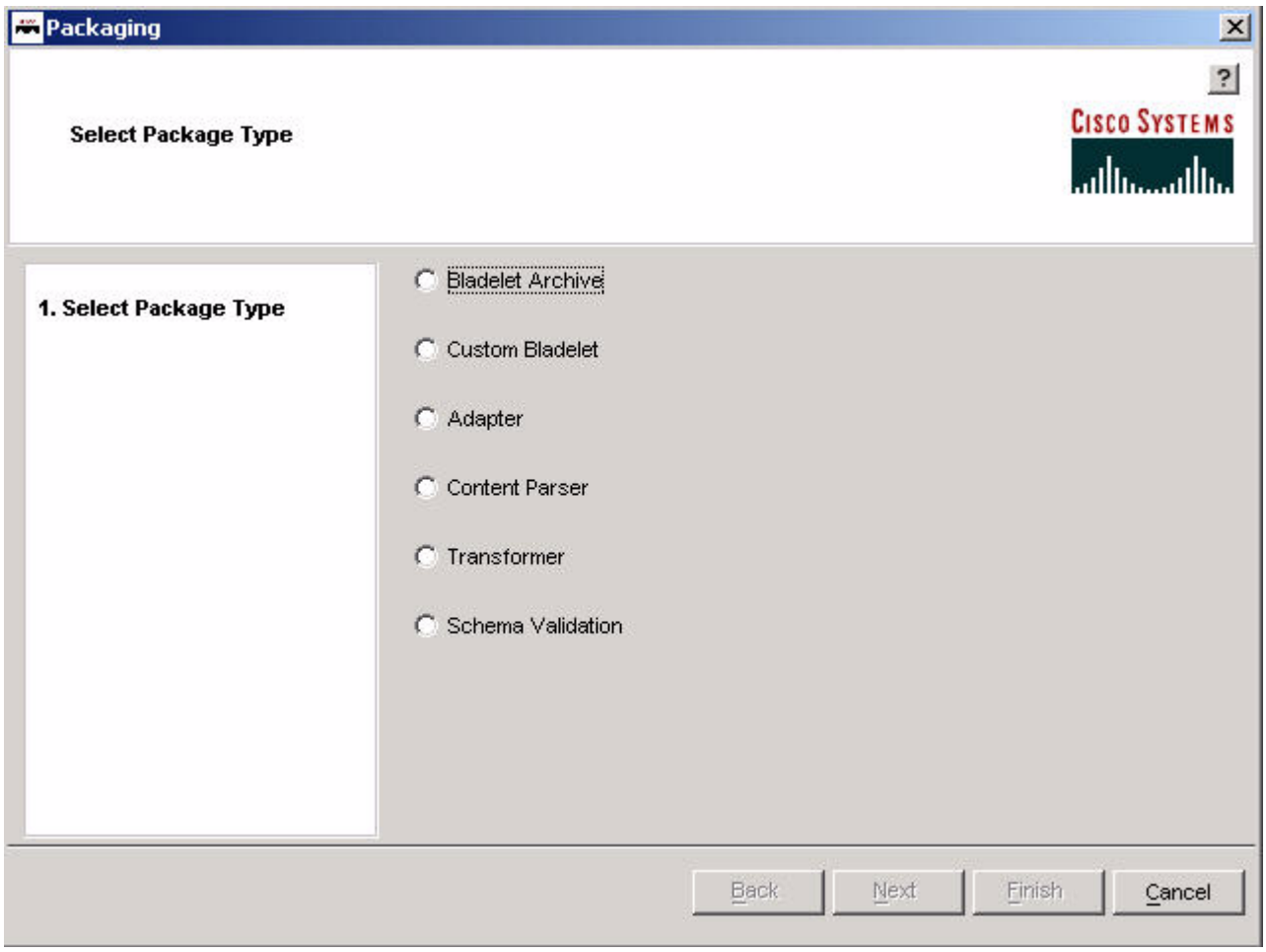
Step 1 Package the Custom Adapter

Using the ADS, package the adapter files.

a. Select Tools > Packaging.

The AON Packaging window (Figure 3-6) appears.

Figure 3-6 AON Packaging Window

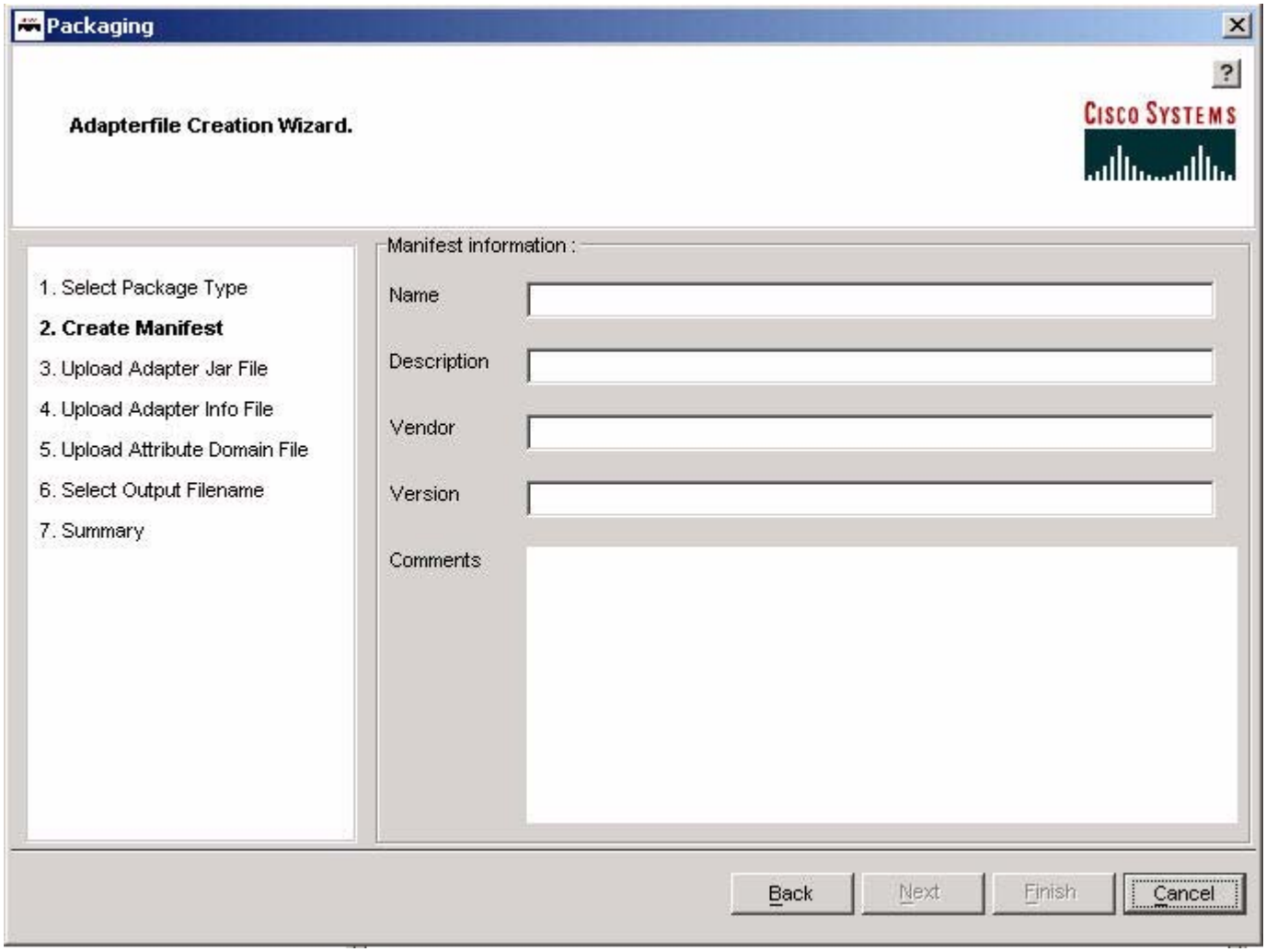


As the figure shows, this window is used for all AON packaging activities.

b. Select the package type (for example, Adapter) and click **Next**.

An AON file creation wizard (Figure 3-7) appears.

Figure 3-7 Adapter File Creation Wizard



An analogous wizard (with different fields) appears for each of the other package types. In each case, the current step is identified in the panel on the left.

c. Fill in the Manifest information.

The fields are described in Table 3-1.

Table 3-1 Adapter File Creation Manifest Information

Field	Description
Name	Filename; for example, an adapter file.
Description	(Optional) Description of the file.
Vendor	Vendor associated with the file; for example, Cisco Systems, Inc.

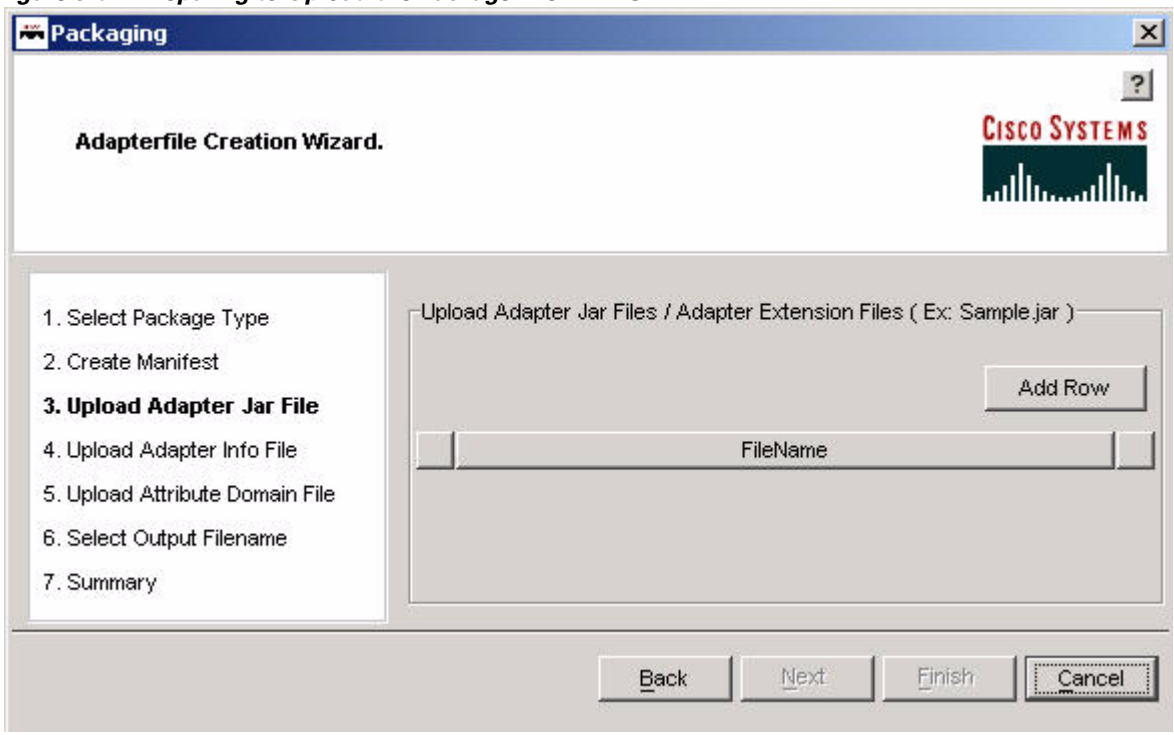
Table 3-1 Adapter File Creation Manifest Information (continued)

Field	Description
Version	(Optional) File version.
Comments	(Optional) Comments about the file.

d. Click **Next**.

The wizard prompts (Figure 3-8) you to upload the Java archive (.jar) file.

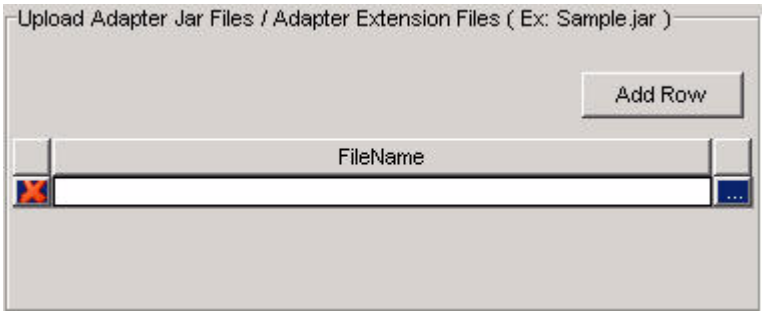
Figure 3-8 Preparing to Upload the Package File in ADS



e. Click **Add Row**.

A space appears (Figure 3-9) for the filename.

Figure 3-9 Selecting a file.

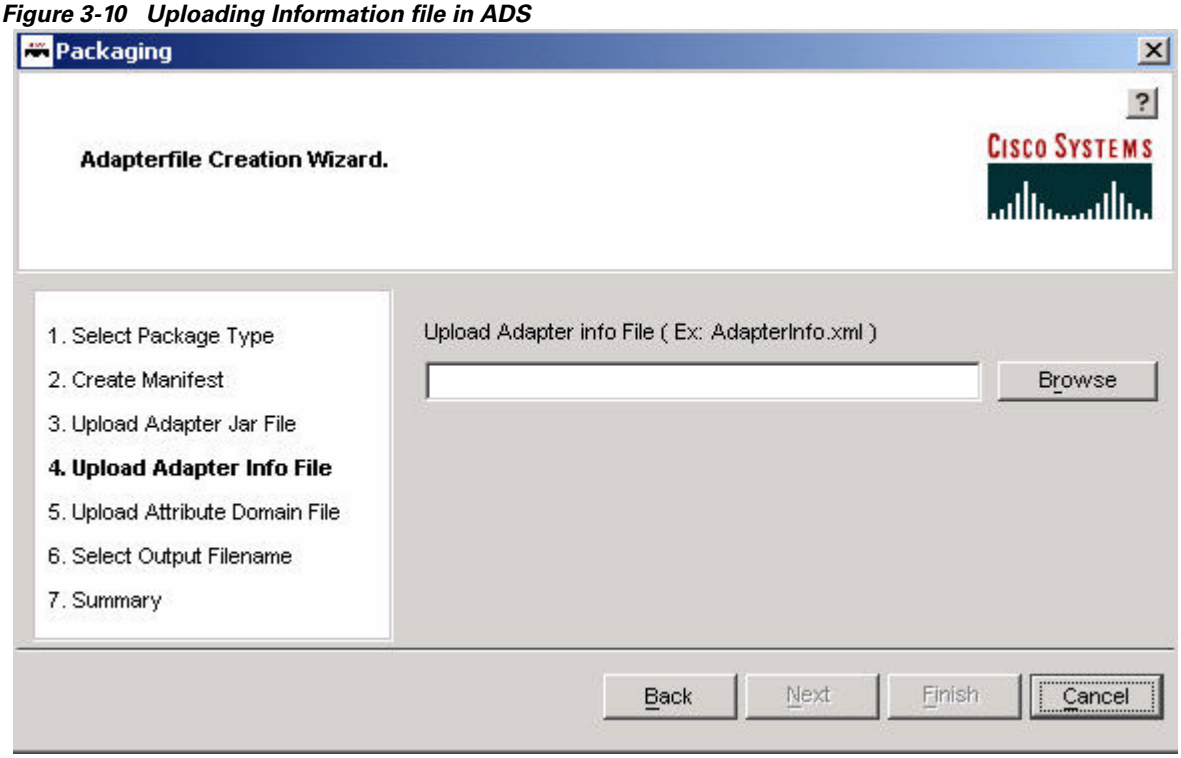


f. Using the navigator button to the right of the field, select a file for uploading.

g. Click **Next**.

The wizard prompts (Figure 3-10) you to upload the information file.

Figure 3-10 Uploading Information file in ADS

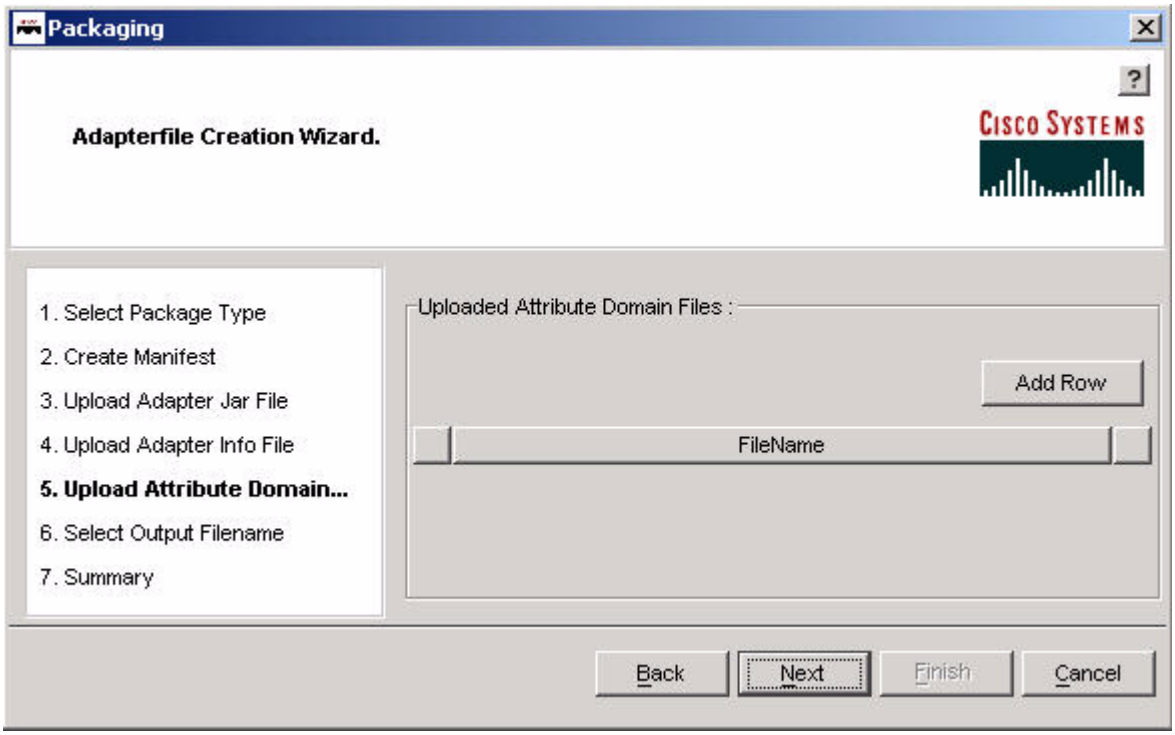


h. Using the **Browse** button, select an information file.

i. Click **Next**.

The wizard prompts (Figure 3-11) you to upload the Attribute Domain file.

Figure 3-11 Uploading Attribute Domain File in ADS



j. Click **Add Row**.

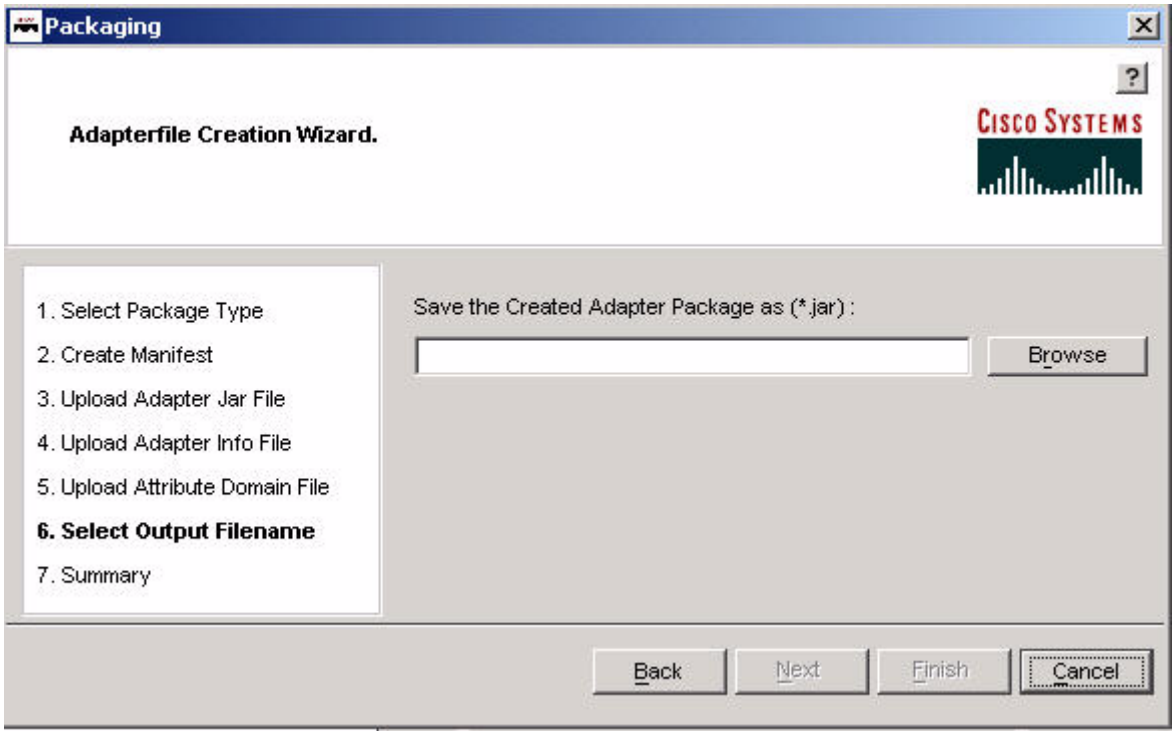
A space appears for the file name.

k. Using the **Browse** button, select an information file.

l. Click **Next**.

You are prompted (Figure 3-12) to save the package.

Figure 3-12 Assigning output filename

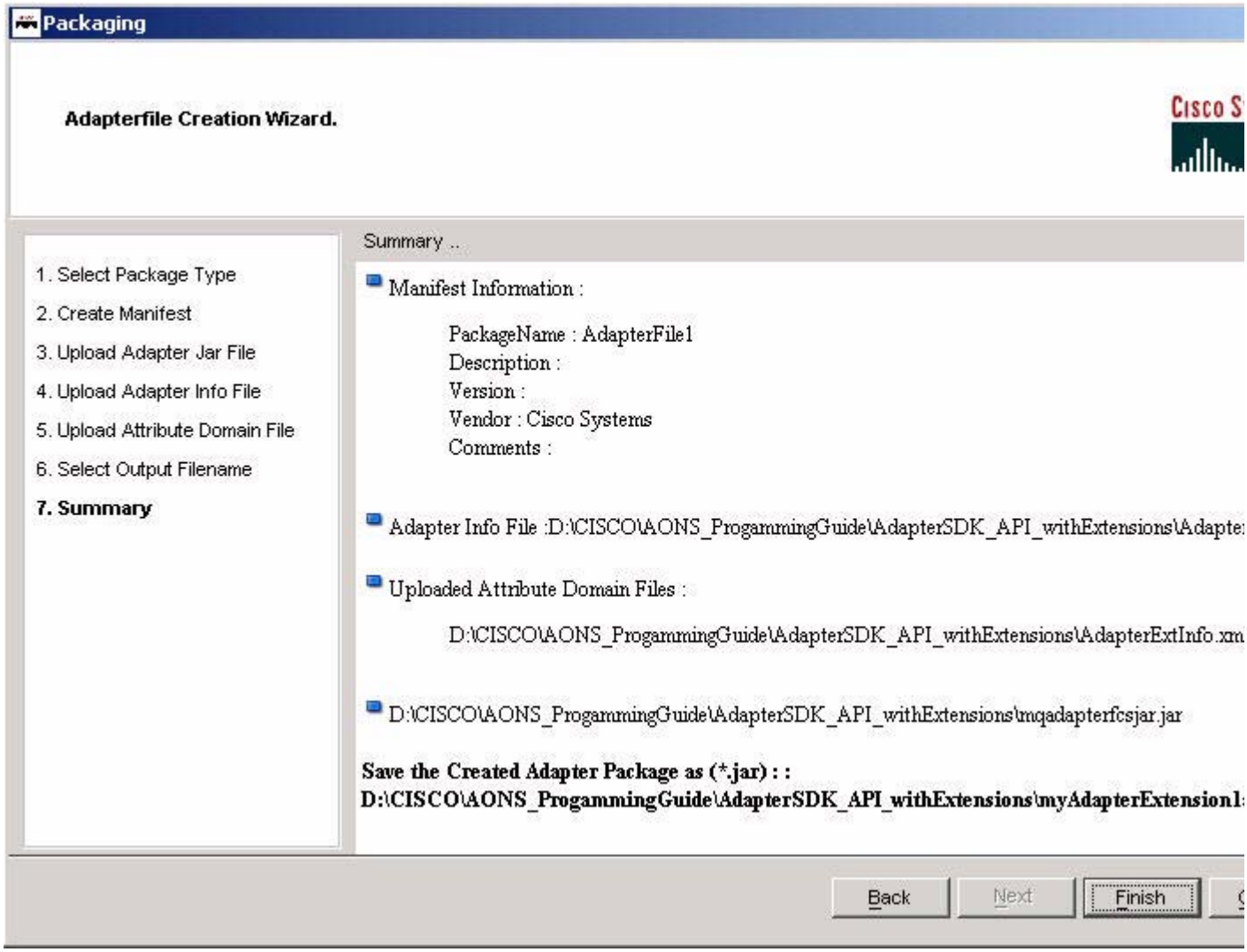


m. Using the **Browse** button, select a package filename (.jar).

n. Click **Next**.

A summary screen (Figure 3-13) appears.

Figure 3-13 Package Summary

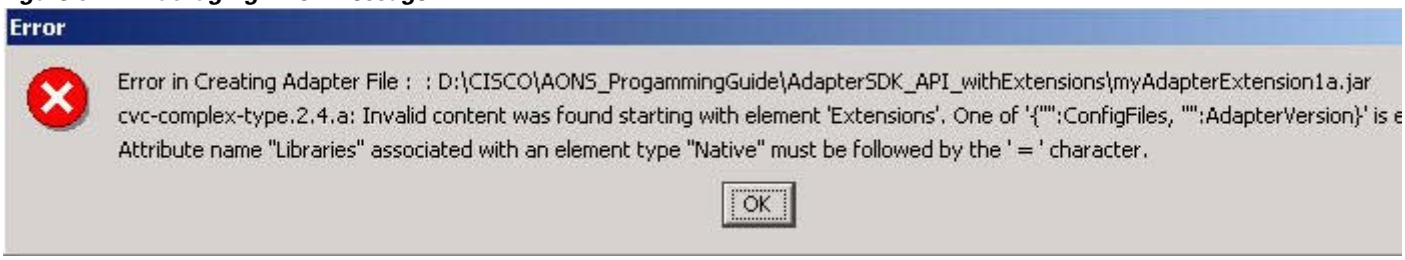


o. Review the package and click **Finish**.

You have successfully uploaded the package.

Otherwise, an error message (Figure 3-14) appears.

Figure 3-14 Packaging Error Message



In this case, click **Back**, enter the correct filename, and complete the process.

For a list of the components in an adapter package, see the "Adapter Package Content" section on page 3-43.

Next, see Uploading the MQ Adapter to learn how to upload the new package to the AON Management Console (AMC).

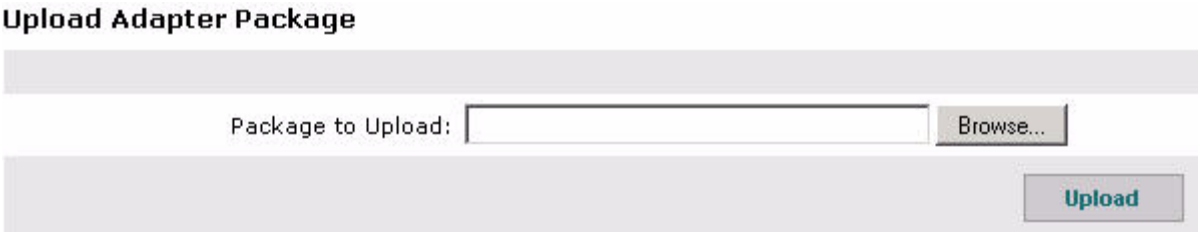
Step 2 Upload the package to AMC.

Using the AMC, upload the package file.

a. Select **Admin > Upload Adapter Package**.

An upload dialog (Figure 3-15) appears.

Figure 3-15 Selecting Package to Upload



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

The package is uploaded. ADS lists (Figure 3-16) the newly uploaded package.

Figure 3-16 Uploading and Registering Package

Step 3 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 3-42
- Stock Trading Company Embedded Adapter Use Case, page 3-42

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 PEP model must be able to modify the Request to add a Request-ID for correlation purposes.

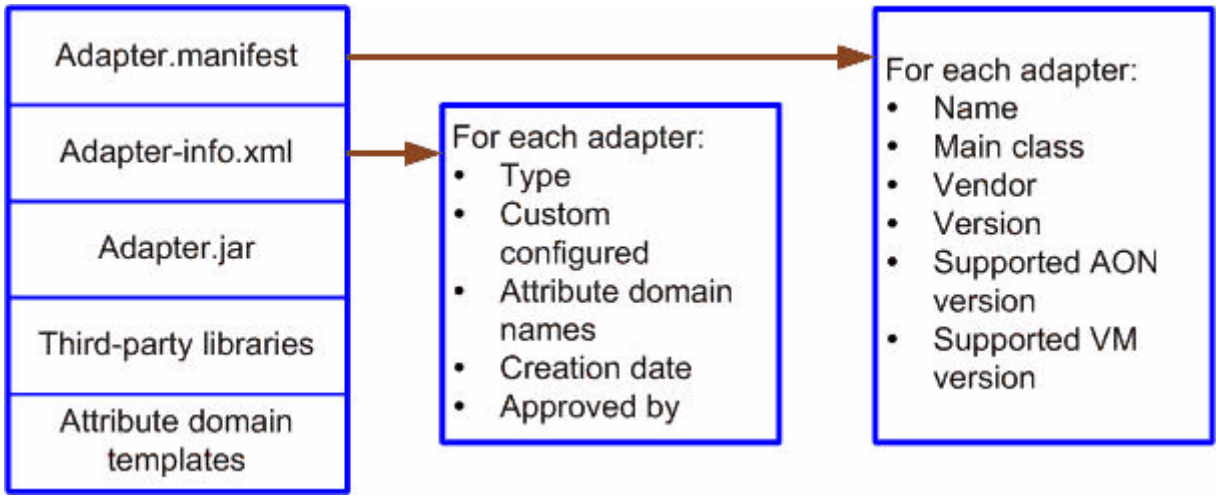
  Requires a globally unique identifier (GUID) bladelet to create the Request-ID.

  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 3-17.

Figure 3-17 Embedded Adapter Package



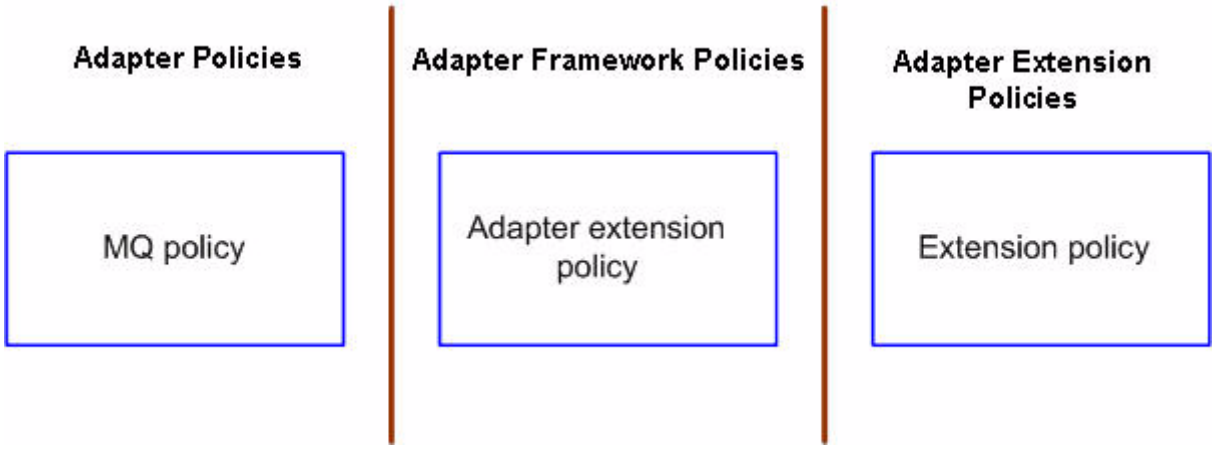
Compiling the Custom Adapter

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

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 3-18 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 3-80.

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 3-47

Downloading and Configuring MQ Visual Edit

You should also configure MQ Visual Edit. Follow these instructions.

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 screens at: http://www.caplitalware.biz/mqve_screenshots.html.

Developing the MQ Adapter for One 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 3-48
- Configuring the MQ Adapter for One Node, page 3-50
- Deploying the MQ Adapter for One Node, page 3-56
- Validating the MQ Adapter for One Node, page 3-58

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**.

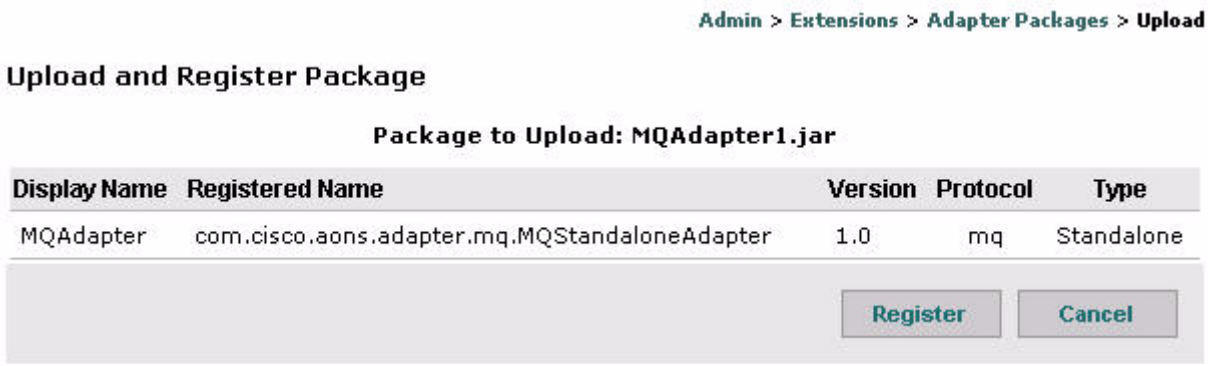
An upload dialog appears.

Figure 3-19 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 3-20 Uploading and Registering the Package

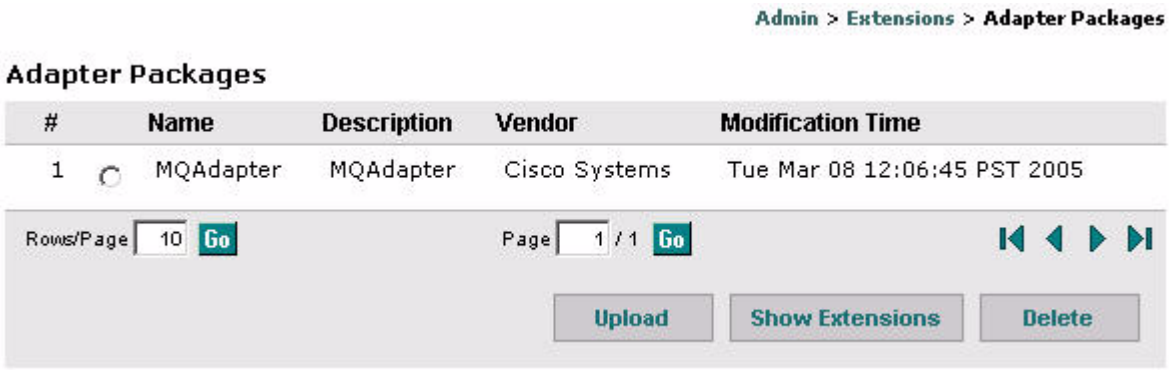


Step 2 Register the package with AMC.

a. Click **Register**.

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

Figure 3-21 Registering the Package



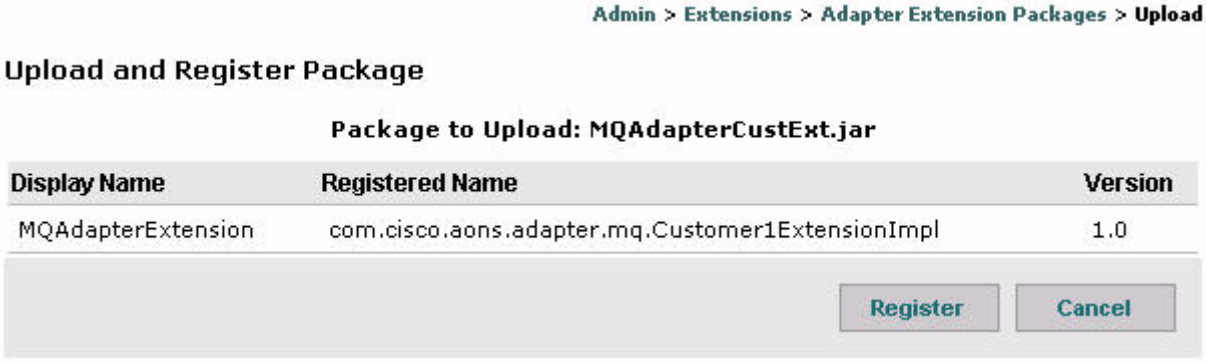
Step 3 Upload the MQ Adapter Extension package to the AMC.

a. Using the AMC, select **Admin > Extensions > Adapter Extension Packages > Upload**. An upload dialog appears.

Figure 3-22 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 3-23 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 3-50.

Configuring the MQ Adapter for One Node

You 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. 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. Set to FINAL_Q.
OutboundQueueName	Name of the outbound queue. Set to FINAL_Q.
OutboundIsDefaultQueue	True or false. Default = false. 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.
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).
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.
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**.

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.
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).

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.

- 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 3-56.

Deploying the MQ Adapter for One Node

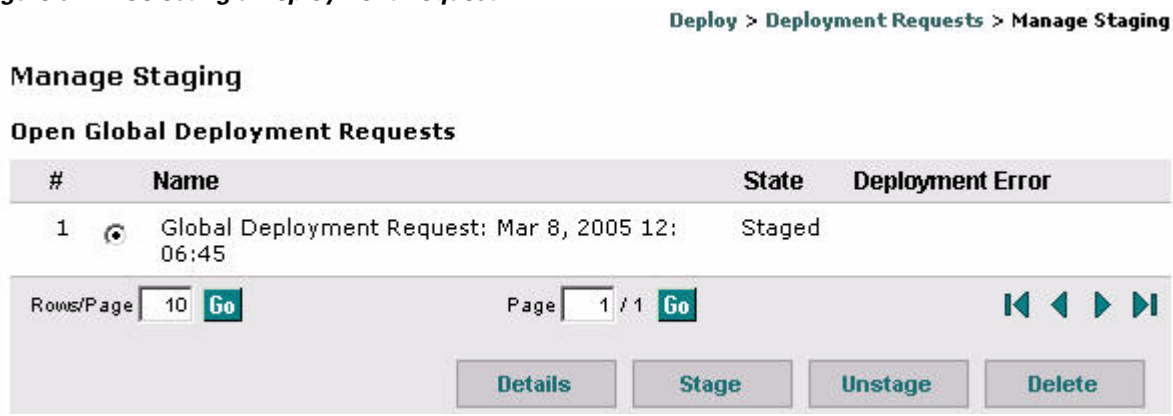
After the MQ adapter is uploaded and registered, you can deploy it. Follow the steps listed below.

Step 1 Stage the MQ adapter.

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

The resulting screen lists the current deployment requests.

Figure 3-24 Selecting a Deployment Request



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

c. Follow the onscreen 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 3-25 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 3-58.

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 3-59
- Configuring the MQ Adapter for Two Nodes Using the Same Queue Manager, page 3-60
- Setting Up a Next Hop Domain, page 3-69

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 3-60.

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.
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.
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.

Field	Definition
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).
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.

Field	Definition
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.

Field	Definition
Protocol	Protocol. Possible values: AONP-HTTP (default) or AONP-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 3-2 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 com.ibm.mq.MQException: MQJE001: Completion Code 2, Reason 2085 at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1527) at com.ibm.mq.MQQueueManager.accessQueue(MQQueueManager.java:1579) at com.cisco.aons.adapter.mq.QueueConfig.setupQueues(QueueConfig.java:289)	• Check for a missing or incorrect TRANSACTION Queue Name. • Verify that the Transaction Queue Name in both the AONS Configuration and the Websphere MQ Configuration. • Verify that the case of the Transaction Queue Name (CAPS or small letter). • Verify that the Transaction Queue Name is correct. • Restart the Queue Manager, if the problem still exists.
Code 2009/ 6001	MQJE001: An MQException occurred: Completion Code 2, Reason 2009 MQJE016: MQ queue manager closed channel immediately during connect Closure reason = 2009 MQJE001: An MQException occurred: Completion Code 2, Reason 2009 MQJE016: MQ queue manager closed channel immediately during connect Closure reason = 2009 ERROR [MEC-Q-4] mec.adapter.MQAdapter errorCode = 6001	• Check for a missing or incorrect CHANNEL Name. • Verify the Channel Name in both the AONS Configuration and Websphere MQ Configuration. • Verify that the Channel Queue Name is correct. • Restart the Queue Manager, if the problem still exists.
Code 6050	MEC-Q-4 ERROR adapter.MQAdapter - errorCode = 6050 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. • Verify that the Destination Queue exists. • Verify the MQ Adapter Destination Configuration. • Verify the endpoint for setting the proper Destination URI.

Table 3-2 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 [java] at java.util.ResourceBundle.throwMissingResourceException(ResourceBundle.java:804) [java] at java.util.ResourceBundle.getBundleImpl(ResourceBundle.java:773) [java] at java.util.ResourceBundle.getBundle(ResourceBundle.java:538)	• Verify that the Adapter jar contains the "MQMessagesBundle.properties" file. • Check to be certain that the jar file is correctly built.
Code 6004	ERROR [MEC-Q-4] mec.adapter.MQAdapter errorCode = 6004 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 "AAAAAAAAAAAAAAAAAAAAAAAAAA" is all blanks. 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. The end-point is not putting messages in the reply queue with the correct CorrelationID.
Code 5003	MQJE011: Socket connection attempt refused 12-22-2004 22:12:22 DEBUG [main] mq - MQOutboundExceptionHandler.handleAdapterException 12-22-2004 22:12:22 ERROR [main] mq - errorCode = 5003 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. • Verify that the MQ Broker is running on the specified port. • Verify that the MQ Adapter configuration specifies the port on which the MQ broker is running.

Table 3-2 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 3-2 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. MEC-Q-2 DEBUG adapter.MQAdapter - QueuePeeker: AON.QM:AON.LOCAL.Q NOT yet acquired...no need to Peek. 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. In a multiblade-virtual cluster situation, one reply is needed for each blade. If there are two blades, you must configure two reply queues. • Verify that the WCCP is setup correctly. • Check the log for each blade to determine if each blade has acquired one reply. 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. 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) 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 3-2 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 2005-07-01 14:15:18,384 main ERROR adapter.MQAdapter - errorCode = 5002 2005-07-01 14:15:18,384 main ERROR adapter.MQAdapter - reason = Error invalid data for attribute OutboundQueueManagerName 2005-07-01 14:15:18,384 main ERROR adapter.MQAdapter - exception = InvalidAttributeName Passed com.cisco.aons.util.DomainException:InvalidAttributeName Passed at com.cisco.aons.pm.DomainReaderImpl.getValues(DomainReaderImpl.java:143) at com.cisco.aons.adapter.mq.Configuration.getAttributeValue(Configuration.java:401) at com.cisco.aons.adapter.mq.Configuration.setOutboundQueueManagers(Configuration.java:276) at com.cisco.aons.adapter.mq.Configuration.doInitialize(Configuration.java:75) at com.cisco.aons.core.ExecutionController.initialize(ExecutionController.java:254) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source) at java.lang.reflect.Method.invoke(Unknown Source)	This message appears when AMC creates the wrong property sets in MQAdapter.xml.

Table 3-2 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) at com.cisco.aons.bootstrap.AONSBootStrapper.bootstrap(AONSBootStrapper.java:92) 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" • Export the following variable: export LD_ASSUME_KERNEL=2.4.19 • Restart the queue manage
Socket Connection Refused - MQJE011	MQJE011: Socket connection attempt refused	• Verify that the MQ Broker is running on the specified port. • 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. If no messages are available in DEFAULT_Q, this error would appear. The server end-point should catch and handle this exception. • 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. • 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 Administrator 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(IAONMessage, 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(IAONMessage). 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. This section introduces the API components. For background information, see the *AON Administration and Installation Guide*.

The AON Adapter SDK includes specialized interfaces and classes listed in the following section:

Note The AONS common set of interfaces and classes should be used in conjunction with the Custom Adapter API. For descriptions, see Appendix A, "AONS Common Specification."

Adapter Package

The Adapter package (com.cisco.aons.adapter) defines the interfaces and classes:

Interfaces

The com.cisco.aons.adapter package includes the interfaces are summarized below.

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 Adapter request.
ADAPTER_RESPONSE	public static final int ADAPTER_RESPONSE Adapter response.
AONS_ADAPTER_LOGGER	public static final java.lang.String AONS_ADAPTER_LOGGER
CLEAR_MODE	public static final int CLEAR_MODE Indicates a clear mode.

EVENT_ACCEPT	public static final int EVENT_ACCEPT Accept the event.
EVENT_CLOSE	public static final int EVENT_CLOSE Close the channel.
EVENT_CONNECT	public static final int EVENT_CONNECT Connect the event.
EVENT_INPUT_CLOSE	public static final int EVENT_INPUT_CLOSE Indicates that the input channel is closed.
EVENT_OUT_OF_BUFFER	public static final int EVENT_OUT_OF_BUFFER Indicates that the event has an out of buffer status.
EVENT_OUTPUT_CLOSE	public static final int EVENT_OUTPUT_CLOSE Indicates that the output channel is closed.
EVENT_READ	public static final int EVENT_READ Read event.
EVENT_WRITE	public static final int EVENT_WRITE Write event.
PROTOCOL	public static final java.lang.String[] PROTOCOL
SECURE_MODE	public static final int SECURE_MODE Indicates a secure mode.
STATUS_OK	public static final int STATUS_OK Indicates an OK status.

IAdapterContext

This interface is used to get the adapter context. It includes the single method summarized below.

Methods	Description
getAdvisoryListener	public IAdvisoryListener getAdvistoryListener() Returns the IAdvisoryListener.

IAdapterDescriptor

This interface defines adapter types such as embedded, standalone, and start up mode.It includes the methods listed below.

Methods	Description
getAdapterClassName	public java.lang.String getAdapterClassName() Returns (string) the adapter class name.
getAdapterName	public java.lang.String getAdapterName() Returns (string) the adapter name.
getAttributeDomainNames	public java.lang.String[] getAttributeDomainNames() Returns (string[]) the attribute domain name.
getConfigFileName	public java.lang.String getConfigFileName() Returns (string) the configuration file name.
getDomainReader	public DomainReader getDomainReader(java.lang.String pAttributeName) throws AdapterException Parameters: pAttributeName—String Returns the domain reader.
getIdentifier	public int getIdentifier() Returns (int) the identifier.
getMode	public int getMode() Returns(int) the mode.
getNetworkListeners	public java.util.Iterator getNetworkListeners() Returns (iterator) the network listeners.
getOutboxHandlerClass	public java.lang.Class getOutboxHandlerClass() throws java.lang.ClassNotFoundException Returns the class.‘
getProtocol	public java.lang.String getProtocol() Returns (string) the protocol name.
getProtocolAliases	public java.lang.String[] getProtocolAliases() Returns sString) one or more alternate protocol aliases for the supported protocol.

getReceiveHandlerClass()	public java.lang.Class getReceiveHandlerClass() throws java.lang.ClassNotFoundException Returns the class.
getResource	public java.io.InputStream getResource(java.lang.String pConfigFile) throws java.util.MissingResourceException Parameters: pConfigFile—String Returns the input stream resource.
getSendHandlerClass	public java.lang.Class getSendHandlerClass() throws java.lang.ClassNotFoundException Returns the class.
getType	public int getType() Returns (int) the type.
isActive	public boolean isActive() Returns a boolean (equivalent to yes/no or is/is not active).

IAdapterManager

This interface defines the adapter manager. It includes the methods listed below.

Methods	Description
dispatchTask	public void dispatchTask(AdapterTask pTask) Parameters: pTask—IAdapterTask, the task. Schedules the task for immediate execution.
getLogger	public Log getLogger() 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() Gets the message builder to create messages

getProtocol	public int getProtocol() Returns the protocol ID associated with the adapter
getPersistentManager	public IPersistentManager getPersistentManager() throws AdapterException Returns IPersistentManager, the persistent manager.
getResourceManager	public IResourceManager getResourceManager() Returns IResourceManager, the resource manager.
handleError	public void handleError(IMessageHandler pHandler, int pErrorCode) Parameters: pErrorCode—Error code Handler for errors. An adapter can use this method to report an error to AON.
handleException	public void handleException(IMessageHandler pHandler, java.lang.Exception pEx) Handler for exceptions. The way that AON handles the exception depends on the error code.
isLocalHostAddress	public boolean isLocalHostAddress(java.lang.String pHostIp) Checks to determine if HostIp belongs to the local host.

getProtocol	public int getProtocol() Returns the protocol ID associated with the adapter
getPersistentManager	public IPersistentManager getPersistentManager() throws AdapterException Returns IPersistentManager, the persistent manager.
getResourceManager	public IResourceManager getResourceManager() Returns IResourceManager, the resource manager.
handleError	public void handleError(IMessageHandler pHandler, int pErrorCode) Parameters: pErrorCode—Error code Handler for errors. An adapter can use this method to report an error to AON.
handleException	public void handleException(IMessageHandler pHandler, java.lang.Exception pEx) Handler for exceptions. The way that AON handles the exception depends on the error code.
isLocalHostAddress	public boolean isLocalHostAddress(java.lang.String pHostIp) Checks to determine if HostIp belongs to the local host.

getProtocol	public int getProtocol() Returns the protocol ID associated with the adapter
getPersistentManager	public IPersistentManager getPersistentManager() throws AdapterException Returns IPersistentManager, the persistent manager.
getResourceManager	public IResourceManager getResourceManager() Returns IResourceManager, the resource manager.
handleError	public void handleError(IMessageHandler pHandler, int pErrorCode) Parameters: pErrorCode—Error code Handler for errors. An adapter can use this method to report an error to AON.
handleException	public void handleException(IMessageHandler pHandler, java.lang.Exception pEx) Handler for exceptions. The way that AON handles the exception depends on the error code.
isLocalHostAddress	public boolean isLocalHostAddress(java.lang.String pHostIp) Checks to determine if HostIp belongs to the local host.

getProtocol	public int getProtocol() Returns the protocol ID associated with the adapter
getPersistentManager	public IPersistentManager getPersistentManager() throws AdapterException Returns IPersistentManager, the persistent manager.
getResourceManager	public IResourceManager getResourceManager() Returns IResourceManager, the resource manager.
handleError	public void handleError(IMessageHandler pHandler, int pErrorCode) Parameters: pErrorCode—Error code Handler for errors. An adapter can use this method to report an error to AON.
handleException	public void handleException(IMessageHandler pHandler, java.lang.Exception pEx) Handler for exceptions. The way that AON handles the exception depends on the error code.
isLocalHostAddress	public boolean isLocalHostAddress(java.lang.String pHostIp) Checks to determine if HostIp belongs to the local host.

registerForAttachment	public int registerForAttachment(java.lang.String pAttachKey) Parameters: pAttachKey—Key that the adapter uses to register the attachment. Returns the key index. IMessageHandler.getAttachment is used with this index to retrieve the attachment. 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. 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, java.lang.String pClassName)
scheduleTask	public void scheduleTask(SchedulableTask pTask, int pInterval) Parameters: pTask—SchedulableTask, the task pInterval—int the interval specified in seconds Schedules the task for execution after a specified interval. The task will not be removed from this schedule until it is de-registered.

IAdvisoryListener

This interface is used to listen for events. It includes the methods listed below.

Methods	Description
onEvent	public void onEvent(AdapterEvent pEvent)

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 methods listed below.

Methods	Description
getConnectionId()	public java.lang.String getConnectionId() Returns the connection ID associated with this connection.
getDestinationIP	public java.net.InetAddress getDestinationIP() Returns the destination IP.
getDestinationPort	public int getDestinationPort() Returns (int) the destination port number.
getReader	public IAdapterReader getReader() Returns the IAdapterReader interface to define a reader.
getSourceIP	public java.net.InetAddress getSourceIP() Returns (InetAddress) the source IP.
getSourcePort	public int getSourcePort() Returns (int) the source port number.
getSSLCertificate	public java.security.cert.Certificate getSSLCertificate() Returns the SSL certificate.
getWriter	public IAdapterWriter getWriter() Returns the IAdapterWriter to define a writer
isSecure	public boolean isSecure() Returns “true” if the connection is a secure (SSL) connection.

IConnectionReceiver

This interface defines the connection receiver. It includes the method listed below.

Methods	Description
release	public void release(IAONSMessage pMessage) Returns a connection receiver for the given pMessage.

IDeliveryContextCallback

This interface returns the DeliveryContexts. It includes the method listed below.

Methods	Description
recoveredDeliveryContexts	public void recoveredDeliveryContexts(java.util.HashMap pContexts) throws AdapterException 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

This interface defines outbound delivery group. One or more DeliveryGroups may be associated with DeliveryGroups. It includes the method listed below.

Methods	Description
getGroup	public IGroup getGroup() Gets the outbound delivery group.

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	public URI getDeliveryDestination() Returns the destination URI associated with this delivery group.
getDeliveryGroupId	public java.lang.String getDeliveryGroupId() Returns the unique delivery group identifier associated with this delivery group.
getGroup	public IGroup getGroup() 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	public void notifyCommit() 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	public void notifyReadyToReceive(IAONSMessage pLastDelivered) Parameters: pLastDelivered—last message written out 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	public void notifyReleaseCompleted() Callback indicating that the adapter has successfully released this delivery group. The adapter must release this delivery group when IDeliveryGroupDispatcher.releaseDestination() is called.

notifyReleaseFailure	public void notifyReleaseFailure(AdapterException pException) 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()	public void notifyRollback() 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

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 listed below.

Methods	Description
getAdapter	public Adapter getAdapter() Returns the adapter associated with this delivery group dispatcher
isTransacted	public boolean isTransacted() Returns true if this delivery group dispatch is transacted, otherwise false. Determines if this delivery group dispatcher is transacted.

releaseDestination	public void releaseDestination() throws AdapterException 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. After releasing the destination, the adapter must indicate completion by calling IDeliveryGroupCallback.notifyReleaseCompleted or IDeliveryGroupCallback.notifyReleaseFailure(AdapterException).
sendMessage	public void sendMessage(IAONSMessage pMessage, IMessageHandler pHandler, int pMsgType) throwsAdapterException Parameters: pMessage—message to be sent pHandler— message handler to be used pMsgType—message type Sends (dispatches) the given message of the given type to the adapter using the given message handler.

IEmbeddedAdapterManager

This interface defines an embedded adapter manager. It includes the methods listed below.

Methods	Description
createConnection	public IConnectionContext createConnection(Adapter pAdapter, java.net.InetAddress pIPAddr, int pPort, boolean pIsSecure, boolean pIsCacheable) throws AdapterException Parameters: pAdapter—adapter which is creating this connection pIPAddr—IP address for connection pPort—port for this connection pIsSecure—asks if the connection is secure connection pIsCacheable—connection to be pooled Creates an active connection. An adapter may create outbound active connections for protocol support. **Caution** This method should not be used for the primary connection to endpoints. That is taken care of AONS runtime.

createListener	public INetworkListener createListener(Adapter pAdapter, java.net.InetAddress pIpAddress, int pPort, boolean pIsSecure) throws AdapterException Parameters: pAdapter—adapter associated with this listener pIPAddress—IP address associated with this listener pPort—port associated with this listener pIsSecure—asks if the listener is secure. 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.
getBufferManager	public IBufferManager getBufferManager() Gets IBufferManager to create a buffer manager. The BufferManager provides the interface for all buffer related operations such as reading and writing messages.
getDefaultConnKeepAlive	public long getDefaultConnKeepAlive() Returns (long) the default connection keep alive value.
getDefaultOutboxHandler	public AbstractOutboxHandler getDefaultOutboxHandler(IAONSMessage pMessage) Gets the AbstractOutboxHandler class to generate a default hotbox handler.

getDeliveryDispatcher	public IDeliveryGroupDispatcher getDeliveryDispatcher(IDeliveryGroupCallback pCallback) throws AdapterInitException Gets the delivery dispatcher associated with this adapter.
registerEvent	public void registerEvent(IConnectionContext pContext, int pEvent) throws java.nio.channels.ClosedChannelException 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

Handles error codes. It has the fields listed below

Fields	Description
INSUFFICIENT_STORAGE	public static final int INSUFFICIENT_STORAGE Indicates an insufficient storage condition
IO_ERROR	public static final int IO_ERROR Indicates an IO error.
REQ_TXN_FAILED	public static final int REQ_TXN_FAILED Indicates that a request transaction failed.

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 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 methods and fields summarized below.

Methods	Description
getGroupId()	public java.lang.String getGroupId() Gets the unique group identifier associated with this group.

getResourceId()	public long getResourceId() Gets the unique resource identifier associated with this group.
getType()	public int getType() Gets the type associated with this group

Fields	Description
ORDERED	public static final int ORDERED Field indicates that the group is ordered
PERSISTENT_GROUP	public static final int PERSISTENT_GROUP Field indicates that the group is persistent.

IMessageContextStore

Interface handles message context store deletion and recovery operations. It includes the methods listed below.

Methods	Description
deleteMessageContext	public boolean deleteMessageContext(java.lang.String pMessageId, URI pRequestURI) throws AdapterException Deletes a specific message context.
deleteMessageContext	public boolean deleteMessageContext(java.lang.String pMessageId) throws AdapterException Deletes a specific (URI) message context.

recoverDeliveryContexts	public void recoverDeliveryContexts(IDeliveryContextCallback pCallback) throws AdapterException Parameters: pCallback—MessageContextCallback Called at initialization to recover all the message context from the adapter. After the adapter recovers the message contexts, they are returned via the callback. The adapter can make the callback multiple times.
recoverMessageContext	public IMessageContext recoverMessageContext(java.lang.String pMessageId) throws AdapterException Parameters: pMessageId—String Returns IMessageContext to recover a specific message context on demand.
Parameters: pMessageId—String recoverMessageContext	public IMessageContext recoverMessageContext(java.lang.String pMessageId, URI pRequestURI) throws AdapterException Parameters: pMessageId—String Recovers a URI-identified message context.

IMessageDispatcher

Dispatches AON messages for processing in the AON run-time system. It includes the methods summarized below.

Methods	Description
dispatch	public void dispatch(IAONSMessage pMessage, IMessageHandlerCallback pHandlerCallback) throws AdapterException Parameters pMessage—message to be dispatched. Dispatches the message read in to AON Inbox for PEP processing The adapter has read the message (context of the adapter thread) and uses this API to dispatch it.
dispatch	public void dispatch(IAONSMessage pMessage, IMessageHandlerCallback pHandlerCallback, IMessageWriteCompleteCallback pCallback) throws AdapterException Parameters pMessage—message to be dispatched. pCallback—the callback to be called when message write has been completed. Dispatches the message read in to AON Inbox for PEP processing
dispatch	public void dispatch(StandaloneMessageReader pMessageReader) throws AdapterException Parameters: pMessageReader—StandaloneMessageReader Dispatches a callback to the AON Inbox. AON 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

This interface holds the AON message. It includes the method listed below.

Methods	Description
getMessage()	public IAONSMessage getMessage() throws AdapterException Returns IAONSMessage to get the message associated with this handler. The message may not have been completely read in

IPersistentManager

This interface provides a persistent manager. It includes the methods listed below.

Methods	Description
persist	public void persist(java.lang.String pSelectKey, java.io.Serializable pObject) throws AdapterException Parameters: pSelectKey—String pObject—Serializable
persist	public void persist(java.lang.String pSelectKey, java.io.Serializable pObject, long pExpiration) throws AdapterException Parameters: pSelectKey—String pObject—Serializable pExpiration—long
retrieve	public java.io.Serializable retreive(java.lang.String pSelectKey) throws AdapterException Parameters: pSelectKey—String Returns: Serializable

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	public void deregisterSource(Source pSource) Parameters: pSource—source to deregister 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	public int registerMessageContextStore(IMessageContextStore pStore, Adapter pAdapter) Parameters: pSource—message context store to register pAdapter—adapter for which the message context store will be registered Registers the given message context store for the given 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	public IGroup registerSource(OrderedSource pSource, ISourceCallback pCallback) Parameters: pSource—ordered source to register pCallback—implements callbacks that will be invoked when the sources to be acquired or yielded. 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.

registerSource	public IGroup registerSource(UnorderedSource pSource, ISourceCallback pCallback) Parameters: pSource—unordered source to register pCallback—implements callbacks that will be invoked when the sources to be acquired or yielded. 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	public boolean shouldYield(Source pSource) Parameters: pSource—source to check Returns true if the source can be distributary, 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.
yieldSource	public void yieldSource(Source pSource) Parameters: pSource—source that was yielded Callback indicating that the given source has been successfully yielded. 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 method shouldYield (Source) returns true.

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	public boolean notifyAcquire() throws AdapterException Returns false if the adapter does not want to acquire this source otherwise, true. 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. Throws AdapterException if an exception occurs while the adapter is acquiring the source.
notifyYield	public boolean notifyYield() throws AdapterException Returns false if the adapter does not want to yield this source otherwise, true. This callback to the adapter requests that it to yield the associated source. The adapter 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. Before attempting to yield, the adapter should determine if it is necessary by calling the IResourceManager method shouldYield(Source). After the adapter yields, it must indicate completion by calling the IResourceManager method yieldSource(Source). Throws AdapterException if an exception occurs as the adapter is yielding the source.

IStandaloneAdapterManager

This interface represents the standalone adapter manager. It inherits the following methods from IAdapterManager: dispatchTask, getLogger, getMessageBuilder, getPersistentManager, getProtocol, getResourceManager, handleError, handleException, isLocalHostAddress, registerForAttachment, registerForDeliveryContext, and scheduleTask.

IStandaloneAdapterManager has the single method summarized below.

Methods	Description
dispatch	public void dispatch(AbstractOutboxHandler pOutboxHandler) throws AdapterException 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.

Classes

The com.cisco.aons.adapter package classes listed below.

AbstractOutboxHandler

Extending java.lang.Object, this abstract class defines an outbox handler. It includes the methods summarized below.

Methods	Description
execute	public final void execute() Sends out the message.

initialize	public void initialize(StandaloneAdapter pAdapter, IMessageHandler pHandler, IAONSMessage pMessage, int pMsgType) Parameters: pAdapter—adapter sending out the message pHandler—message processing the handler associated with this message processing instance. pMessage—message to be sent out pMsgType—message type being sent out. Initializes the Outboxhandler for the message to be sent out of AON. This method is called by the AONS runtime framework. It should not be invoked by adapters.
sendMessage	public abstract void sendMessage(int pMsgType) throws AdapterException 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.

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 methods summarized below.

Methods	Description
doInitialize	protected abstract void doInitialize() throws AdapterInitException 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.
doShutdown	public abstract void doShutdown() throws AdapterException Shuts down the adapter.
doShutdownInput	public abstract void doShutdownInput() throws AdapterException Shuts down input to the adapter. Adapters close their input mechanism and no longer process new application messages. This call is made to the adapter when AON is deactivated or the adapter is reloaded.
doShutdownOutput	public abstract void doShutdownOutput() throws AdapterException Shuts down adapter output. Adapters stop sending out messages.
doStart	public abstract void doStart() throws AdapterException Starts the adapter.
getAdapterDescriptor	public IAdapterDescriptor getAdapterDescriptor() Returns the adapter descriptor associated with this adapter.
getAdapterManager	public final IAdapterManager getAdapterManager() Uses IAdapterManager to return an adapter manager for this adapter.
getAdapterName	public final java.lang.String getAdapterName() Returns the adapter name.

getDeliveryDispatcher	public abstract IDeliveryGroupDispatcher getDeliveryDispatcher(IDeliveryGroupCallback pCallback) throws AdapterException Parameters: pCallback—callback on which the adapter send notification. 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() 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.
getOutboxHandler	public abstract AbstractOutboxHandler getOutboxHandler(IMessageHandler pMsgHandler, IAONSMessage pMessage, int pMsgType) throws AdapterException Uses the AbstractOutboxHandler class to generate an outbox handler for the adapter. 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() Returns the adapter type (integer).
initialize	protected void initialize(IAdapterDescriptor pDescriptor, IAdapterManager pAdapterManager) throws AdapterInitException Initializes the descriptor and adapter manager.
isLoopBack	public boolean isLoopBack(URI pDestURI) throws AdapterException 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 Reloads the adapter with a new configuration

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, java.lang.Throwable pEx)
AdapterException	public AdapterException(int pErrorCode, java.lang.Throwable pEx)
AdapterException	public AdapterException(int pErrorCode)
AdapterException	public AdapterException(java.lang.String pMessage, 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() Returns the policy key (string) of the extension.
getExtPolicyReader	public final DomainReader getExtPolicyReader() Returns the extension policy reader (DomainReader).
initialize	public final void initialize(java.lang.String pAdapterPolicyKey, java.lang.String pExtensionPolicyKey, DomainReader) throws AdapterException Parameters: pAdapterPolicyKey—String adapter policy key, the property set key. pExtensionPolicyKey—String extension policy key, the property set key/ 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 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 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:

he methods summarized below.

Constructor	Description
AdapterInitException	public AdapterInitException(java.lang.Exception pException)
AdapterInitException	public AdapterInitException(java.lang.String pMessage, int pErrorCode, java.lang.Throwable pException)
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

This is a tagging interface that marks implementations of this interface as an adapter task AdatperTask includes the following methods inherited from java.lang.Object:clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, and wait (three expressions). AdapterTask also includes one constructor: public AdapterTask().

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) throws AdapterException Parameters: pCallback—callback on which the adapter send notification. 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

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, IConnectionContext pContext) throws AdapterException Parameters: pEvent—int pContext—IConnectionContext After connection initialization, this method is called until the adapter makes the connection handshake.
discardMessage	public void discardMessage() throws AdapterException Called to discard a message
doInitialize	public abstract void doInitialize(int pMode, int pMessageType, IConnectionContext pContext) throws AdapterException Called to initialize the handler.
fetchMessage	public abstract int fetchMessage(int pEvent, IConnectionContext pContext) throws AdapterException Parameters: pEvent—IO event on which this method has been invoked. pContext—IConnectionContext Returns: 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. EVENT_WRITE—if the adapter has to write 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() Gets the adapter associated with this handler

getConnectionContext	public final void setConnectionContext(IConnectionContext pConnCtx) Gets the connection context associated with this adapter handler.
getHandlerType	public final int getHandlerType() Returns the handler type: HANDLER_RECEIVE—type if the adapter is of type AdapterReceiveHandler HANDLET_SEND—type if the adapter is of type AdapterSendHandler
getMessage	public abstract IAONSMessage getMessage() throws AdapterException 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() 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() throws AdapterException Gets the adapter reader associated with this adapter handler.
getWriter	public IAdapterWritergetWriter() throws AdapterException Gets the adapter writer associated with this adapter handler.
initialize	public final void initialize(Adapter pAdapter, IConnectionContext pConnCtx, IMessageHandler pMsgHandler, int pMode, int pMessageType) throws AdapterException Parameters: pAdapter—adapter to which this handler belongs. pMode—protocol mode for this handler: ADAPTER_REQUEST—mode if the handler has to make a protocol request. ADAPTER_RESPONSE—mode if the handler has to make a protocol reply. Initializes the adapter handler.

keepConnectionAlive	public long keepConnectionAlive() Returns: 0—connection will be close immediately > 0—connection will remain open for specified amount of time. 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) throws AdapterException Called when a new connection is established. Returns: EVENT_READ—if the adapter needs to read in data first. (for example, HTTP). EVENT_WRITE—if the adapter needs to write some data first. (SMTP, FTP, and so on)
setConnectionContext	public final IConnectionContext getConnectionContext() 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() throws AdapterException This method is called after message processing is complete.

fetchHeader	public abstract int fetchHeader(int pEvent, IConnectionContext pContext) throws AdapterException Parameters: pEvent—int pContext—IConnectionContext Returns (int) the message header.
isHeaderComplete	public abstract boolean isHeaderComplete() throws AdapterException Checks the status of header reading. Returns true if the header is read in completely. This happens after fetchHeader returns STATUS_OK. This method also returns true if the protocol that does not require a message header.
isMessageReadComplete	public abstract boolean isMessageReadComplete() throws AdapterException 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) Parameters: pSuccess—true if the write was successful. Otherwise, false. 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.

releaseConnection	public void releaseConnection(IAONSMessage pMessage) After the message read is complete, this method Releases the underlying connection.
updateMessageContext	public abstract void updateMessageContext(IMessageHandler pMsgHandler) throws AdapterException Parameters: pMessageHandler—IMessageHandler Updates the context of a request messages. This method is not called for a response message This method is called after the adapter has dispatched a request message and just before a dispatch worker starts executing it. 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 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() 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, Adapter pAdapter, IConnectionContext pConnCtx, IMessageHandlerpMsgHandler, int pMode, int pMessageType) throws AdapterException Parameters pMessage—AON message pAdapter—adapter for this handler pMsgHandler—message handler pMode—protocol mode for this handler ADAPTER_REQUEST—mode if the handler is to make a protocol request ADAPTER_RESPONSE -- mode if the handler is to make a protocol reply Initializes the adapter handler.
reset	public void reset() throws AdapterException Clears the handler.
writeMessage	protected abstract int writeMessage(int pEvent, IConnectionContext pContext) throws AdapterException 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() Returns (string) the unique resource identifier associated with this OrderedSource.

Field	Description
mResourceId	protected java.lang.String mResourceId 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() Deregisters the tasks.
isDeregistered	public final boolean isDeregistered() 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
getAdapter	public final Adapter getAdapter() Returns the adapter associated with this source.
getAssociatedGroup	public final IGroup getAssociatedGroup() Returns the IGroup object associated with this Source.

getSourceGroup	public final java.lang.String getSourceGroup() Returns the source group associated with this Source.
getSourceSetId	public abstract java.lang.String getSourceSetId() 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() Returns the URI associated with this Source.
setGroup	public void setGroup(IGroup pGroup) Parameters: pGroup—IGroup object to associate with this object Sets the IGroup object associated with this Source.

Fields	Description
mAdapter	protected Adapter mAdapter Adapter associated with this Source.
mGroup	protected IGroup mGroup Group associated with this Source.

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) throws AdapterException 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, IAONSMessage pMessage, int pMsgType) throws AdapterException Parameters: pMessage—IAONSMessage 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() Returns a standalone adapter.
initialize	protected void initialize(IAdapterDescriptor pDescriptor, IAdapterManagerpAdapterManager) throws AdapterException Parameters: pDescriptor—IAdapterDescriptor pAdapterManager—IAdapterManager Overrides: initialize in class Adapter 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() This is the execute for this class.
readMessage	public abstract void readMessage() throws AdapterException Returns the read message. 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 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 summarized below.

Fields	Description
mAdapter	protected Adapter mAdapter Adapter associated with this Source.
mGroup	protected IGroup mGroup Group associated with this Source.

IO Package

The IO Package (com.cisco.aons.io) contains the IO handling interfaces and classes described in the following sections:

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() Clears the underlying buffers.
getAONSBuffer	public IAONSBuffer getAONSBuffer() Returns IAONSBuffer, the underlying buffers.
getBytesRead	public int getBytesRead() Returns (int) the actual bytes read from the last read operation.
getContext	public IConnectionContext getContext() Returns the underlying connection context associated with this reader.
readData	public int readData(int pSize) throws java.io.IOException Parameters: pSize—int specifies the size of the data to read, actual bytes read could be retrieve by calling getBytesRead() Reads data from the underlying channel stream. Returns the int status of the read operation: 0—successful operation < 0—an error occurred
reset	public void reset() Initializes the reader to read a new message.

Fields	Description
BUFFERS_FULL	public static final int BUFFERS_FULL Indicates that internal buffers are full.
BUFFERS_UNAVAILABLE	public static final int BUFFERS_UNAVAILABLE Indicates that buffers are unavailable.
CHANNEL_CLOSED	public static final int CHANNEL_CLOSED Indicates that the underlying channel is closed.
EOF	public static final int EOF End of stream.
OK	public static final int OK 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() Gets the underlying connection context associated with the writer.

write	public long write(java.nio.ByteBuffer[] pBuffers, int pOffset, int pLength) throws java.io.IOException Parameters: pBuffers—ByteBuffer[] The buffers from which bytes are to be retrieved 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 pLength—Maximum number (int) of buffers to be accessed. This must be non-negative number and no larger than srcs.length - offset 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. 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 $p + n - 1$. 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. 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.

write	public int write(java.nio.ByteBuffer pBuffer) throws java.io.IOException Parameters: pBuffer—ByteBuffer The buffer from which bytes are to be retrieved Returns (int) the number of written bytes, possibly zero. This method writes a sequence of bytes to this channel from a sequence of the given buffers. 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. 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. 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. 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() Returns (long) the available size for write operations.
clear	public void clear() Clears the underlying buffers for reuse.
createDataReader	public IDataReader createDataReader() Returns a data reader.
getDataSize	public int getDataSize() Returns (int) the data size.
isReleased	public boolean isReleased() Returns true if the buffer is already released, otherwise false.
isWaitingForAllocation	public boolean isWaitingForAllocation() Returns true if this buffer is waiting for new allocation.
release	public void release() Releases all buffers back to the pool.
release	public void release(int pIdx) Releases all buffers up to the specified index.
size	public int size() Returns the current size of allocated buffers.
write	public void write(byte[] b) throws java.io.IOException Parameters: b—byte[] the data Writes the specified bytes into the native buffers
write	public void write(byte[] b, int off, int len) throws java.io.IOException Parameters: b—byte[] the data off—offset (int) len—length (int) Writes data from the byte array to native buffers from the specified offset, up to the given length.

write	public void write(byte b) throws java.io.IOException Parameters: b—byte Writes a single byte to native buffers
write	public void write(java.nio.CharBuffer pCharBuffer, java.nio.charset.CharsetEncoder mEncoder) throws java.io.IOException Parameters: pCharBuffer—CharBuffer the data mEncoder—CharsetEncoder the encoder Writes data from the char buffer to native buffers using the supplied charset encoding.
write	public int write(IAdapterReader pReader) throws java.io.IOException Parameters: pReader—IAdapterReader the adapter reader 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, int pQualifier) throws BufferException Parameters: pSize—requested size requested (int) pQualifier—qualifier (int) indicating the processing state of the message Increases the buffer by the specified size. The qualifier indicates the processing state of the message for which buffers are requested.
getBuffer	public IAONSBuffer getBuffer(IAONSBuffer pAONSBuffer, int pSize, int pQualifier) throws BufferException Parameters: pAONSBuffer—AONSBuffer the existing AONS buffer pSize—size (int) increase for the buffer size. pQualifier—qualifier indicating the processing state of the message Increases the size of an existing AON buffer by the specified size.

Fields	Description
IN_BOUND	public static final int IN_BOUND Specifies an inbound message.
IN_PROC	public static final int IN_PROC Specifies a message in process.
OUT_BOUND	public static final int OUT_BOUND 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() throws java.io.IOException `Returns (int) number of bytes that can be read from this data reader.`
close	public void close() Closes the read-only view and decreases the count of open read-only views by one.
compact	public int compact() throws java.io.IOException 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() Returns the parent AON Buffer.
mark	public void mark() Marks the current read offset so that reset call will set the read offset to this saved state.

read	public int read(byte[] b) throws java.io.IOException Parameters: b—byte[] the buffer into which data is read 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 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. 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. 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).

read	public int read(byte[] b, int off, int len) throws java.io.IOException Parameters: b—byte[] the buffer into which the data is read off—start offset (int) in array b at which the data is written. len—maximum number (int) of bytes to read 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. 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 f len is zero, no bytes are read and 0 is returned. Otherwise, this method attempts 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. 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. 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. In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are 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 is closed.
read	public int read(java.nio.CharBuffer pCharBuffer, java.nio.charset.CharsetDecoder pDecoder) throws java.io.IOException Parameters: pCharBuffer—CharBuffer the char buffer pDecoder—CharsetDecoder the Decoder to apply on the raw data Returns the total number (int) of bytes read into the buffer or -1 if the end of the stream has been reached 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.

readByte	public byte readByte() throws java.io.IOException, java.io.EOFException Reads a single byte.
reset	public void reset() throws java.io.IOException 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) throws java.io.IOException Parameters: len—int Rewinds the read offset by a specified length.
shift	public void shift() throws java.io.IOException 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) throws java.io.IOException Parameters: len—number (int) of bytes to be skipped 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) throws java.io.IOException 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.

update	public void update() 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) throws java.io.IOException Parameters: pWriter—IAdapterWriter adapter writer for data writing 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 AONException.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 3-137
- Classes, page 3-139

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 PEP 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.

Utilities Package

The Utilities package (com.cisco.aons.util) contains the byte buffer processing classes.

Classes

The Utility package classes are summarized below.

ByteBufferArrayInputStream

This class defines a byte buffer input stream. It includes the methods listed below.

Methods	Description
available	
close	
compact	
getDataReader	Gets a data reader.
mark(int pReadLimit	Marks the read limit.
markSupported	
read	Reads the byte buffer.
read	Reads the byte buffer.
read	Reads the byte buffer.
reset	
setBytesWritten	Sets the number of bytes to read.
skip	

ByteBufferArrayOutputStream

This class defines the byte buffer output stream. It includes the methods listed below.

Methods	Description
close	
flush	
getAONSBuffer	Gets the AON buffer.
isClosed	
write	Writes to the buffer output stream
write	Writes to the buffer output stream.
write	Writes to the buffer output stream.

ByteBufferArrayWriter

This class defines the byte buffer array writer. It includes the methods listed below.

Methods	Description
close	
flush	
getAONSBuffer	Gets the AON buffer.
write	Writes to the buffer output stream.
write	Writes to the buffer array.
write	Writes to the buffer array.
write	Writes to the buffer array.
write	Writes to the buffer array.
write	Writes to the buffer array.
write	Writes to the buffer array.

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

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.AONException) 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.AONException) inherits methods from class.java.lang.Throwable and class.java.lang. Object. For more information, see “InitializationException” in “Appendix A. AONS Common Specification.”

Utilities Pool Package

The AON Utilities Pool package (com.cisco.aons.util.pool) has the single class described in the next section.

Class

The Utilities Pool Package class is described below.

PooledJob

The PooledJob fields and methods are summarized below.

Fields	Description
mNext	`public PooledJob mNext`
mPrevious	`public PooledJob mPrevious`

Methods	Description
execute This method is invoked when a pooled job becomes runnable (pulled from the queue for execution).	public void **execute**()
Methods inherited from class java.lang.Object	clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

External Services

Each AON service operates through a set of APIs. Customers and partners can expand existing AON service functions by creating custom bladelets that operate in a message Policy Execution Plan (PEP). The External Services Extension provides a set of APIs that can be incorporated into custom bladelets and adapters.

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

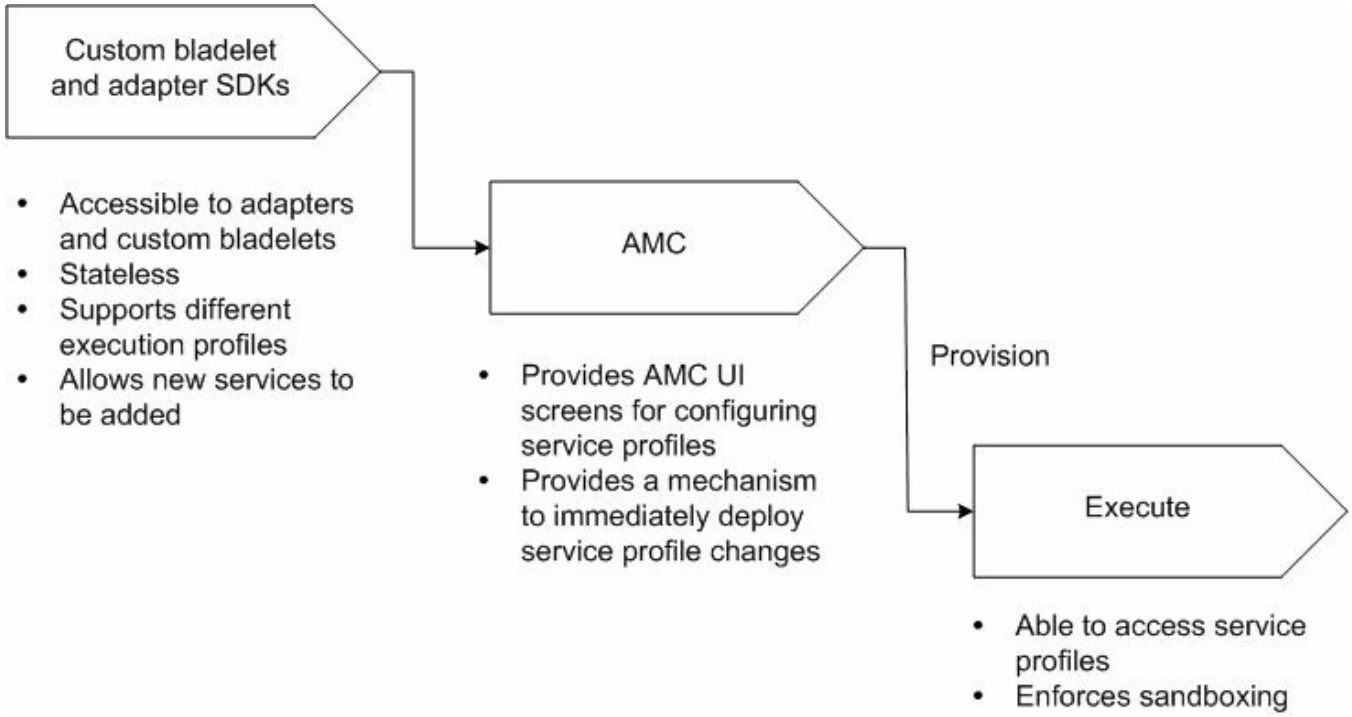
This chapter describes the External Services API, focusing on:

For background information, see Chapter 2, "Custom Bladelets" and Chapter 3, "Custom Adapters" in this guide, the *AON Administration and Installation 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 4-1 External Services API Life Cycle



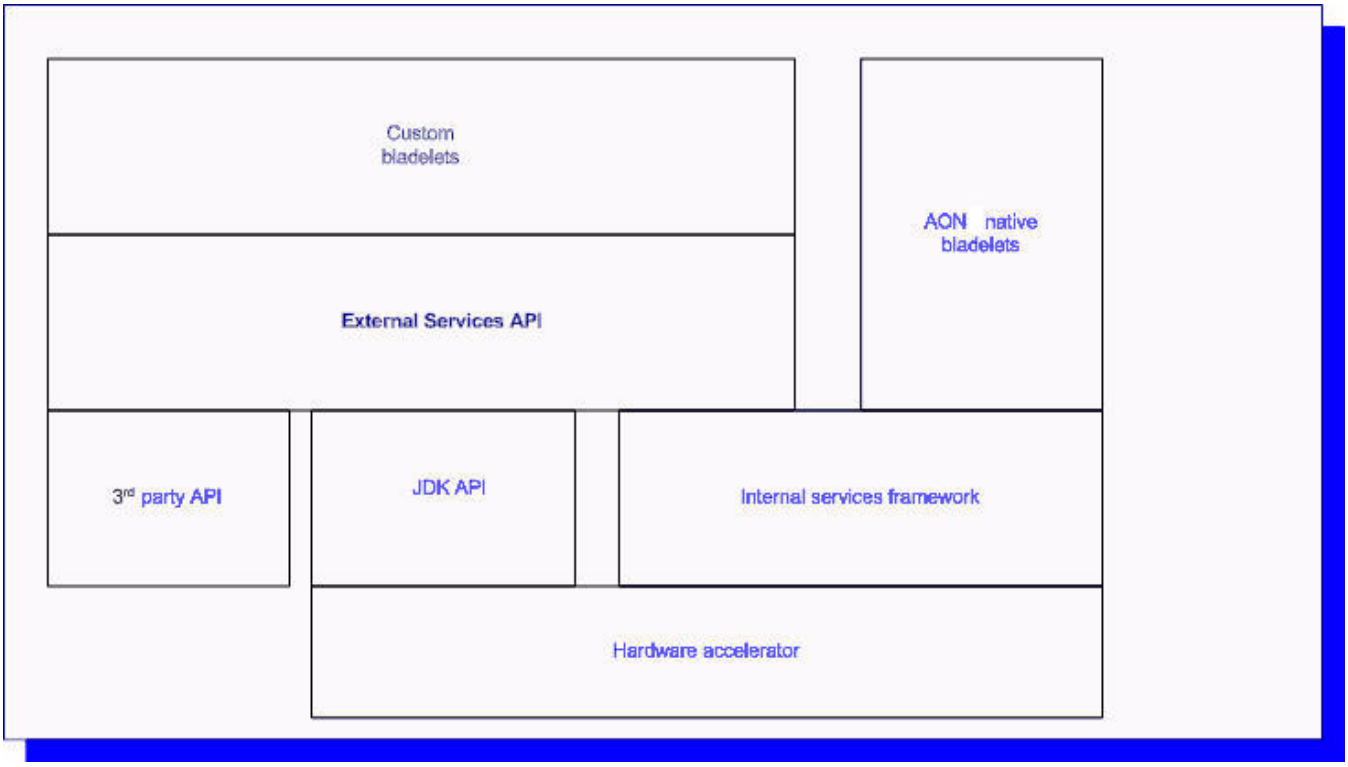
A service profile is a set of attributes that describe the service. Each service has one profile. A profile contains multiple named contexts. You use AMC screens to create named contexts.

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 4-2 External Services Architecture



Developing Interface Extensions

You can extend this interface by developing new transformer and parser plugin extensions. These procedures are summarized in the following sections:

- Creating a Transformer Extension, page 4-4n
- Creating a Parser Plugin Extension, page 4-5
- Packaging and Uploading Extension Files, page 4-5

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 Create a Content Parser package.

Use AON Development Studio (ADS) screens for this step. See "Content Parser Packages, page 5-7.

Step 3 Upload and Register the Content Parser Package.

Use AON Management Console (AMC) screens for this step.

Step 4 Define a Content Parser policy that specifies the class of AONSTransformerFactory defined in step 1.

Use the AMC Policies screens to perform this task.

Step 5 Define a PEP that uses the Transform bladelet and the Content Parser policy defined above.

Note For packaging and uploading steps. see the "Packaging and Uploading Extension Files" section on page 4-5.

For more information, see "Creating a Parser Plugin Extension" section on page 4-5, *AON Administration and Installation Guid*e, and the *AON Development Studio Guide*.

Creating a Parser Plugin Extension

You may have to create a parser plug-in 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. Follow the steps listed below.

Step 1 Define a class that implements `XMLReader.`

Step 2 Create a content parser package.

See "Content Parser Package" in Chapter 5.

Step 3 Upload and register the content parser package/

Use AMC screens 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 screens for this task.

Step 5 Define a PEP that uses Transform bladelet and uses the content parser policy defined above.

Use the ADS PEP Developer for this task.

Note For packaging and uploading steps. see the "Packaging and Uploading Extension Files" section on page 4-5.

For more information, see "Creating a Transformer Extension" section on page 4-4 and the *AON Administration and Installation Guid*e and the *AON Development Studio Guide*.

Packaging and Uploading Extension Files

After you create an extension, you use the ADS to package and upload it to AMC. These tasks are identical to the packaging and uploading activities that you perform for custom adapters, except that you select extension files. For directions see "Packaging the Custom Adapter, page 3-33" in Chapter 3, Custom Adapters. For additional information, see the description of packaging in the *AON Development Studio 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.

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, java.lang.String stylesheetProfileName, javax.xml.transform.Source src, javax.xml.transform.Result target, javax.xml.transform.Templates xsltTemplate, org.xml.sax.XMLReader xmlReader) throws ExtServiceException **Parameters:** pluginProfileName—Specifies the parser plugin profile to be used. stylesheetProfileName—Specifies the transform policy profile to be used. src—Input data. target—Contains transformation results. xsltTemplate—Template object corresponding to the style sheet specified in the stylesheetProfileName property set. xmlReader —XML Reader obtained by instantiating the parser class specified in the pluginProfileName property. 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 Interface Extensions" section on page 4-4.

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**() throws ExtServiceException **Returns:** Transformer. Creates a transformer object.

For additional information, see "Developing Interface Extensions" section on page 4-4 and "Creating a Parser Plugin Extension" section on page 4-5.

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, java.lang.String password, java.lang.String LDAPProfile) throws com.cisco.aons.exception.ExtServiceException 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.

The object cache is shared across PEPs and PEP instances. It is not a distributed cache. 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, java.lang.Object value, java.lang.Long timeoutInSeconds) throws ExtServiceException **Parameters:** 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. value—Objects are stored in a local variable cache in memory. If a distributed cache is exposed, the object must be serializable. timeoutInSeconds—Timeout value **Returns:** True if the operation is successful, otherwise false Stores the object in the cache.
get	public java.lang.Object **get** (java.lang.Object key) throws ExtServiceException **Parameters:** key—Serializable object used in the put operation It is unique across PEPs, instances of PEPs, and messages. **Returns:** Cached object Retrieves the object from the cache
remove	public boolean **remove** (java.lang.Object key) throws ExtServiceException **Parameters:** key—Serializable object used in the put operation It is unique across PEPs, instances of PEPs, and messages. **Returns:** True if the object is removed successfully from cache, otherwise false 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) throws ExtServiceException **Parameters:** inputData —Content data to be compressed **Returns:** byte[] compressed output data as byte array Compresses input data and generates the output data.
decompress	public com.cisco.aons.message.IContent **decompress** (byte[] inputData, int contentType) throws ExtServiceException **Parameters:** inputData —Byte array data to be decompressed contentType—Should be one of the content types defined ex: XML_CONTENT_TYPE **Returns:** Content decompressed output content 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, java.util.HashMap exps) throws ExtServiceException **Parameters:** input—Content to be evaluated 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. **Returns:** 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. 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, java.util.HashMap exps, ExtService Context context) throws ExtServiceException Regular expression lookup content. **Parameters:** input—Content to be evaluated exps— Hashmap contains key and value. For regular expressions, the resulting value is an expression. For XPath expressions, the output is an expression. context—Input configurable parameters such as lookup type. **Returns:** 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) throws ExtServiceException **Parameters:** content—Input content to be validated **Returns:** True if successfully validates the input stream, otherwise false 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, ExtServiceContext context) throws ExtServiceException **Parameters:** content—Input content to be validated. context—ExtService Context to pass information such as type of validation (xsd,schema). **Returns:** True if successfully validates the input stream, otherwise false Validates the input content with the schema-DTD.

For more information, see Chapter 6, "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, java.lang.String dataToEncrypt) throws ExtServiceException **Parameters:** content—Input/output content dataToEncrypt—Data to encrypt **Returns:** True if encryption is successful otherwise false 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, java.lang.String dataToEncrypt, ExtServiceContext context) throws ExtServiceException **Parameters:** content—Input/output content dataToEncrypt—Data to encrypt context—Service Context parameters **Returns:** True if encryption is successful otherwise false Encrypts the given input content and produces the output content

Method	Description
decrypt	public int **decrypt**(com.cisco.aons.message.IContent content, java.lang.String elementsToDecrypt) throws ExtServiceException **Parameters:** content—Input/output content elementsToDecrypt—Elements to be decrypted **Returns:** 0—If successful -1—If decryption element not found -2—If decryption key not found 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, java.lang.String elementsToDecrypt, ExtServiceContext context) throws ExtServiceException **Parameters:** content—input/output content elementsToDecrypt—Elements to be decrypted **Returns:** 0—If successful -1—If decryption element not found -2—If decryption key not found) 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**() **Returns:** String name of the service
geProfile	public ExtServiceProfile **getProfile**() **Returns:** 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) **Parameters:** attrName—Input attribute name defined in the context **Returns:** List of all attribute values for the attribute name.
getAttributes	public java.util.Map **getAttributes**() **Returns:** Map Key is attribute name, Value is List of values Returns all information defined in the context.
setAttributes	public void **setAttributes**(java.util.Map attrMap)
getName	public java.lang.String **getName**() **Returns:** 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) **Parameters:** contextName—Input context name **Returns:** Service context defined in the profile.
getContexts	public java.util.List **getContexts**() **Returns:** Contexts defined in the profile.
getCurrentContext	public ExtServiceContext **getCurrentContext**() **Returns:** Current context set in the service. The context must be defined already in profile.

Method	Description
setCurrentContext	public void **setCurrentContext** (java.lang.String contextName) throws ExtServiceException **Parameters**: contextName - Name of the context. Sets the current context.
setCurrentContext	public void **setCurrentContext** (java.lang.String contextName, java.util.HashMap attrMap) throws ExtServiceException **Parameters**: contextName— Name to be set for the current context. attrMap—Contains: name—attribute name value—list of string values for the attribute. Sets the current context.
getDefaultContext	public ExtServiceContext **getDefaultContext**() **Returns:** defaultContext ExtServiceContext type Returns the default context from the profile (name as “default”).
getName	public java.lang.String **getName**() **Returns:** Name (string) of the profile.
getServiceName	public java.lang.String **getServiceName**() **Returns:** 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**

Fields	Description
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) **Parameters:** input—MIME input content newContent—Content to be added **Returns:** output MIME content Adds new content
delete	**delete** (IMIMEContent input, IContent deleteContent) **Parameters:** input—MIME input content. deleteContent—Content to be deleted. **Throws:** ExtServiceException **Returns:** output MIME content Deletes the (part) content in the given MIME content
delete	**delete** (MIMEContent input, int index) **Parameters:** input—MIME input content. index—At which content to be deleted. **Throws:** ExtServiceException **Returns:** output MIME content Deletes the (part) content in the given MIME content

Method	Description
extract	extract (IMIMEContent content, java.lang.String selector) **Parameters:** content—Input MIME input content. selector—format is field==value **Throws:** ExtServiceException **Returns:** list of the parts for the given selector Extracts the given fields from the MIME Content
insert	**insert** (IMIMEContent input, IContent newContent, int index) **Parameters:** input—MIME input content newContent—Content to be inserted index—Index in which it has to be inserted **Throws:** ExtServiceException **Returns:** output MIME content Inserts new context at the index.

Method	Description
replace	**replace** (IMIMEContent input, IContent newContent) **Parameters:** input—MIME input content. newContent—Content to be inserted. **Throws:** ExtServiceException **Returns:** output MIME content Replaces the input content with the new one
replace	**replace** (.IMIMEContent input, IContent newContent, int index) **Parameters:** input—MIME input content. newContent—Content to be inserted. index—At which the content has to be replaced. **Throws:** ExtServiceException **Returns:** output MIME content 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) throws ExtServiceException **Returns:** String name of the service 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 **Parameters:** content—Input to be signed nodeToSign—Node to be signed ExtServiceContext—Input context. **Returns:** True if successful. Otherwise, returns false. Signs an input XML document
verify	boolean verify (IContent content, ExtServiceContext serviceContext) throws ExtServiceException **Parameters:** content—Input to be verified serviceContext—Input service context **Returns:** True if successful. Otherwise, returns false. 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, java.lang.String stylesheetProfileName, javax.xml.transform.Source src, javax.xml.transform.Result target) throws ExtServiceException This function performs the transformation. It applies the XSLT transform on a specified src document and produces a result document. **Parameters:** 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. 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. src—Specifies the input source document. target—Result of the transformation. **Throws:** ExtServiceException 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-8.

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

Note Although ExtService Exception 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.

Transformation

The XSLT Transformation (XSLT) service determines how an input message is transformed to produce the output message. The service ca n take an incoming XML or non-XML message and generate an XML or non-XML message.

This section explains transformation services, focusing on:

Note See External Services API Specification, page 4-6 for descriptions of the key interfaces AONSTransformer and AONSTransformerFactory.

Preliminary Activities

Before you perform the activities summarized in this chapter, you must:

- Create an Extensible Stylesheet Language (XSL) transformation file with any XSL editor. A sample file is shown in the "Friends.xsl" section on page 5-10.
- Use ADS to create transform and content parser packages. A transform package requires one or more XSLT files and the "Transform Extension Information File" section on page 5-6 (transform-info.xml).
- Use the AMC to register the packages so that they can be deployed on AON nodes.
- Use AMC property management screens to create transformation and content parser properties. These can be created at a node or global level.
- Use ADS to create transform PEP and associated message classification properties.

For details, see the *AON Administration and Installation Guide* and the *AON Development Studio Guide*.

Transforming Messages

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 Transforming a Message, page 5-2 and Transformation Examples, page 5-3.

Transforming a Message

Follow the steps listed below to transform a message. For more information, see Transformation Examples, page 5-3 and the description of Transform bladelet in the *AON Development Studio Guide*.

Step 1 Create a transform bundle containing the XSLT or Java plugin.

Using ADS, package the adapter files.

Step 2 Upload the bundle to AMC.

Using AMC, select **Admin >Extensions > Transform Package > Upload**.

Step 3 Create policy files for the transformation.

Using AMC, select **Properties > Application > Global > Transformation**.

Step 4 Synchronize ADS with AMC.

Using ADS, click **Sync**.

This action makes the newly created policy file available for use.

Step 5 Create a PEP.

Using ADS, create a new PEP and move a Transform bladelet to the displayed PEP.

Step 6 Upload the PEP to AMC.

Using ADS, click **Sync**.

Step 7 Deploy the PEP to AON.

Using AMC, select **Deploy**.

Step 8 Send the XML file to the end server.

See the samples below.

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 5-3
- Java Plugin Transformation Example, page 5-4

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 5-10
- XSLT file = Friends.xsl, page 5-10
- 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 5-1

Step 1) Create the Transform bundle and save it.

- Using ADS, create a Transform bundle including friends.xsl file in the bundle.
- Save this file on the local hard disk as transform.xfn.

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) 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) Download the new policy files from AMC.

- Using ADS, click **Sync** to receive the newly created policy files from AMC.

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) 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) Deploy the file to AON.

Using AMC, select **Deploy**.

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 PEP), 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 under including 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 transformation 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 screens 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 5-1.

Table 5-1 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 4-4 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 5-2.

Table 5-2 Content Parser Package File Sections

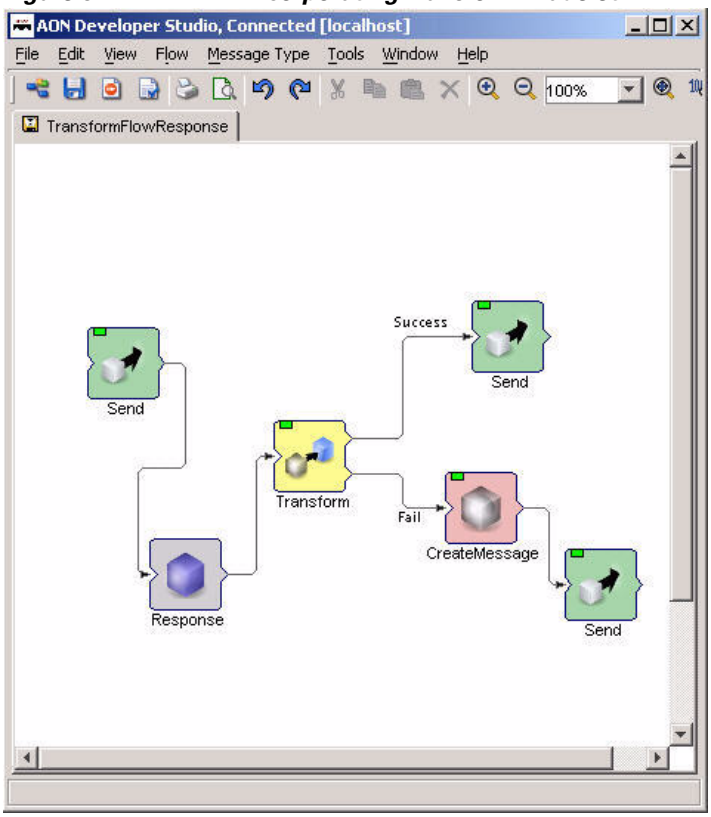
File Section	Description
<TransformExtensionInfo> ... </>	This is the root element in the transform-info.xml file. All extensions are children of this section. It includes one or more ParserPluginInfo sections.
<ParserPluginInfo> ... </>	This section specifies a Java class name that implements a Parser 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. In the sample above, a Content Parser class sample.csvplugin.CSVfriendsReader implements XMLReaser class while sample.plugin.TransformerFactory implements AONSTransformerFactory. 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 Parser Plugin Extension, page 4-5 the description of packaging in the *AON Development Studio Guide*.

Using XSLT Transformation

To use XSLT transformation, you design a message PEP that uses various AON bladelets (including transformation) to extract data, transform it, and compose the output message with transformed content. Figure 5-1 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 5-1 PEP Incorporating Transform Bladelet



As this figure indicates, the message processing PEP has the following steps:

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" section on page 5-10) 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 a friends XML document to HTML.

```
<?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>
```

APIs

The key APIs, AONSTransformer and AONSTransformerFactory are summarized below. For additional information, see individual bladelet descriptions in the *AON Development 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 4-7.

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 4-8.

Schema Validation

The AON Schema Validation service provides message content validation. This chapter introduces the service in the following sections:

For more information, see the description ContentValidation, page 4-12.

Prerequisites

Before incorporating schema validation into a message PEP, you must:

- Use AON Development Studio (ADS) screens 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 Administration and Installation Guide* and the *AON Development Studio Guide*.

Content Validation

You include the Content Validation 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. Your message PEP can be designed to reject or drop any message that does not conform to the given schema.

AON uses the Apache Xerces validating Parser 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. Due to this, the grammar files do not have to be parsed each time for every incoming message that has 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 Content Validation 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 following parameters can be configured for Schema Validation.

- Content Carrying Signature – XML Content to be validated against a schema or DTD.
- 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. At least one of these checkboxes needs to be enabled, indicating that the user wants either XSD validation or DTD validation. By default, both forms of validation are enabled.
- Xerces Validation Features – The following are optional validation features that can be enabled or disabled for advanced usage. All these features are disabled by default.
  - Ignoring Namespaces – This feature causes the parser to ignore namespaces in the XML instance document being validated. If the schema file refers to a target namespace and corresponding element prefixes, these will be ignored.
  - Full Schema Validation – This feature enables Full Schema Validation. This validation checks the schema grammar itself 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
  - Validation for messages with no grammar declaration – AON can validate XML messages against a schema that is not declared within the document itself. If this feature is enabled, then the Policy reference for the Schema to be imposed needs to be specified. AON can also validate a subset of the incoming XML message against a specified schema reference. The user has the option of specifying an XPath expression indicating that only relevant element(s) of the document at the XPath need to be verified against the schema reference provided. The XPath expression parameter is optional, implying that by default, the entire document is validated against the schema provided.

For example, if an incoming message is a large XML document containing multiple purchase orders and other information, and the user is only interested in verifying that each PurchaseOrder element is valid and conforms to a PurchaseOrder schema. In this case, the user can specify an XPath expression (such as //PurchaseOrder) and provide a schema reference (such as PurchaseOrder.xsd policy) to enforce the schema on all the PurchaseOrder elements found in the message. For more information, see ContentValidation, page 4-12 and the Content Validation bladelet description in the *AON Development Studio Guide*.

Schema Package Content

Schema packages include a manifest to define package name, version, vendor name, description and those packages that have one or more.xsd or.dtd files. The package also has a schemas-info.xml file that defines metadata for the schema extensions. A sample is shown below.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<SchemaExtensionInfo xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" >
<XSDSchemaType>
    <XSDSchemaInfo name="books.xsd" display-name="books.xsd"
version="1.0"></XSDSchemaInfo>
    <XSDSchemaInfo name="NGD100k.xsd" display-name="NGD100k.xsd" version="1.0"/>
```

```
    <XSDSchemaInfo name="SecurityPolicy.xsd" display-name="SecurityPolicy.xsd"
version="1.1"/>
    <XSDSchemaInfo name="quotations.xsd" display-name="quotations.xsd" version="1.0"/>
</XSDSchemaType>
<DTDSchemaType>
    <DTDSchemaInfo name="books.dtd" display-name="books.dtd" version="1.1">
</DTDSchemaInfo>
</DTDSchemaType>
<AONVersion version="1.0"/>
</SchemaExtensionInfo>
```

AONSCommon Specification

AONSCommon packages are used in conjunction with Custom Bladelet, Custom Adapter, External Service, XSLT Transformation and Schema Validation software development kits (SDKs) It includes the interfaces described below.

API Summary

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 components are described in the following sections.

Classes

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 Application type.
Recoverable	public static final ExceptionType RECOVERABLE Recoverable type.
System	public static final ExceptionType SYSTEM System type.

Method	Description
getType()	public abstract int getType() Returns the type of exception: application, recoverable, or system types.

Exceptions

The following AONSCommon exceptions are used:

- AONSException, page A-3
- AONSRuntimeException, page A-8
- ExtServiceException, page A-8
- InitializationException, page A-10
- NamedExtenstionException, page A-9
- NoSuchVariable, page A-12

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) Parameters: pErrorCode—String Returns: An AONSException based on the error code.
create	public static final AONSException create(java.lang.String pErrorCode, Exception pEnum) Parameters: pErrorCode—String pEnum—ExceptionType Returns an AONSException based on the error code and type
getArgs	public final java.lang.Object[] getArgs() Gets object arguments.
getErrorCode	public int getErrorCode() Gets the error code.
getErrorResourceKey	public final java.lang.String getErrorResourceKey() Returns an error key used to lookup aonsErrors.properties for the error message.

Method	Description
getErrorResourceKey	public static final java.lang.String getErrorResourceKey(java.lang.String pPrefix, int pErrorCode) Parameters: pPrefix—gives the module of the exception pErrorCode—error code of the exception Throws: java.util.MissingResourceException Returns an error key consisting of the parameters that are used to lookup aonsErrors.properties.
getExceptionEnum	public final ExceptionType getExceptionEnum() Gets the exception number.
getPrefix	protected java.lang.String getPrefix() Returns: String containing a three letter acronym that identifies the component or module that threw the exception. It defaults to a system acronym.
getReplyMessage	public java.lang.Object getReplyMessage() Returns the reply message set by the throwing class. Returns the reply message object it is set by the throwing class. Otherwise, returns null.
setReplyMessage	public void setReplyMessage(java.lang.Object pMsg) Parameters: pMsg—Reply message object Sets the reply message to pMsg.
getResourceBundle()	public static final java.util.ResourceBundle getResourceBundle(java.util.Locale pLocale) throws java.util.MissingResourceException Parameters: pLocale Gets a resource bundle.
getResourceBundle()	public static final java.util.ResourceBundle getResourceBundle(java.util.Locale pLocale, java.lang.ClassLoader pClassLoader) throws java.util.MissingResourceException Gets a resource bundle.

Method	Description
getResourceMessage()	public final java.lang.String getResourceMessage() throws java.util.MissingResourceException 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) throws java.util.MissingResourceException Parameters: pClassLoader—ClassLoader Throws MissingResourceException if no error number is specified. Returns (string) the resource tokens (name-value pairs) associated with the error number.
getResourceMessage()	public final java.lang.String getResourceMessage(java.lang.String pQualifier) throws java.util.MissingResourceException Parameters: pQualifier pQualifier—Further qualifies the resource bundle 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, java.lang.ClassLoader pClassLoader, java.lang.String pResourceKey) throws java.util.MissingResourceException Parameters: pQualifier—String pClassLoader—ClassLoader Returns: String containing the resource tokens (name-value pairs) associated with the error number or exception message if there is no error number specified.

Method	Description
getResourceMessage	public static final java.lang.String getResourceMessage(java.lang.String pPrefix, int pErrorCode) throws java.util.MissingResourceException Parameters: pErrorCode
getResourceMessage	public static final java.lang.String getResourceMessage(java.lang.String pPrefix, int pErrorCode, java.lang.ClassLoader pClassLoader) throws java.util.MissingResourceException
getResourceMessage	public static final java.lang.String getResourceMessage(java.lang.String pPrefix, int pErrorCode, java.util.Locale pLocale) throws java.util.MissingResourceException Parameters: pErrorCode pLocale

Method	Description
getResourceMessage	public static final java.lang.String getResourceMessage(java.lang.String pPrefix, int pErrorCode, java.util.Locale pLocale, java.lang.ClassLoader pClassLoader) throws java.util.MissingResourceException Parameters: pErrorCode pLocale
getResourceMessage()	public final java.lang.String getResourceMessage(java.lang.String pQualifier, java.lang.ClassLoader pClassLoader) throws java.util.MissingResourceException Parameters: pQualifier—String pClassLoader—ClassLoader 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 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() Gets the exception ID.
isRecoverable	public boolean isRecoverable() Indicates whether the exception is recoverable or not.
setExceptionID	public void setExceptionID(java.lang.String pExId) Sets the exception ID.
setRecoverable	public void setRecoverable(boolean pRecover) 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, “External Services”.

PEP Package

The AON PEP package (com.cisco.aons.pep) contains one interface class: PEPData.

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() Returns the Id (string) associated with this PEP.
getName()	public java.lang.String getName() Returns the PEP name as a string.
getTime()	public long getTime() Returns the PEP time associated with the PEP time.
getVariable()	public java.lang.Object getVariable(java.lang.String pName) throws NoSuchVariable Parameters: pName—Input name. Returns the value associated with the Variable.
getMsgTypeName()	public java.lang.String getMsgTypeName() Returns the message classifier type.

Method	Description
setVariable	public java.lang.Object setVariable(java.lang.String pName, java.lang.Object pValue) throws NoSuchVariable Sets the Variable value.
setVariable	public java.lang.Object setVariable(java.lang.String pName, java.lang.Object pValue, boolean pCreate) throws NoSuchVariable Parameters: pName—input name pValue—input object value 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. 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.

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-17
- Classes, page A-43
- Exceptions, page A-45

Interfaces

The message package includes the interfaces listed below.

- IAONSMessage, page A-18
- ICloseable, page A-19
- IContent, page A-20t
- IContentAttachment, page A-21
- IContentVisitable, page A-22
- IContentVisitor, page A-22
- IContextSerializable, page A-24
- IDeliveryContext, page A-24
- IEncodingConstants, page A-25
- IMapContent, page A-26
- IMessageBuilder, page A-28r
- IMessageConstants, page A-32
- IMessageContext, page A-32
- IMessageDeliveryContext, page A-34
- IMessageHeaders, page A-35
- IMIMEContent, page A-38
- IMsgAttachment, page A-40
- INullContent, page A-40
- IRNContent, page A-40
- ISOAPContent, page A-41t
- IStreamContent, page A-41
- IXMLContent, page A-41

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() Returns the message content in a canonical container, which can be any subtype of IContent. For more information, see IMapContent, IMIMEContent, INullContent, ISOAPContent, IStreamContent, and IXMLContent.
getMessageBuilder()	public IMessageBuilder getMessageBuilder() Returns IMessageBuilder, the message builder associated with this AON message.It can be used to create a new AON message.
getMessageContext()	public IMessageContextgetMessageContext() Returns the context that contains message metadata.
getMessageContextID()	public java.lang.String getMessageContextId() Returns the context ID for this message.
getMessageHeaders()	public IMessageHeadersgetMessageHeaders() Returns the message headers of a message.
getMessageId()	public java.lang.String getMessageId() Returns the message ID associated with this message.
getMessageTimestamp()	public long getMessageTimestamp() Returns the long date and time when the message arrived in milliseconds.
getMessageType()	public int getMessageType() 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() Returns the AON cloud node that sent the message. 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() 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() Returns the SSL certificate.

Method	Description
isApplicationReply()	public boolean isApplicationReply() Returns “true” (boolean) if the message is an application reply message, responding to a request message.
IsApplicationRequest()	public boolean isApplicationRequest() 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) 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.

This interface should be implemented, if the normal cleanup process is not efficient enough to reclaim the resources attached to the particular object. Typically, these are external resources such as connections, open files, and direct byte buffers. Classes that implement this interface should use the close() method to release these resources For additional information, see IAONSMessage.

The single ICloseable method is summarized below.

Method	Description
close()	public void close() throws java.lang.Exception Releases resources. The ICloseable implementation should be able to handle where close() is called many times on the same object.

IContent

Extending IContentVisitable, this is the top level interface for all content types. The following subinterfaces are based on this interface: IMapContent, IMIMEContent, INullContent, ISOAPContent, IStreamContent, and IXMLContent. IContent includes the methods and fields summarized below.

Method	Description
addHeader	public void addHeader(java.lang.String name, java.lang.String value) Adds a header with the given value. This value may duplicate an already existing value.
getALLHeaders()	public java.util.Iterator getAllHeaders() Returns an iterator over all the headers (items of the type javax.xml.soap.MimeHeader).
getContentAttachment()	public IContentAttachmentgetContentAttachment() Returns the content attachment.
getContentType()	public int getContentType() Returns the content type (int).
getInputStream()	public java.io.InputStream getInputStream() throws java.io.IOException Returns an input stream on the content. This stream must be marked before use and reset after use so that it can be reused. Throws an IOException if the method does not return an input stream.
getMatchingHeaders	public java.util.Iterator getMatchingHeaders(java.lang.String hdrName) Parameters: hdrName—Name of header to match Returns an iterator over a collection of headers that match the given header.
getParent()	public IContent getParent() Returns the parent of the current content or null.
IsInputStreamAvailable()	public boolean isInputStreamAvailable() Returns true if the input stream is available, otherwise false.
removeHeader	public void removeHeader(java.lang.String name) Parameters: name—Name of the header to be removed Removes headers with the given name.

Method	Description
setContentAttachment	public void setContentAttachment (IContentAttachment pAttachment) Parameters: pAttachment—IContentAttachment Sets the attachment with the content.
setHeader	public void setHeader(java.lang.String name, java.lang.String value) Parameters: name—Header name value—Header value Replaces the given header with a new value or adds a header.

Field	Description
MAP_CONTENT	public static final int MAP_CONTENT Map content.
MIME_CONTENT	public static final int MIME_CONTENT MIME content.
NULL_CONTENT	public static final int NULL_CONTENT Null content.
SOADP_CONTENT	public static final int SOAP_CONTENT SOAP content.
STREAM_CONTENT	public static final int STREAM_CONTENT Stream content.
XML_CONTENT	public static final int XML_CONTENT XML content.

IContentAttachment

This is content attachment interface. It the single method summarized below.

Method	Description
close()	public void close() Releases any resources held by the attachment.

For more information, see "IContent" section on page A-20.

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) throws AONSException Parameters: pVisitor—IContentVisitor the content visitor 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) throws AONSException Parameters: pContent—IMapContent, the map content Visits a map content. Throws AONSException if unable to visit a map content.
visit MIME Content	public void visitMIMEContent(IMIMEContent pContent) throws AONSException Parameters: pContent—IMIMEContent the MIME content Visits a MIME content. Throws AONSException if unable to visit a MIME content.

Method	Description
visitNullContent	public void visitNullContent(INullContent pContent) throws AONSException Parameters: pContent—INullContent, the null content Visits a null content. Throws AONSException if unable to visit a null content.
visitSOAPContent	public void visitSOAPContent(ISOAPContent pContent) throws AONSException Parameters: pContent—IMapContent, the SOAP content Visits a SOAP content. Throws AONSException if unable to visit a SOAP content.
visitStreamContent	public void visitStreamContent(IStreamContent pContent) throws AONSException Parameters: pContent—IStreamContent, the stream content Visits a stream content. Throws AONException if unable to visit the stream content.
visitXMLContent	public void visitXMLContent(IXMLContent pContent) throws AONSException Parameters: pContent—IXMLContent, the XML content 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) throws java.io.IOException Parameters: pOS—OutputStream Serializes the message context to the output stream
serialize	public void serialize(java.io.OutputStream pOS) throws java.io.IOException Parameters: pIS—InputStream 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() Returns the endpoint message ID (if one exists) for this message.
getMessageContextId	public java.lang.String getMessageContextId() Returns the message ID context for this message.
getMessageId	public java.lang.String getMessageId() 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 Indicates that the content is chunked.
CONTENT_ENCODING	public static final int CONTENT_ENCODING Indicates content encoding.
GZIP	public static final int GZIP Indicates that the content is compressed.
TRANSFER_ENCODING	public static final int TRANSFER_ENCODING 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) Parameters: name—String, the name. 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. 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). If you use this method with a multivalued parameter, the returned value is equal to the first value in the array returned by getParameterValues. 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() 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() 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) Parameters: name—String 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, java.lang.String value) Parameters: name—String value—String Adds a single name-value pair to the map content.
setParameter	public void setParameter(java.lang.String name, java.lang.String[] values) Parameters: name—String values—String[] 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) throws MessageBuildException Parameters: pMessage—IAONSMessage 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, IMessageContext pContext, IMessageHeaders pHeaders, IContent pContent, java.lang.String pMsgContextId) Parameters: 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 pContext—IMessageContext pHeaders—IMessageHeaders pContent—IContent pMsgContextId—String the message context id. Used for correlation. Returns IAONSMessage, creating a new AON message using supplied values. For more information, see IMessageConstants.

Method	Description
createAONSMessage	public IAONSMessage createAONSMessage(int mMsgType, IMessageContext pContext, IMessageHeaders pHeaders, IContent pContent) Parameters: 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 pContext—IMessageContext pHeaders—IMessageHeaders pContent - IContent Returns IAONSMessage, creating a new AON message using the supplied values. For more information, see IMessageConstants.
createAONSMessage	public IAONSMessage createAONSMessage(int mMsgType, IMessageContext pContext, IMessageHeaders pHeaders, java.lang.String pMsgContextId) Parameters: 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 pContext—IMessageContext pHeaders—IMessageHeaders pMsgContextId—String Returns IAONSMessage, creating a new AON message using the supplied values. For more information, see IMessageConstants.
createAONSMessage	public IAONSMessage createAONSMessage(int mMsgType, IMessageContext pContext, IMessageHeaders pHeaders) Parameters: 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 pContext—IMessageContext pHeaders—IMessageHeaders Returns IAONSMessage, creating a new AON message using supplied values. For more information, see IMessageConstants.

Method	Description
createDeliveryContext	public IDeliveryContextcreateDeliveryContext() throws MessageBuildException Creates an empty AON message delivery context.
createErrorAONSMessage	public IAONSMessage createErrorAONSMessage(IAONSMessage pMessage, AONSException pException) throws MessageBuildException Parameters: pMessage—IAONSMesspException - pException—AONSException Returns IAONSMessage, creating an error message. The method copies appropriate context and header information from the input AON message.
createMapContent	public IMapContent createMapContent() Returns IMapContent, creating map content.
createMapContent	publicIMapContent createMapContent(java.lang.String pContentType) Parameters: pContentType—String Creates map content of a specified type.
createMessageContext	publicIMessageContext createMessageContext() Returns IMessageContext, creating a new message context.
createMessageContext	public IMessageContext createMessageContext(int pRedirect) Parameters: pRedirect—int Returns IMessageContext, creating a new message context with a redirect directive.
createMessageHeaders	public IMessageHeaders createMessageHeaders() Returns IMessageHeaders, creating a new message header.
createMIMEContent	publicIMIMEContent createMIMEContent() Returns IMIMEContent, creating MIME content.
createMIMEContent	public IMIMEContent createMIMEContent(java.lang.String pContentType) Parameters: pContentType—String Returns IMIMEContent, creating MIME content of a specified type.

Method	Description
createNullContent	public INullContent createNullContent() Returns INullContent, creating null content.
createReplyAONSMessage	public IAONSMessage createReplyAONSMessage(IAONSMessage pMessage, IContent pContent) throws MessageBuildException Parameters: pMessage—original message content pContent—new content Returns IAONSMessage, creating an AON message from supplied content. It copies appropriate content and header information.
createReplyAONSMessage	public IAONSMessage createReplyAONSMessage(IAONSMessage pMessage, IMessageHeaders pHeaders, IContent pContent) throws MessageBuildException Parameters: pMessage—original message content pHeaders—IMessageHeaders, message headers of the returned AON message pContent—new content 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() Returns ISOAPContent, creating SOAP content.
createSOAPContent	public ISOAPContent createSOAPContent(java.lang.String pContentType) Parameters: pContentType—String Returns ISOPContent, creating SOAP content of a specified type.
createStreamContent	public IStreamContent createStreamContent() Returns IStreamContent, creating stream content.

Method	Description
createStreamContent	public IStreamContent createStreamContent(java.lang.String pContentType) Parameters: pContentType—String Returns IStreamContent, creating stream content of a specified type.
createXMLContent	public IXMLContent createXMLContent() Returns IXML, creating XML content.
createXMLContent	public IXMLContent createXMLContent(java.lang.String pContentType) Parameters: pContentType—String Returns IXMLContent, creating XML content of a specified type.
replaceContent	public void replaceContent(IAONSMessage pMessage, IContent pContent) throws MessageBuildException Parameters: pMessage—IAONSMessage pContent—IContent 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.
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() throws java.lang.CloneNotSupportedException All sub-types implement this method.
count	public int count() Returns the count of the delivery context headers.
fromMap	public void fromMap(java.util.Map pMap) Sets the delivery context using a map.
getContextType	public int getContextType() Returns the context type.
initialize	public void initialize(java.lang.String pKey, int pIndex) Parameters: pKey - String pIndex - int Initializes the context with type.
initialize	public void initialize(int pIndex) Parameters: pIndex - int Initializes the context with name and type.
toMap	public java.util.Map toMap() 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) Parameters: name - the name of the field value - the value of the field. 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, java.util.Date date) Parameters: name - the field name date - the field date value Adds the value of a date field.
addDataField	public void addDateField(java.lang.String name, long date) Parameters: name - the field name date - the field date value Adds the value of a date field.
clear	public void clear() Removes all headers.
clone	 Clones a message header.
containsKey	public boolean containsKey(java.lang.String name) Parameters: name - the field name Checks for the existence of a header field
ger	public java.lang.String get(java.lang.String name) Parameters: name - the case-insensitive field name Returns the value of the header or null if not found. For multiple fields of the same name, only the first is returned.

Method	Description
getDateField	public long getDateField(java.lang.String name) Parameters: name - String Returns the value of a date field, or "–1" if no date field is found.
getFieldNames	public java.util.Enumeration getFieldNames() 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) throws java.lang.NumberFormatException Parameters: name - String 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) throws java.lang.NumberFormatException Parameters: name - String 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() Returns an iterator for field name value pairs.
put	public void put(java.lang.String name, java.util.List list) Parameters: name - the name of the field list - the List value of the field. If null the field is cleared. Sets a field.
put	public java.lang.String put(java.lang.String name, java.lang.String value) Parameters: name - the name of the field value - the value of the field. If null the field is cleared. Sets a field.

Method	Description
putDateField	public void putDateField(java.lang.String name, java.util.Date date) Parameters: name - the field name date - the field date value Sets the value of a date field.
putDateField	public void putDateField(java.lang.String name, long date) Parameters: name - the field name date - the field date value Sets the value of a date field.
putIntField	public void putIntField(java.lang.String name, int value) Parameters: name - the field name value - the field integer value Sets the value of an integer field.
putLongField	public void putLongField(java.lang.String name, long value) Parameters: name - the field name value - the field long value Sets the value of a long field.
clone	public java.lang.Object **clone**() throws java.lang.CloneNotSupportedException Returns a clone object.

Method	Description
getValue	public java.util.Enumeration getValues(java.lang.String name, java.lang.String separators) Parameters: name - the case-insensitive field name separators - String of separators. 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) Parameters: name - the case-insensitive field name 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) Parameters: content—content to add Adds content to the end of a multipart MIME message.
addContentByIndex	public void addContentByIndex(IContent content, int idx) Parameters: content—content to add idx—index at which to add, other parts pushed back Adds content at a specified position.
addEnvelop	public void addEnvelope(int type) Parameters: type—type constant, e.g. IMIMEContent.ENV_RN11 Add an envelope around the MIME content

Method	Description
getContentByHeader	public IContent getContentByHeader(java.lang.String[] names, java.lang.String[] values) Parameters: names—array of header names values—corresponding array of values to match Gets content that matches all given names and values.
getContentByHeader	public IContent getContentByHeader(java.lang.String name, java.lang.String value) Parameters: name—header name value—header value Gets content with the given name and value in the header or null.
getContentById	public IContent getContentByID(java.lang.String id) Parameters: id—content ID to select Returns the IContent (part) corresponding to the ID.
getContentByIndex	public IContent getContentByIndex(int idx) Parameters: idx—part number to select Returns the IContent corresponding to the part number.
getCount	public int getCount() Returns the number of parts (count) of a multipart message.
getEnvType	public int getEnvType() Returns envelope type (default IMIMEContent.ENV_NONE)
getIndex	public int getIndex(IContent content) Parameters: content—content to match 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) Parameters: idx—index of part to remove Removes content with the same Content-ID as the specified content.

Method	Description
removeContent	public void removeContent (IContent content) Parameters: content—content with the Content-ID to remove Removes content at a specified index. The parts behind the point are pulled forward.
stripEnvelope	public void stripEnvelope() 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() Closes the message attachment.
getContext	public java.lang.Object getContext() 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() Returns the signature currently present in the envelope.

Method	Description
getVersion	public int getVersion() 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) 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() throws MessageParseException 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() throws java.io.IOException 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, java.nio.ByteBuffer toBuffer) throws AONSException Copies a message buffer.
getAsDocument	public org.w3c.dom.Document getAsDocument() throws MessageParseException Returns a W3C document.

Method	Description
getMsgAttachment	public IMsgAttachment getMsgAttachment(java.lang.String name) Returns an AON message attachment.
getRawStreamBuffer	public IRawStreamBuffer getRawStreamBuffer() throws AONSException Gets a raw stream buffer.
queryDocument	public org.w3c.dom.NodeList queryDocument(java.lang.String pXpath) throws MessageParseException Parameters: pXpath—String Evaluates the XPath expression on the document and returns a node list.
removeMsgAttachment	public void **removeMsgAttachment**(java.lang.String name) Removes an AON message attachment.
setMsgAttachment	public void **setMsgAttachment**(java.lang.String name, IMsgAttachment attachment) Sets an AON message attachment.
writeDocument	public void writeDocument(org.w3c.dom.Document pDocument) throws MessageParseException Parameters: pDocument—document Saves the XML document into this container.

Classes

The AONSCommon message package includes the classes listed below.

- AbstractContentVisitor, page A-43
- AbstractMessageDeliveryContext, page A-44
- MessageBuilderRegistry, page A-45

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) throws AONSException Parameters: pContent—IMapContent the map content Visits a map content.
visitMIMEContent	public void visitMIMEContent(IMIMEContent pContent) throws AONSException Parameters: pContent—IMIMEContent the MIME content Visits MIME content.
visitNullContent	public void visitNullContent(INullContent pContent) throws AONSException Parameters: pContent—INullContent the null content Visits null content.
visitSOAPContent	public void visitSOAPContent (ISOAPContent pContent) throws AONSException Parameters: pContent—IMapContent the SOAP content Visits SOAP content.

Method	Description
visitStreamContent	public void visitStreamContent(IStreamContent pContent) throws AONSException Parameters: pContent—IStreamContent the stream content Visits stream content.
visitXMLContent	public void visitXMLContent (IXMLContent pContent) throws AONSException Parameters: pContent—IXMLContent the XML content 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() throws java.lang.CloneNotSupportedException Clones this context.
count	public int count() Returns number of name tokens in this context.
getContextHeader	public final java.lang.String getContextHeader() Returns the context name.
getContextType	public final int getContextType() Returns the context type.
initialize	public final void initialize(int pCtxType) Parameters: pCtxType—int Initializes the message context with the context type.
initialize	public final void initialize(java.lang.String pHeader, int pCtxType) Parameters: pHeader—String pCtxType—int 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() Returns the single instance of the message builder registry.
getMessageBuilder	public IMessageBuilder getMessageBuilder(java.lang.String pProtocol) Returns the protocol specific message builder.
getMessageBuilder	public IMessageBuilder getMessageBuilder() Returns the default message builder.

Exceptions

The Message package uses the exceptions listed below.

- MessageBuildException, page A-45
- MessageException, page A-45
- MessageParseException, page A-46
- MessageWriteException, page A-46

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, printStackTrace, 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, printStackTrace, 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, printStackTrace, 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, printStackTrace, 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 classes discussed below.

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) Formats a date according to a stored formatter.
format	public java.lang.String format(long inDate) Formats a date according to a stored formatter.

Method	Description
format	public void format(long inDate, java.lang.StringBuffer buffer) Parameters: inDate - Date the format buffer - StringBuffer Formats to a string buffer.
getFormat	public java.text.SimpleDateFormat getFormat() Gets the format.
getFormatString	public java.lang.String getFormatString() Gets a format string.
getTimeZone	public java.util.TimeZone getTimeZone() Returns the time zone.
setTimeZone	public void setTimeZone(java.util.TimeZone tz) Parameters: tz—TimeZone Sets the time zone.
setTimeZoneID	public void setTimeZoneID(java.lang.String timeZoneId) 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, java.lang.Object item) Parameters: list—The list to add to or null if none yet created. item—The item to add. Adds an item to a LazyList.
add	public static LazyList add(LazyList list, java.util.Collection collection) Parameters: list—The list to add to or null if none yet created. Adds an item to a LazyList.
add	public static LazyList add(LazyList list, int initialSize, java.lang.Object item) Parameters: list—The list to add to or null if none yet created. initialSize—A size to use when creating the real list item—The item to add. 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, int i) Parameters: list—A LazyList returned from LazyList.add(Object) or null 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) Parameters: list—A LazyList returned from LazyList.add(Object). Returns real List of added items, which may be an EMPTY_LIST or a SingletonList.
getList	public static java.util.List getList (LazyList list, boolean nullForEmpty) Parameters: list—A LazyList returned from LazyList.add(Object) or null nullForEmpty—If true, null is returned instead of an empty list. 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, java.lang.Object o) Remove a LazyList.
size	public static int size(LazyList list) Parameters: list—A LazyList returned from LazyList.add(Object) or null Returns the size of a LazyList.
size	public int size() 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, java.lang.Object value) Parameters: name—entry key. value—entry value. 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, java.util.List values) Parameters: name—entry key. 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, java.lang.String[] values) Parameters: name—entry key. 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) Parameters: name—entry key. 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, int i) Parameters: name—entry key. i—index of the element to get. 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) Parameters: name—entry key. Gets multiple values. Single valued entries are converted to singleton lists.
put	public java.lang.Object put(java.lang.Object name, java.lang.Object value) Parameters: name—entry key. value—entry value. Returns the previous value or a null. Puts an entry into the map.
putAll	public void putAll(java.util.Map m) Parameters: m—Map Puts all contents of the map.
putValues	public java.lang.Object putValues(java.lang.Object name, java.util.List values) Parameters: name—entry key. Puts a multi valued entry.
putValues	public java.lang.Object putValues(java.lang.Object name, java.lang.String[] values) Parameters: name—entry key. Puts a multi valued entry.

Method	Description
removeValue	public boolean removeValue(java.lang.Object name, java.lang.Object value) Parameters: name—entry key. value—entry value. Returns “true” if the value is removed. Remove a values.
toStringArrayMap	public java.util.Map toStringArrayMap() 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, int offset, int length) Parameters: key—byte array containing the key. A simple ASCII byte to char mapping is used. offset—offset of the key within the array. length—length of the key 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, int length) Parameters: key—char array containing the key length—length of the key Gets a map entry by char array key.

Method	Description
getEntry	public java.util.Map.Entry getEntry(java.lang.CharSequence key, int pStart, int pEnd) Parameters: key—char array containing the key Gets a map entry by char array key.
getEntry	public java.util.Map.Entry getEntry(java.lang.CharSequence key) Parameters: key—char array containing the key Gets a map entry by char array key.
getEntry	public java.util.Map.Entry getEntry(char[] key, int offset, int length) Parameters: key—char array containing the key offset—offset of the key within the String. length—length of the key 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, int offset, int length) Parameters: key—String containing the key offset—Offset of the key within the String. length—length of the key 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, java.lang.Object value)
put	public java.lang.Object put(java.lang.String key, java.lang.Object value)
readExternal	public void readExternal(java.io.ObjectInput in) throws java.io.IOException, java.lang.ClassNotFoundException

Method	Description
getEntry	public java.util.Map.Entry getEntry(java.lang.CharSequence key, int pStart, int pEnd) Parameters: key—char array containing the key Gets a map entry by char array key.
getEntry	public java.util.Map.Entry getEntry(java.lang.CharSequence key) Parameters: key—char array containing the key Gets a map entry by char array key.
getEntry	public java.util.Map.Entry getEntry(char[] key, int offset, int length) Parameters: key—char array containing the key offset—offset of the key within the String. length—length of the key 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, int offset, int length) Parameters: key—String containing the key offset—Offset of the key within the String. length—length of the key 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, java.lang.Object value)
put	public java.lang.Object put(java.lang.String key, java.lang.Object value)
readExternal	public void readExternal(java.io.ObjectInput in) throws java.io.IOException, java.lang.ClassNotFoundException

Method	Description
getEntry	public java.util.Map.Entry getEntry(java.lang.CharSequence key, int pStart, int pEnd) Parameters: key—char array containing the key Gets a map entry by char array key.
getEntry	public java.util.Map.Entry getEntry(java.lang.CharSequence key) Parameters: key—char array containing the key Gets a map entry by char array key.
getEntry	public java.util.Map.Entry getEntry(char[] key, int offset, int length) Parameters: key—char array containing the key offset—offset of the key within the String. length—length of the key 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, int offset, int length) Parameters: key—String containing the key offset—Offset of the key within the String. length—length of the key 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, java.lang.Object value)
put	public java.lang.Object put(java.lang.String key, java.lang.Object value)
readExternal	public void readExternal(java.io.ObjectInput in) throws java.io.IOException, 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) Parameters: ic—If true, the map is case insensitive for keys. Sets the ignoreCase attribute.
setWidth	public void setWidth(int width) Parameters: width—Width of hash tables, larger values are faster but use more memory. Sets the hash width.
size	public int size()
writeExternal	public void writeExternal(java.io.ObjectOutput out) 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, java.lang.String s, int offset, int length) Parameters: buf - StringBuffer to append to s—string to append from offset—offset of the substring length—length of the substring Appends substring to StringBuffer
append	public static void append(java.lang.StringBuffer buf, byte b, int base)

Method	Description
asciiToLowerCase	public static java.lang.String asciiToLowerCase(java.lang.String s) Parameters: s—string to convert Performs a fast lower case conversion.Only works on ascii, not on unicode
endsWithIgnoreCase	public static boolean endsWithIgnoreCase(java.lang.String s, java.lang.String w)
equals	public static boolean equals(java.lang.String s, char[] buf, int offset, int length)
indexFrom	public static int indexFrom(java.lang.String s, java.lang.String chars) Returns the next index of a character from the chars string.
nonNull	public static java.lang.String nonNull(java.lang.String s) Parameters: s—string Returns a non null string.
replace	public static java.lang.String replace(java.lang.String s, java.lang.String sub, java.lang.String with) Replaces substrings within string.
startWithIgnoreCase	public static boolean startsWithIgnoreCase(java.lang.String s, java.lang.String w)
unquote	public static java.lang.String unquote(java.lang.String s) 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) Parameters: b—ascii encoded character 0-9 a-f A-F 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) Parameters: name—class or type name. Returns the class from a canonical name for a type.
newInteger	public static java.lang.Integer newInteger(int i) Converts an int to an integer using cache.
parseBytes	public static byte[] parseBytes(java.lang.String s, int base)
parseInt	public static int parseInt(java.lang.CharSequence s, int start, int end, int base) throws java.lang.NumberFormatException
parseInt	public static int parseInt(java.lang.CharSequence s, int base) throws java.lang.NumberFormatException Parameters: 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, int offset, int length)
toName	public static java.lang.String toName(java.lang.Class type) Parameters: type—class, which may be a primitive TYPE field. Returns the canonical name for a type.
toSring	public static java.lang.String toString(int i) Converts an int to a string using a cache.
toString	public static java.lang.String toString(byte[] bytes, int base)

Method	Description
valueOf	public static java.lang.Object valueOf(java.lang.Class type, java.lang.String value) Parameters: type—class of the instance, which may be a primitive TYPE field. value—value as a string. Converts a string value to an instance.
valueOf	public static java.lang.Object valueOf(java.lang.String type, java.lang.String value) Parameters: type—classname or type (for example, int) value—value as a string. 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) Parameters: p1—URI path segment p2—URI path segment Adds two URI path segments.
canonicalPath	public static java.lang.String canonicalPath(java.lang.String path) 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() Clears the URI mParameters.
clone	public java.lang.Object clone() Returns a URI clone.
decodePath	public static java.lang.String decodePath(java.lang.String path) Parameters: path—path the encode Decodes a URI path.

Method	Description
emcodePath	public static java.lang.String encodePath(java.lang.String path) Parameters: path—path the encode 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, java.lang.String path) Parameters: path—path the encode buf—StringBuffer to encode path into (or null) Returns the StringBuffer or null if no substitutions required. Encodes a URI path.
encodeString	public static java.lang.StringBuffer encodeString(java.lang.StringBuffer buf, java.lang.String path, java.lang.String encode) Parameters: path—path the encode buf—StringBuffer to encode path into (or null) encode—string of characters to encode. 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) Gets the named value
getEncodedPath	public java.lang.String getEncodedPath() Gets the encoded URI path.
getHost	public java.lang.String getHost() Gets the URI host.
getParameterNames	public java.util.Set getParameterNames() Gets URI query mParameters names.
getParameters	public MultiMap getParameters() Gets URI query mParameters.
getPath	public java.lang.String getPath() Gets the URI path.
getPort	public int getPort() Gets the URI port.

Method	Description
getQuery	public java.lang.String getQuery() Gets the URI query string.
getScheme	public java.lang.String getScheme() Gets the URI scheme.
getUnmodifiableParameters	public java.util.Map getUnmodifiableParameters() Gets the URI query mParameters.
getValues	public java.util.List getValues(java.lang.String name) Parameters: name—parameter name Returns named multiple values.
hashCode	public int hashCode()
hasScheme	public static boolean hasScheme(java.lang.String uri) Parameters: uri—URI Returns true if the URI has a scheme.
isAbsolute	public boolean isAbsolute() Checks to determine whether the URI is absolute.
parentPath	public static java.lang.String parentPath(java.lang.String p) Returns the parent path. Treats a URI like a directory path and return the parent directory.
put	public void put(java.lang.String encoded) Parameters: encoded—HTTP encoded string of mParameters: e.g. “a=1&b=2” Adds a dictionary to the URI query mParameters.
put	public java.lang.Object put(java.lang.Object name, java.lang.Object value) Parameters: name—name of value value—value, which may be a multi valued list or String array. Adds a name value pair to the URI query mParameters.
put	public void put(java.util.Map values) Adds encoded mParameters.
putParametersTo	public void putParametersTo(MultiMap map) Adds the URI query mParameters to a MultiMap.
remove	public void remove(java.lang.String name) Removes a named value.

Method	Description
stripPath	public static java.lang.String stripPath(java.lang.String path) Strips parameters from a path.
toString	public java.lang.String toString() Returns path up to any semicolon parameters.
toURL	public java.net.URL toURL() throws java.net.MalformedURLException 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() This method overrides clone in class MultiMap
decode	public void decode(java.lang.String query)
decode	public void decode(java.lang.String query, java.lang.String charset)
decodeString	public static java.lang.String decodeString(java.lang.String encoded) Returns a decoded string with % encoding.
decodeString	public static java.lang.String decodeString(java.lang.String encoded, java.lang.String charset) Returns a decoded string with% encoding.
decodeString	public static java.lang.String decodeString(java.lang.String encoded, int offset, int length, java.lang.String charset) Returns decoded parameters to map.
decodeTo	public static void decodeTo(java.lang.String content, MultiMap map) Parameters: content—string containing the encoded parameters Returns decoded parameters to map.

Method	Description
decodeTo	public static void decodeTo(java.lang.String content, MultiMap map, java.lang.String charset) Parameters: data—byte[] containing the encoded parameters
decodeTo	public static void decodeTo(byte[] data, MultiMap map, java.lang.String charset) Parameters: data—ByteBuffer[] containing the encoded parameters Returns decoded parameters to a map.
encode	public java.lang.String encode() Encodes hashtable with % encoding.
encode	public java.lang.String encode(java.lang.String charset) Encodes hashtable with % encoding.
encode	public java.lang.String encode(java.lang.String charset, boolean equalsForNullValue) Parameters: equalsForNullValue—if True, then an '=' is always used, even for parameters without a value.For example, "blah?a=&b=&c=". Encodes hashtable with % encoding.
encodeString	public static java.lang.String encodeString(java.lang.String string) Performs URL encoding.
encodeString	public static java.lang.String encodeString(java.lang.String string, java.lang.String charset) 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, printStackTrace, 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.

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, int groupID)
checkIndexInfo	public MetaData checkIndexInfo(java.lang.String flowName, long flowTime)
compileXPathQueries	public boolean compileXPathQueries(java.lang.String[] exps, int grpId)
classifyXPathExp	public boolean classifyXPathExp(java.lang.String exp, int groupId)
compileXPathQueries	public boolean compileXPathQueries(java.util.ArrayList exps, int grpId)
createGrpID	public int createGrpID(java.lang.String flowName, java.lang.Object flowDef)
getGrpID	public int getGrpID(java.lang.String flowName)
lookup	public IFindResult lookup(IXMLContent content, int groupID, int xpathExpressionID) throws XPathEngineException

Method	Description
lookup	public java.util.HashMap lookup(IXMLContent content, int groupID, 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.

DataTypes File

The AON DataType.xml file contains definitions for all data types used in AON including AON-specific and Java-standard data types. The"Data Types" section on 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

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 screens. 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 Documenttype 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(). 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. 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.

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.