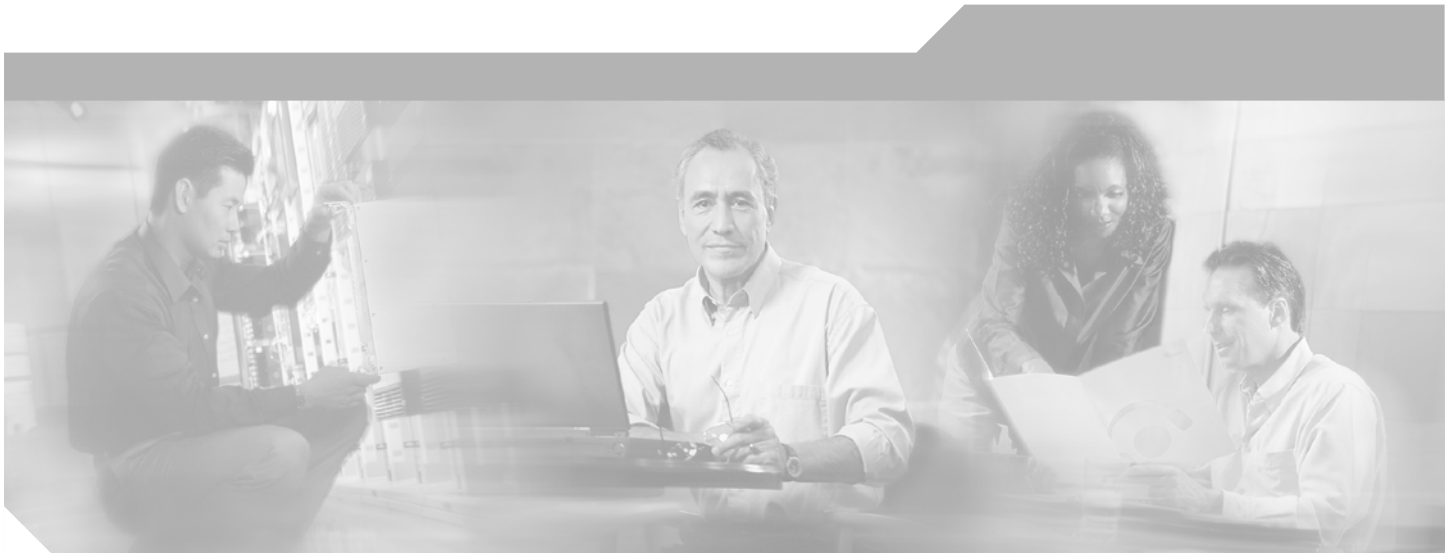


Cisco AON Development Studio User Guide

Cisco 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

Customer Order Number:
Text Part Number:

Cisco AON Development Studio User Guide

CONTENTS

CHAPTER 1

Getting Started with Cisco ADS

Cisco Application-Oriented Networking (AON) technology is the foundation for a class of network-embedded products and solutions that help converge intelligent networks with application infrastructure.

AON technology works at the application-message level by inspecting a full message, including all headers and content. It therefore understands the context of the message and can operate on those messages while they are in transit and according to business policies. AON enables you to embed intelligence capabilities into the network and significantly improve application communication.

To enable AON technology in your network, you use the following tools:

- Cisco AON Development Studio (ADS)—Windows-based tool for configuring how application messages are handled at runtime.
- Cisco AON Management Console (AMC)—Linux- and web-based server for managing an AON network. AMC synchronizes and processes input from all ADS systems on your network to ensure consistent, up-to-date configurations across all AON-enabled switches and routers.

This chapter describes how to get started using ADS.

Note For more information on implementing an AON network, see the following:

- Other chapters in this guide:
  - Chapter 2, “ADS Bladelets Reference”
  - Chapter 3, “ADS PEP Attributes Reference”
  - Chapter 4, “ADS Message Types Reference”
- Other guides in the AON library:
  - *AON Installation and Administration Guide* (for information on the AMC server and nodes)
  - *AON Programming Guide* (for information on custom Bladelets, custom adapters, and application program interfaces)

Contents

Prerequisites for Cisco ADS Installation and Operation

- Ensure that you have a Microsoft Windows 2000 or Windows XP system.
- Contact your Cisco representative to learn how to access the Cisco ADS application. Download it and make a note of where the download package resides on your system.
- Ensure that your system can connect to an AMC server.

Information About Cisco ADS

AON technology operates on your network switches and routers by means of Bladelets, Policy Execution Plans (PEPs), and message types that specify how to process particular traffic streams.

- A *Bladelet* is an operation that is performed on a message. ADS provides a repository of predefined Bladelets that are organized by category—for example, logic, message handling, security, transformation, and so on.
- A *PEP* is an assembly of Bladelets in a particular sequence.
- A *message type* is a filter that determines what type of message a PEP is to process.

You use ADS to assemble and interconnect multiple Bladelets into a PEP and assign to the PEP one or more message types. You then synchronize your ADS with your network's AMC server to deploy the PEP across your network switches and routers. AON-enabled switches and routers constitute a logical network of nodes that operate at Layer 5 and Layer 6 of the Open System Interconnection (OSI) model.

How to Use Cisco ADS

This section provides the following information:

Installing Cisco ADS

To install ADS, perform the following steps.

Step 1 Locate the ADSInstallerWin32.exe executable file.

Step 2 Double-click the file icon. The InstallShield Wizard starts up.

Step 3 In the Welcome window, click **Next**.

Step 4 In the License Agreement window, click **Accept**.

Step 5 Provide the requested information (name, organization, and e-mail address) and click **Next**.

Step 6 Specify where to install ADS as follows:

- To use the displayed location, click **Next**.
- To specify another location, click **Browse**, select a new location, and click **Next**.

Step 7 Specify a database port as follows:

- To use the displayed port, click **Next**.
- To specify another port, type another port number and click **Check Port Availability**. Repeat as needed until a message confirms that the port is available. Then click **Next**.

Step 8 Review the displayed summary information and do one of the following:

- If all settings are correct, click **Install**.
- If any setting is incorrect, click **Back**, correct the setting, return to this window, and click **Install**.

Step 9 Click **Finish**.

Step 10 Reboot your system.

Note To uninstall ADS, in Microsoft Windows choose **Start > Programs > AON Development Studio > Uninstall AON Development Studio** and follow instructions.

Starting and Exploring Cisco ADS

To start and explore ADS, perform the following steps.

Note For help at any time, from the toolbar click **Support** and then either **Release Notes** or **FAQs**.

Step 1 In Microsoft Windows, choose **Start > Programs > AON Development Studio > AON Development Studio** (or click the **AON Development Studio** icon on your desktop).

Note Startup time depends on your system hardware: more RAM and faster bus and processor speeds mean shorter startup time.

Step 2 At the ADS Login window (Figure 1-1), do the following:

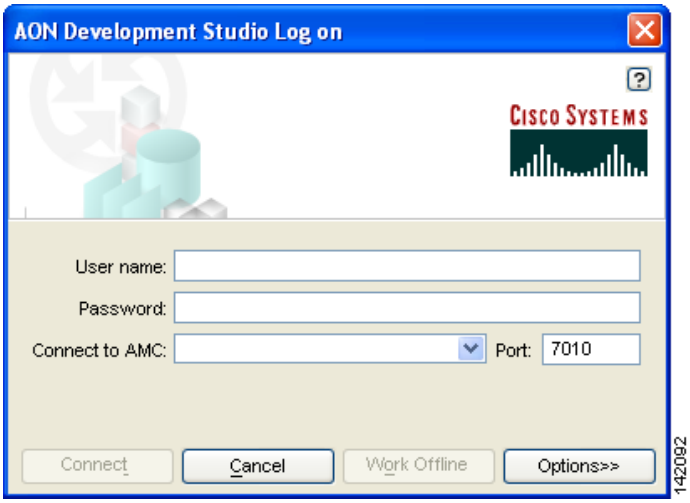
a. Provide the following requested information:

- Username
- Password
- Connect to AMC (click the dropdown arrow to display AMC choices; choose or provide the hostname or IP address for your AMC server)
- Port (port on which the AMC server listens for traffic; default is 7010)

b. Click **Connect**. ADS connects to and synchronizes with the AMC server.

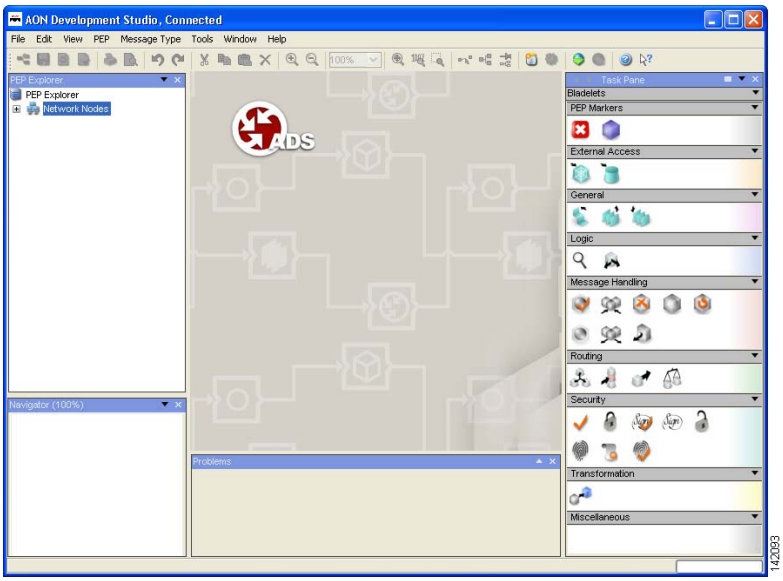
Note On first login, you must connect so that your ADS can display existing nodes. On future logins, you can either connect or work offline.

Figure 1-1 ADS Login Window



Step 3 Examine and familiarize yourself with the initial ADS window (Figure 1-2).

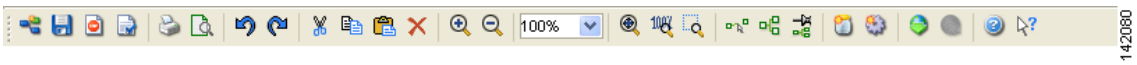
Figure 1-2 Initial ADS Window



Notice that the window has a toolbar and an icon bar (icons are dimmed until operable) across the top and several screen panes below.

Icons (Figure 1-3) provide shortcuts to various ADS functions, most of which you can also access both from the toolbar and by means of a mouse right-click. You can determine what an operable (that is, undimmed) icon does by holding your mouse over it.

Figure 1-3 ADS Icons



The various panes of the ADS window are as follows:

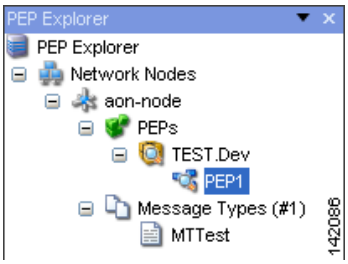
- PEP Explorer Pane, page 1-6
- Navigator Pane, page 1-6
- PEP Developer Pane, page 1-7
- Problems Pane, page 1-7
- Repository Pane, page 1-8

Note Panes other than the Repository pane are empty when you start up for the first time. They are shown below as they would look if you were in the midst of PEP design.

PEP Explorer Pane

The PEP Explorer pane (Figure 1-4) in the upper left portion of the window displays the hierarchy of available system nodes and associated PEPs and message types that reside in your ADS. The PEPs and message types are of your own creation or were created by others and downloaded to your ADS during synchronization with the AMC server. You can turn the display on or off by clicking **View** and checking or unchecking **PEP Explorer**.

Figure 1-4 ADS WIndow: PEP Explorer Pane



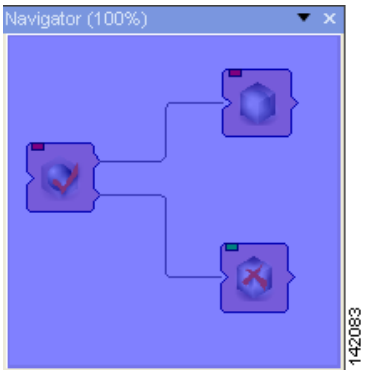
For you to be able to create a PEP and message type, at least one node must have been created on the AMC server and displayed on your ADS. You can create any number of PEPs and message types beneath a node.

After you synchronize your ADS with the AMC server, the PEP Explorer pane refreshes to display any additional PEPs that other users may have posted to the AMC server.

Navigator Pane

The Navigator pane (Figure 1-5) in the lower left portion of the window displays a map of the entire PEP that you are configuring and, in blue, the portion of that mat that is displayed in the PEP Developer pane (described below). It enables you to navigate to different parts of the PEP quickly and easily, which is particularly useful if the PEP is large and complex. You can turn the display on or off by clicking **View** and checking or unchecking **Navigator**.

Figure 1-5 ADS Window: Navigator Pane

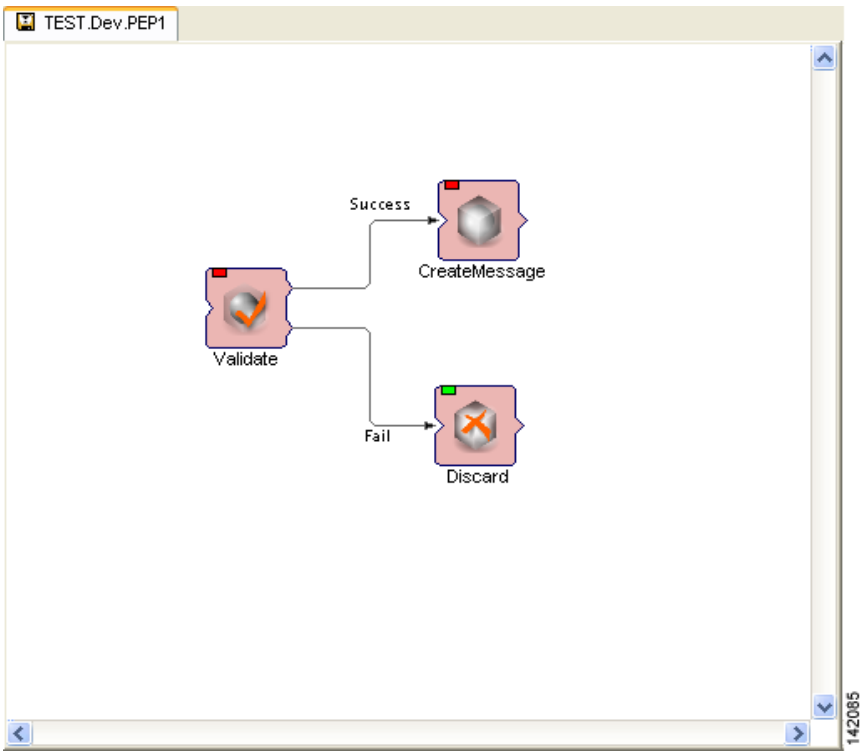


PEP Developer Pane

The PEP Developer pane (Figure 1-6) in the middle of the window is your workspace for designing a PEP. You drag and drop various Bladelets to that pane and interconnect them to create a PEP. This pane is always displayed.

You can display multiple PEPs at one time, each within its own tabbed view in the pane. PEP names are displayed in the tabs and also in the PEP Explorer pane hierarchy.

Figure 1-6 ADS WIndow: PEP Developer Pane



Problems Pane

The Problems pane (Figure 1-7) beneath the PEP Developer window displays a list of critical problems and alerts, as follows:

- Critical problems (denoted by red circles with Xs in them) prevent the PEP from performing a valid action. You must resolve all critical problems before you can synchronize your ADS with the AMC server or save the PEP as a template for future PEP development.
- Alerts (denoted by yellow triangles) prevent the PEP from operating properly.

Figure 1-7 ADS Window: Problems Pane

You can rearrange the display by clicking any of the column headings (Severity, Description, and Location). By default, problems are displayed by severity type (critical problem or alert) and, within a severity type, in alphabetical order. You can toggle between ascending and descending order by clicking a heading. You can turn the display on or off by clicking **View** and checking or unchecking **Problems**. Double-clicking any individual problem selects the offending Bladelet in the PEP Developer pane.

Repository Pane

The Repository pane (Figure 1-8) on the right side of the window is the source from which you drag and drop Bladelets into the PEP Developer pane. You can turn the pane display on or off by clicking **View** and checking or unchecking **Repository**.

ADS provides a predefined list of Bladelets for you to use. Different Bladelet categories have different functions.

Note

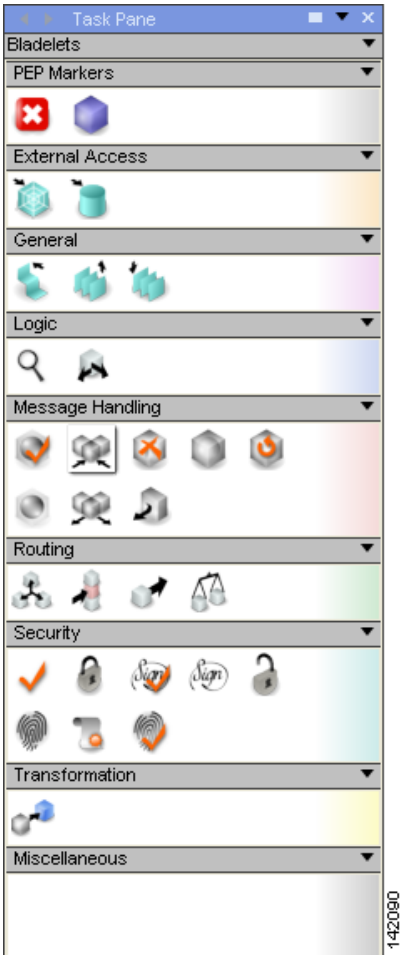
- Two Bladelet types of particular importance are the exception and response markers, at the top of the Repository pane. The exception marker tracks and records exceptions in the PEP. The response marker responds to a particular action that a message has undergone in the PEP. You place and interconnect these Bladelets just as you do any other Bladelet.

  For information on these and other predefined Bladelets, see Chapter 2, "ADS Bladelets Reference."

- Although doing so should rarely be necessary, you can create custom Bladelets. Custom Bladelets are best created by programmers. The programmer develops the Bladelet, uses the ADS Packaging Wizard to package the resulting files into a single file, and uploads the file to the AMC server. The new Bladelet becomes available to ADS users after synchronization.

  For information on custom Bladelets, see the *AON Programming Guide*.

Figure 1-8 ADS Window: Repository Pane



Step 4 Set your tools options as follows:

a. From the toolbar, click **Tools > Options**.

b. In the Options window, for each of the following tabs in turn, set your preferences and click **Apply**:

- Settings: Set preferences for language, look and feel, tool tips, frame size, and containers state.
- Files and Folders: Set preferences for directories last used, file-list size, and save preview.
- Renderer: Set preferences for interaction and navigation settings.

c. Click **OK**.

Step 5 When you are done with your work session, close ADS by choosing **File > Quit**.

Creating PEPs

To create a PEP, perform the following steps.

Timesaver Most steps instruct you to click an icon or, alternatively, choose a command sequence. Instructions to choose a command sequence (example: choose **PEP > New**) refer to commands on the ADS toolbar. In many cases you can access the same command sequence by right-clicking the relevant entry in the PEP Explorer pane or the relevant Bladelet in the PEP Developer pane.

Step 1 Start ADS and log in (in the "Starting and Exploring Cisco ADS" section, see Steps 1 and 2).

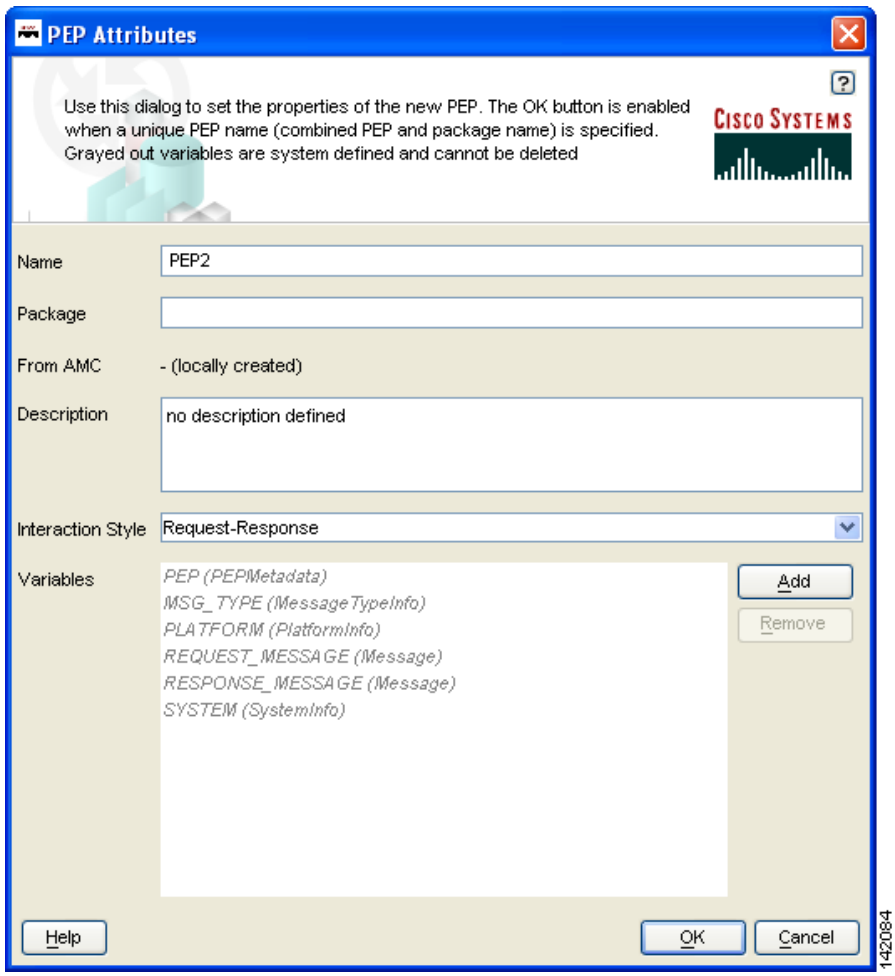
Step 2 In the PEP Explorer pane, click the AON node where the PEP is to reside.

Step 3 Start a new PEP by clicking the **New** icon (or choosing **PEP > New**).

Note On subsequent use, if you have saved a PEP template, you can choose **PEP > New PEP from Template**. Preview your saved templates, select one, and click **OK**.

Step 4 In the PEP Attributes window (Figure 1-9), provide the required information (name, package, description, interaction style, and variables) to the new PEP and click **OK**:

Figure 1-9 PEP Attributes Window



Note We recommend that you define most PEP attributes when you start to create a PEP; however, you can define or modify them later.

Use the default name (PEP *N*) or assign a new name. If a PEP with the same name exists, the OK button in this window is dimmed, enforcing the rule that every PEP name must be unique.

The package an optional mechanism for organizing your PEPs in one area of a node. You can assign one or more PEPs to a package. The default is to leave the package field blank. We recommend that you either leave it blank or define it later, just before you deploy the PEP. If you define it now, make sure it is defined correctly; you cannot change it later.

For information on PEP attributes, see Chapter 3, "ADS PEP Attributes Reference."

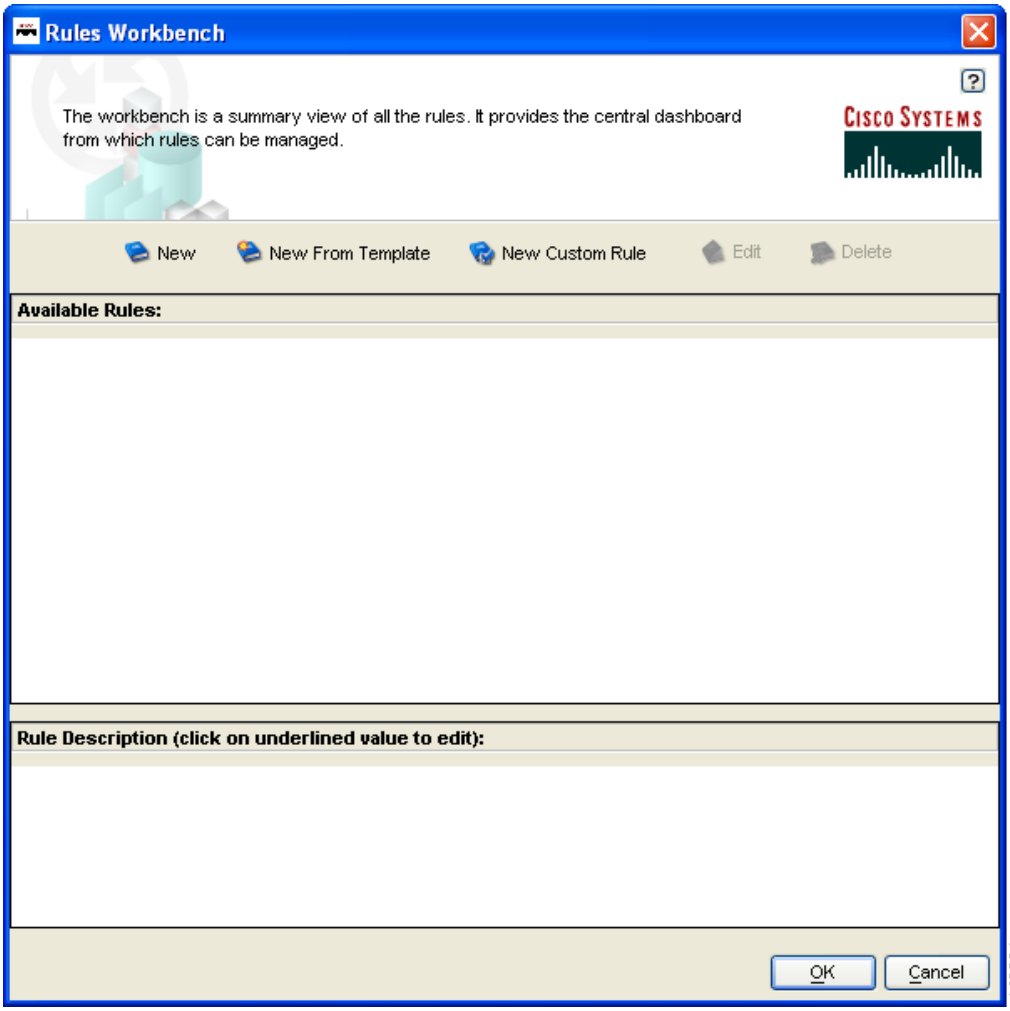
Step 5 Construct the PEP as follows:

a. Optionally, invoke PEP-development rules as follows:

Note For information on rules, see Chapter 2, "ADS Bladelets Reference."

1. Click the **Rules Wizard** icon (or choose **Tools > Rules Wizard**).
2. In the Rules Workbench window sequence (Figure 1-10, first in the sequence), create rules as needed, select rules as appropriate, and click **OK**.

Figure 1-10 Rules Workbench



b. Drag and drop Bladelets from the Repository pane to the PEP Developer pane.

Note Alternatively, copy and paste Bladelets from another PEP or from different areas of the same PEP.

c. Reposition Bladelets by dragging and dropping as needed:

- To select a single Bladelet for dragging, click it.
- To select multiple Bladelets for dragging as a unit, hold the left mouse button down, draw a box around the Bladelets, and release the button.

d. (Optional) Add portions of this or another PEP as needed by clicking a corner of its PEP Development screen, dragging the cursor to draw a rectangle around the relevant area, dropping the cursor, copying the selected area, and pasting it into the new PEP.

e. Add paths between Bladelets by either of the following methods:

- Drag and drop: Drag a Bladelet until it touches or overlaps another Bladelet and a plus sign appears, then drop it.

- Edge creation: Click the Edge Creation icon. Then click a Bladelet, drag the cursor to another Bladelet, and release.

The line and arrow connecting the two Bladelets show the path over which and direction in which information passes through the PEP.

Note You can turn edge creation on or off by clicking the Toggle Edge Creation icon. When it is on, the background of the PEP Developer pane changes from white to blue.

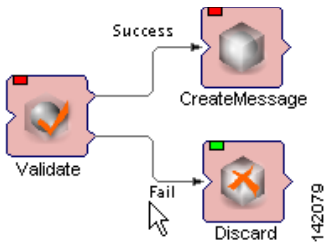
f. (Optional) Change paths between Bladelets as follows:

- To remove a path altogether, click the path in the middle, then either press **Delete** on your keyboard or drag it away and drop it.
- To connect a path to another Bladelet, click the end of the path that you want to redirect, then drag it elsewhere—to another bladelet or to an exception marker—or leave it open-ended for now.

g. As needed, create paths for branching Bladelets (Figure 1-11) as follows:

- To create a success path, drag the path starting from the top half of the branching Bladelet or drop the target Bladelet over the top half of the branching Bladelet.
- To create a failure path, drag the path starting from the bottom half of the branching Bladelet or drop the target Bladelet over the bottom half of the branching Bladelet.

Figure 1-11 Branching Bladelet with Success and Failure Paths



h. Configure each Bladelet as follows:

1. Right-click a Bladelet and click **Bladelet Properties**.
2. Adjust import and export parameters and other settings as needed. Parameters and settings differ for different Bladelet categories.

Note For information on Bladelet properties, see Chapter 2, "ADS Bladelets Reference."

3. Click **OK**.
4. Repeat for all Bladelets.

i. Edit your work by selecting one or more Bladelets and clicking an icon (or clicking **Edit** and an option).

Typical GUI edit options are available, including those listed in Table 1-1(listed in their order of appearance on the icon bar).

Table 1-1 ADS Edit Operations

Operation	Notes
Undo, Redo	• Permits you to undo and redo a virtually unlimited series of changes. However, if you perform one or more undo operations and then make new changes, the original chain of operations is broken and a new chain is started. The default is to use the most recent chain.
Cut, Copy, Paste	• Retains one cut or copied item on the clipboard. The item can come from ADS or any other application and can be used by ADS or any other application. • Deletes that item from the clipboard if a second item is added. • If you paste an item, positions it in the center of the window, rather than at the cursor position.
Delete	• Completely removes the item from the PEP.
Select All	• Is useful when the scope of the PEP falls beyond the viewable pane. • Operates on a single PEP only. If you are working on multiple PEPs, it operates only on the active PEP. • Is available only from the toolbar and not from the icon bar.

Typical GUI view options are available, including those listed in Table 1-2 (listed in their order of appearance on the icon bar).

Table 1-2 ADS View Operations

Operation	Notes
Zoom	• +10%, –10% zooms the current PEP by plus or minus 10% per click. • 1:1 returns any zoom level to the default setting.
Fit to Screen	• Provides a high-level view of your PEP, which is useful for very large PEPs. If Bladelet images and text become too small to read, view your PEP in the Navigator pane instead.
Zoom Rectangle	• Permits you to create a “rubber band” rectangle around a portion of the PEP Developer pane and zoom it in any number of times.

Typical GUI layout options are available, including those listed in Table 1-3 (listed in their order of appearance on the icon bar).

Table 1-3 ADS Layout Operations

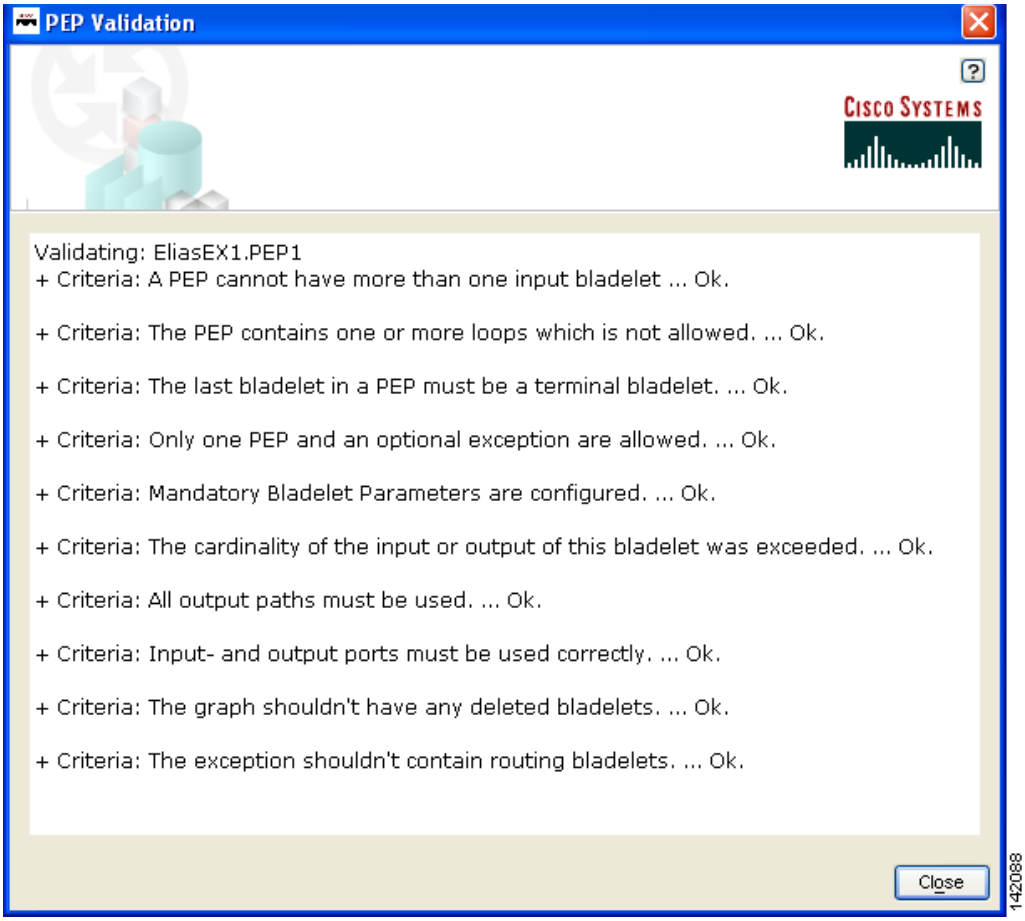
Operation	Notes
Toggle Edge Creation	• When toggled on, permits you to create paths between Bladelets by connecting them with arrows, without having to move Bladelets around the screen. • When toggled off (default state), permits you to create paths between Bladelets only by dragging one Bladelet over another until a plus sign indicates that a path is created.
Layout	• Rearranges your PEP in a hierarchal manner with the top layer on the left and the bottom layer on the right.
Toggle Automatic Layout	• When toggled on, adjusts the spacing between Bladelets and paths to achieve minimal line intersections, text runover, and graphical overlaps. Any Bladelet that you drag and drop on the PEP Developer window is automatically placed on the left side of the window. • When toggled off (the default setting), displays Bladelets in the window where you drop them.

j. Validate your PEP often as follows:

1. Address problems that are displayed in the Problems pane. Double-click a problem to select the offending Bladelet in the PEP Developer pane.
2. Address problems that are displayed in the PEP Validation Report (Figure 1-12). Generate the report by clicking **Tools > Validate PEP**.

Note We recommend that you address problems as they arise. Validation ensures adherence to all rules and parameters that govern individual Bladelets, paths among Bladelets, and the routing of messages through a PEP. A PEP must be valid before you can deploy it during synchronization of your ADS with the AMC server.

Figure 1-12 PEP Validation Report



k. Save your PEP often:
   - To save the PEP with its current name, click the **Save** icon (or choose **PEP > Save**).
   - To save the PEP with a new name, choose **PEP > Save As**.

l. Optionally, save the PEP as a template for future use by choosing **PEP > Save PEP as Template**.

m. Optionally, close the PEP by clicking the **Close** icon (or choosing **PEP > Close**.)

Step 6 As needed, create additional PEPs.

Note Navigate among multiple open PEPs by clicking the appropriate tab in the PEP Developer pane (or clicking **Window** and the desired PEP).

Step 7 As needed, modify or finish defining attributes for each PEP as follows:

a. Click the appropriate tab in the PEP Developer pane.

b. Click the **Attributes** icon.

c. In the PEP Attributes window (similar to that for creating a new PEP except that the name and package, if already defined, are uneditable), provided the requested information and click **OK**.

Note For information on PEP attributes, see Chapter 3, "ADS PEP Attributes Reference."

Step 8 Close each open PEP by clicking its tab and then clicking the **Close** icon (or choosing **PEP > Close**).

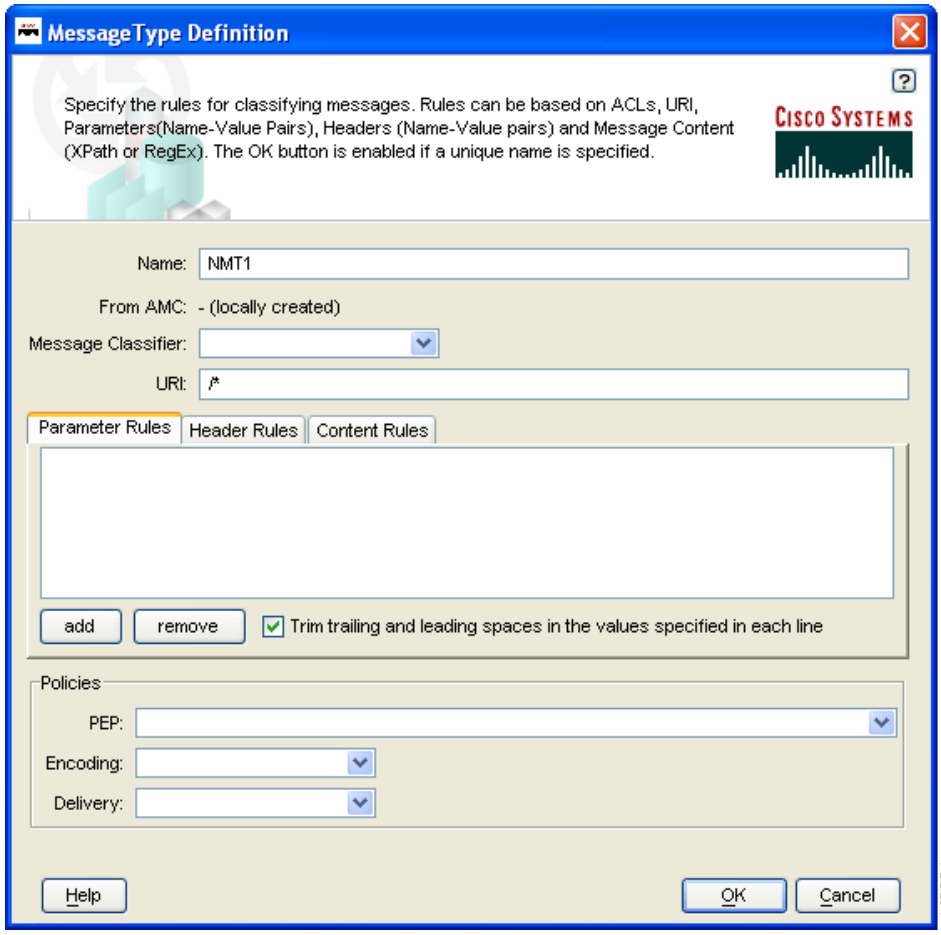
Step 9 Create and assign message types for a PEP as follows:

a. In the PEP Explorer pane, under the appropriate node, select **Message Types**. Then choose **Message Type > Message Type**.

b. In the Message Type Definition window (Figure 1-13), provide the requested information and click **OK**.

Note For information on message types, see Chapter 4, "ADS Message Types Reference."

Figure 1-13 Message Type Definition Window



The new message type now appears in the PEP Explorer pane.

c. As needed, create additional message types for the PEP.

Note Until you reorder them, message types are displayed in the PEP Explorer pane in the order in which you create them. However, because message types contain conditions that govern whether or not a message proceeds to the PEP for processing, their order is important. You can reorder them later (in the "Deploying PEPs" section, Step 3), after you synchronize your ADS with the AMC server.

Step 10 Optionally, print your PEP as follows:

a. Set up the page by clicking the **Print Preview** icon (or choosing **PEP > Page Setup**).

b. Preview the page by clicking the **Print Preview** icon (or choosing **PEP > Print Preview**).

c. Print the page by clicking the **Print** icon (or choosing **PEP > Print**).

Step 11 Optionally, exit ADS by choosing **File > Quit**.

Deploying PEPs

Deploying PEPs involves synchronizing your ADS with the AMC server. ADS posts your valid PEPs and message types to the server. The server posts new nodes, PEPs, Bladelets, message types, and other information to your ADS for your use.

To deploy one or more PEPs (and associated message types), perform the following steps.

Timesaver Most steps instruct you to click an icon or, alternatively, choose a command sequence. Instructions to choose a command sequence (example: choose **PEP > New**) refer to commands on the ADS toolbar. In many cases you can access the same command sequence by right-clicking the relevant entry in the PEP Explorer pane or the relevant Bladelet in the PEP Developer pane.

Step 1 Start ADS and log in ("Starting and Exploring Cisco ADS" section, Steps 1 and 2).

Step 2 Synchronize your ADS with the AMC server as follows:

a. Ensure that you are connected to an AMC server. If you are working offline, click the **Connect** icon (or choose **File > Connect**) and connect now.

b. Click the **Synchronize** icon (or choose **File > Synchronize**).

c. In the Synchronization window, select valid PEPs (Figure 1-14) and message types (Figure 1-15) as needed, and click **OK**.

During synchronization, your selections post to the AMC server, and any updates on the server post to your ADS.

Figure 1-14 Synchronization Window: PEPs

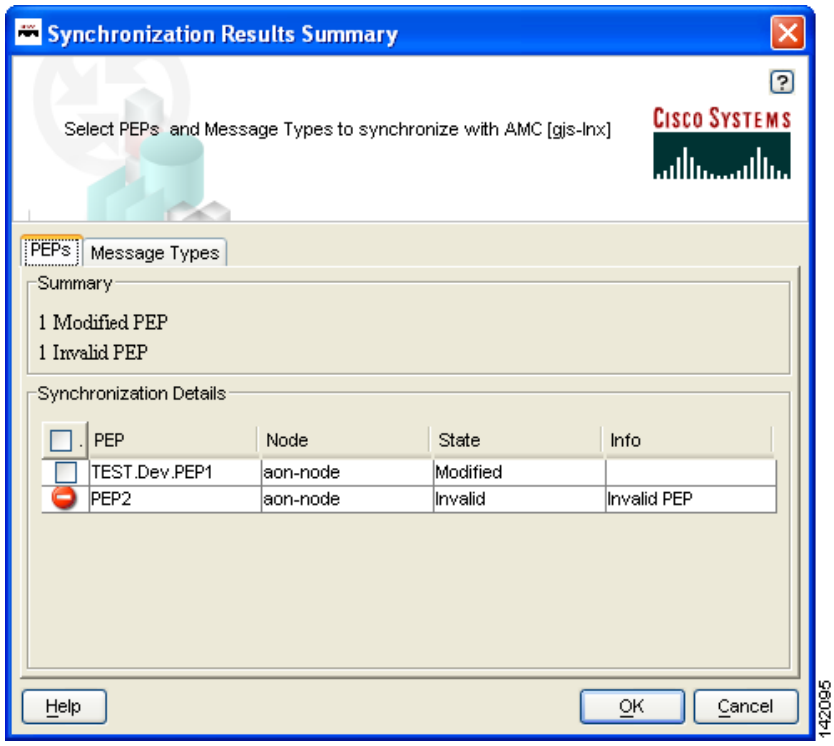
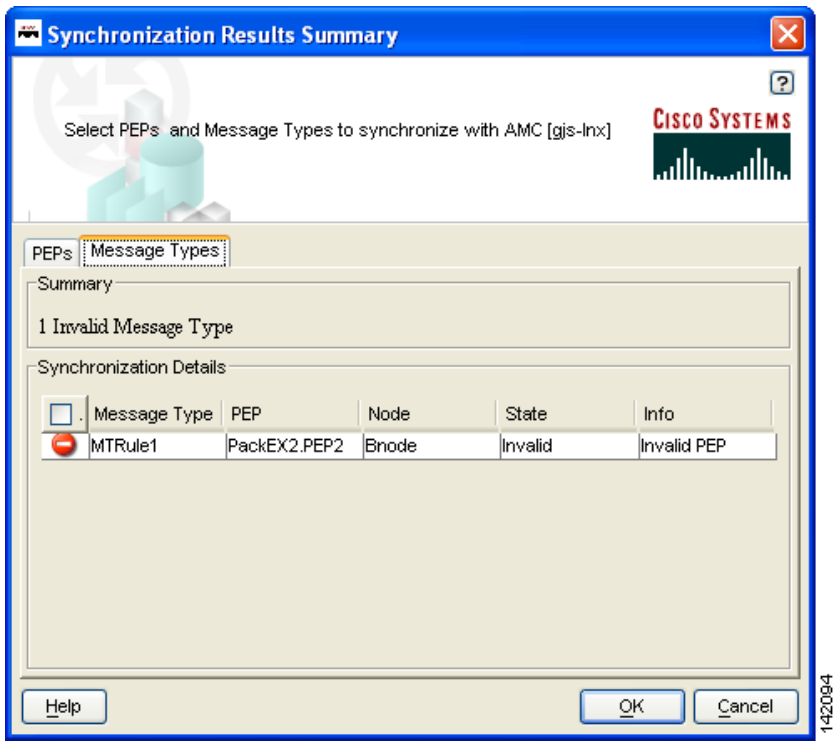


Figure 1-15 Synchronization Window: Message Types



Step 3 As needed, reorder your message types as follows:

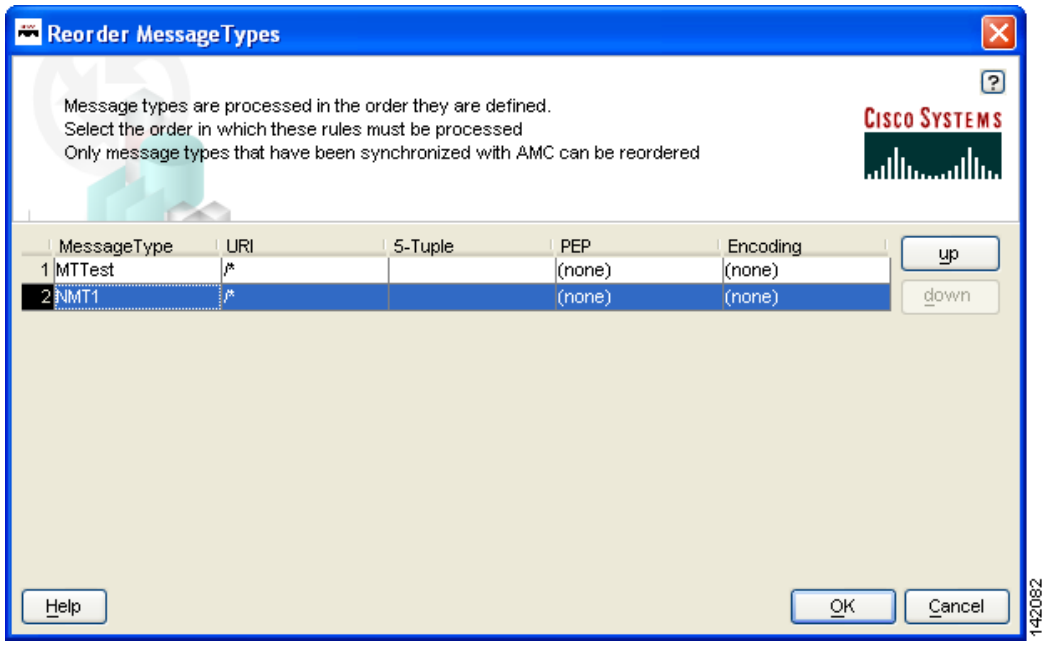
Note Because message types contain conditions that govern whether or not a message proceeds to the PEP, the order in which you list multiple message types is important.

a. Save and close all open PEPs.

b. In the PEP Explorer pane under the relevant node, select **Message Types (#*N*)** and choose **Message Type > Reorder Message Types**.

c. In the Reorder Message Types window (Figure 1-16), select a message type, click up or down as needed, and click **OK**.

Figure 1-16 Reorder Message Types Window



Step 4 Synchronize again to preserve the new order (see Step 2).

Step 5 Exit ADS by choosing **File > Quit**.

Where To Go Next

- For information on predefined Bladelets and rules, see Chapter 2, "ADS Bladelets Reference."
- For information on PEP attributes, see Chapter 3, "ADS PEP Attributes Reference."
- For information on message types, see Chapter 4, "ADS Message Types Reference."
- For information on the AMC server and nodes, see the *AON Installation and Administration Guide*.
- For information on custom Bladelets, custom adapters, and application program interfaces, see the *AON Programming Guide*.

CHAPTER 2

ADS Bladelets Reference

A Bladelet is an operation that is performed on a message. Cisco AON Development Studio (ADS) provides a repository of standard Bladelets that are organized by category—for example, logic, message handling, security, transformation, and so on. This chapter presents detailed reference information that you need to choose and use ADS Bladelets.

Note For more information on implementing an AON network, see the following:

- Other chapters in this guide:
  - Chapter 1, "Getting Started with Cisco ADS"
  - Chapter 3, "ADS PEP Attributes Reference"
  - Chapter 2, "ADS Bladelets Reference"
- Other guides in the AON library:
  - *AON Installation and Administration Guide* (for information on the AMC server and nodes)
  - *AON Programming Guide* (for information on custom Bladelets, custom adapters, and application program interfaces)

Contents

Information About Bladelets

Bladelets are used in the construction of Policy Execution Plans (PEPs). You construct a PEP with the graphical-user-interface (GUI) ADS tool, which enables you to drag and drop icons representing Bladelets onto a canvas. You then "connect" the Bladelets, thus forming a PEP.

Bladelets are highly configurable. Using ADS, you configure Bladelets during normal PEP construction by setting their properties, which are grouped hierarchically into three levels:

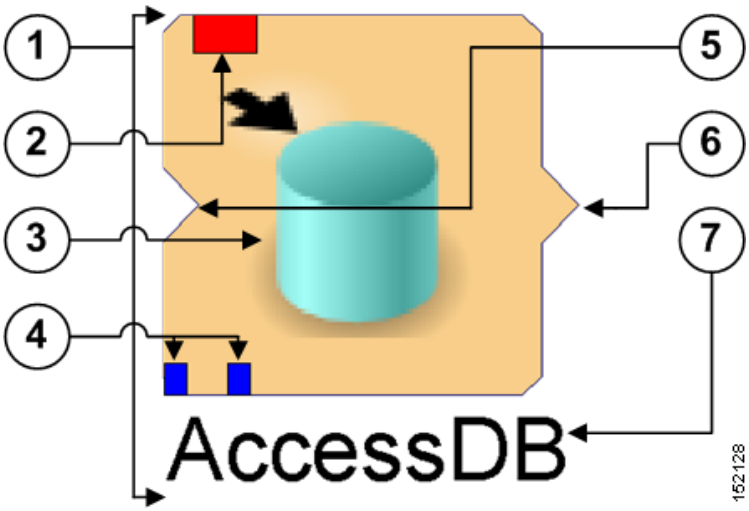
<configuration-group>

<configuration-subgroup>

<parameter-group> and <parameter>

Figure 2-1 shows the various components of a Bladelet. (The example shown below is an Access DB Bladelet.)

Figure 2-1 ADS Bladelet



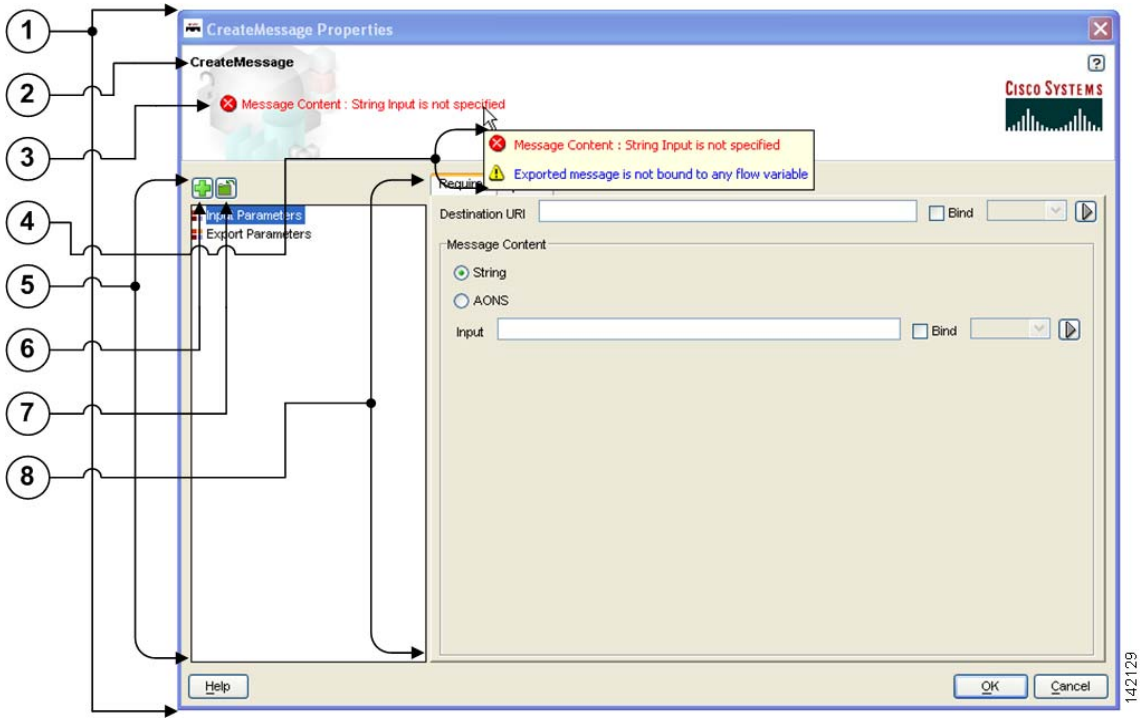
1	Whole Bladelet icon.	**5**	Bladelet input connection (connects to the output connection of another Bladelet).
2	Bladelet configuration status: • Red—One or more critical errors • Yellow—One or more warnings • Green—No critical errors or warnings	**6**	Bladelet output connection (connects to the input connection of another Bladelet). If two output connections exist, output paths usually designate the top one for a successful outcome and the bottom one for a failed outcome.
3	Bladelet graphic.	**7**	Bladelet label.
4	Bladelet exception PEP markers (connection points for specific types of exceptions).		

Bladelet Properties Window and Dialog Boxes

Common tasks involving creating PEPs are discussed in Chapter 1, "Getting Started with Cisco ADS." This section describes how to assign Bladelet properties.

You assign Bladelet properties by means of the Bladelet Properties window (Figure 2-2) and subsequent dialog boxes. (To open this window, follow the procedure in the "Creating PEPs" section on page 1-9.) (The window shown here is for a CreateMessage Bladelet.)

Figure 2-2 Bladelet Properties Window



1	Bladelet properties window	**5**	Properties area
2	Bladelet label	**6**	Add Variable icon (displays the Add PEP Variable dialog box)
3	Error-log message	**7**	Restore Previous Values icon
4	Error-log messages popup window	**8**	Value Settings area (those marked with a red asterisk are required for validation)

The window provides access to the following dialog boxes:

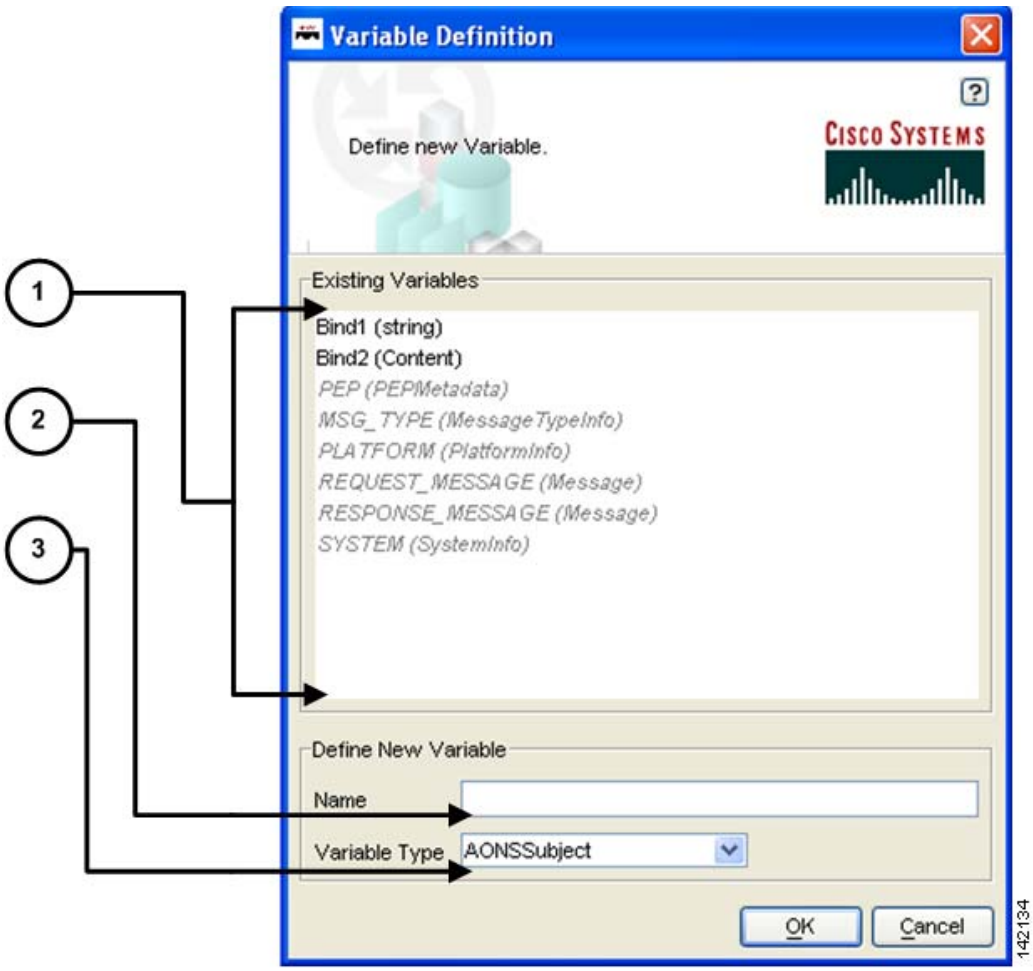
- Variable Definition Dialog Box, page 2-4
- Variable Picker Dialog Box, page 2-5
- Rules Workbench Dialog Box, page 2-6

Changes that you make to one dialog box are reflected, as appropriate, in the same dialog box for other related Bladelets.

Variable Definition Dialog Box

The Variable Definition dialog box (Figure 2-3) appears when you click the **Add Variable** icon in the Bladelet Properties window.

Figure 2-3 Variable Definition Dialog Box



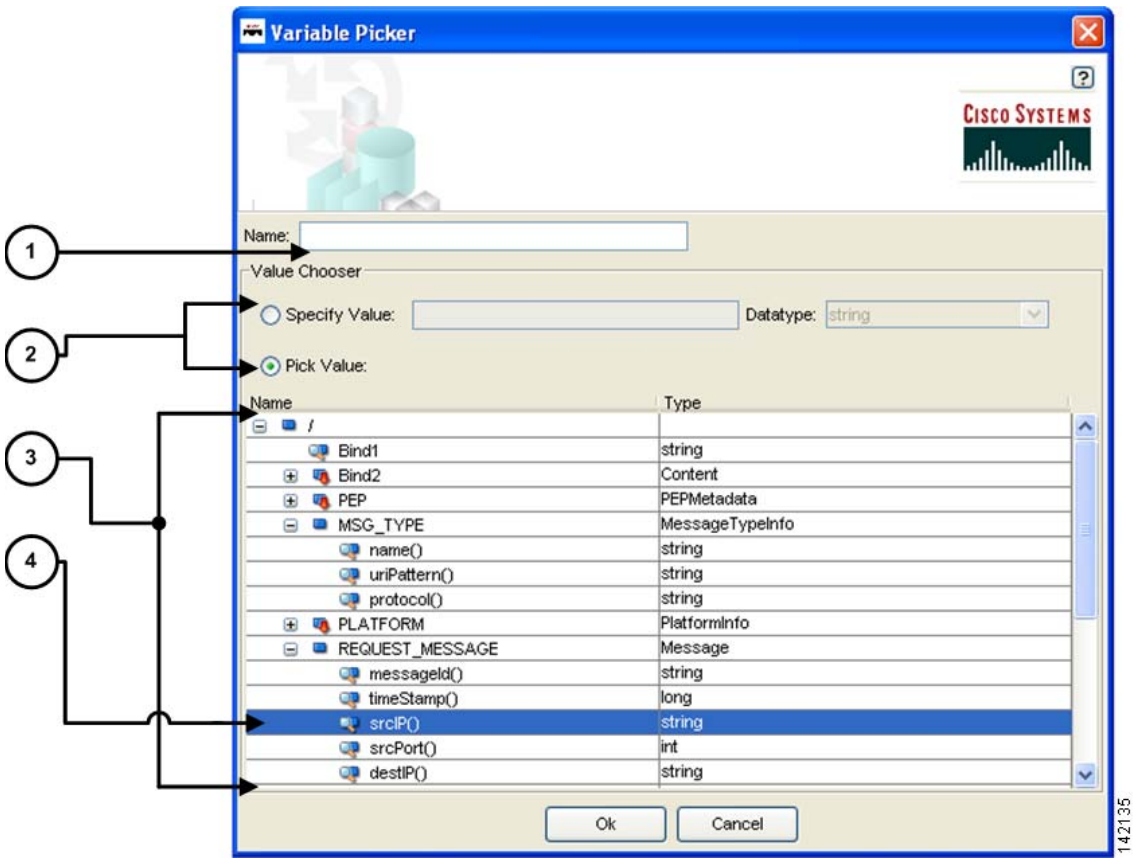
1	Existing Variables area (lists current existing variables and their types)	**3**	Define New Variable area; Variable Type drop-down list
2	Define New Variable area; Name field (enter the new variable's name here)		

Variable Picker Dialog Box

The Variable Picker dialog box (Figure 2-4) appears when you click the arrow to the far right of any required, optional, or advanced property in the Bladelet Properties window.

- There are instances when working in the properties section of a Bladelet where you can bind an input parameter to a PEP variable value that already exists.

Figure 2-4 Variable Picker

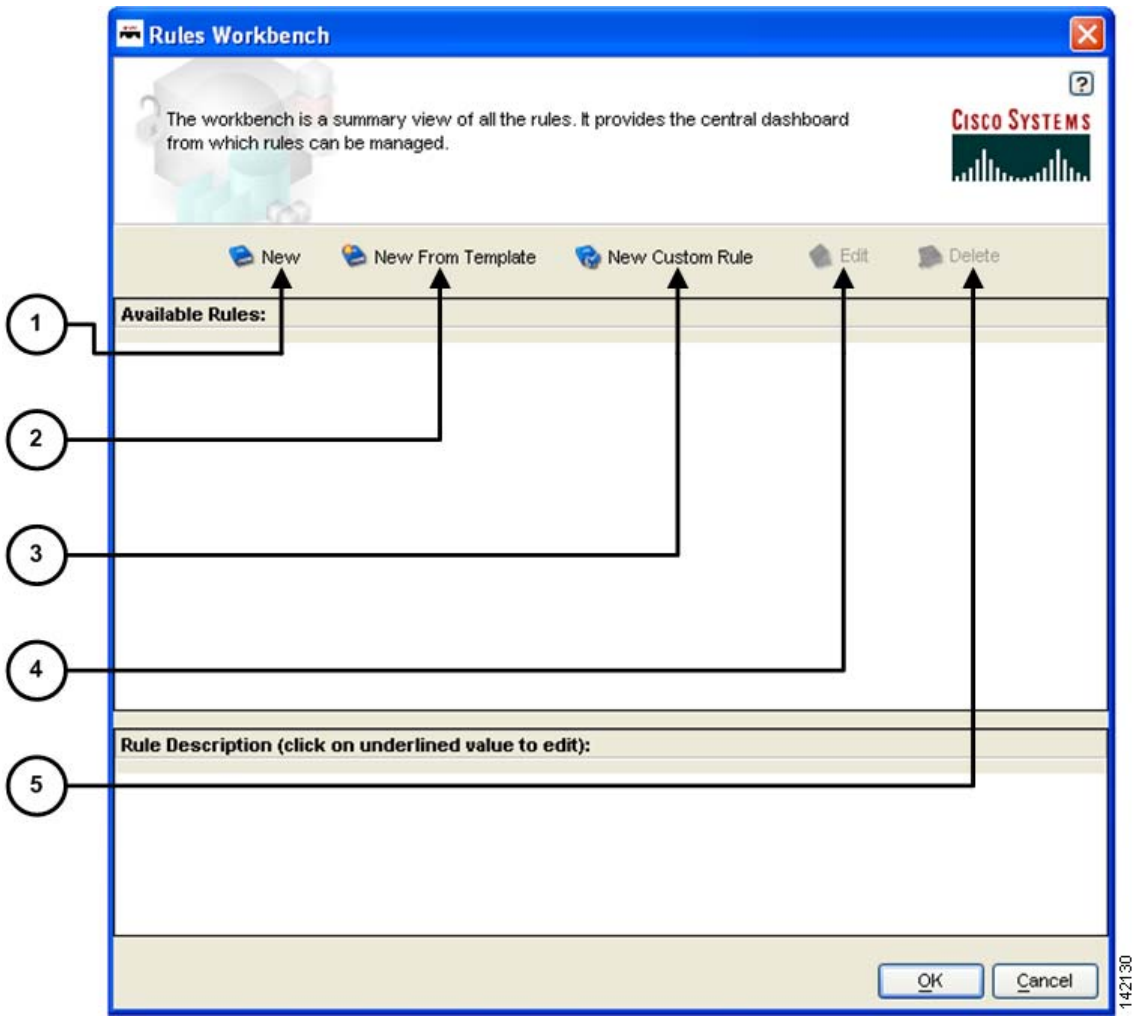


1	Name	**3**	List of available values for picking
2	Value Chooser area (specify or pick one)	**4**	Chosen value

Rules Workbench Dialog Box

The Rules Workbench dialog box (Figure 2-5) appears when you click the **Rules Wizard** icon (or choose **Tools > Rules Wizard**).

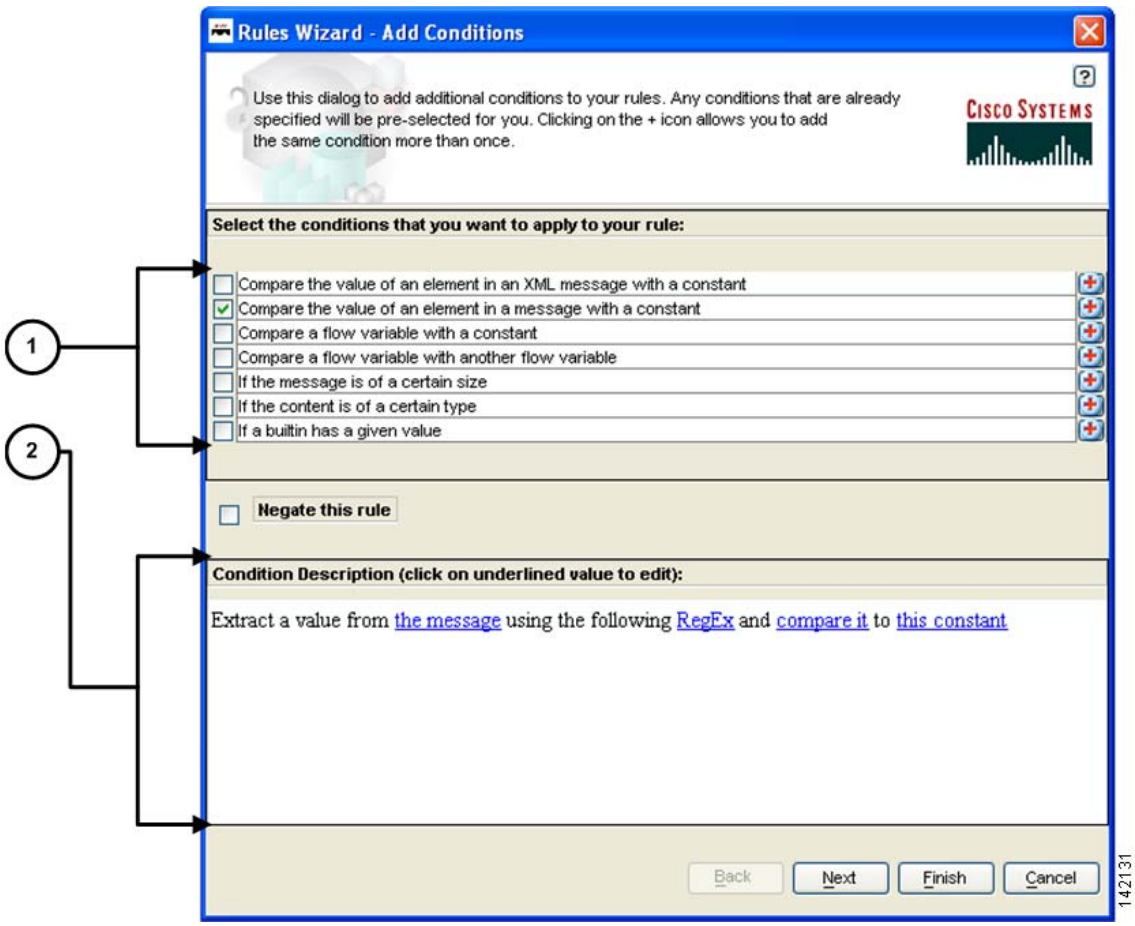
Figure 2-5 Rules Workbench Dialog Box



1	New icon	**4**	Edit icon
2	New from Template icon	**5**	Delete icon
3	New Custom Rule icon		

The Rules Wizard—Add Conditions dialog box (Figure 2-6) appears when you click the **New** icon in the Rules Workbench dialog box.

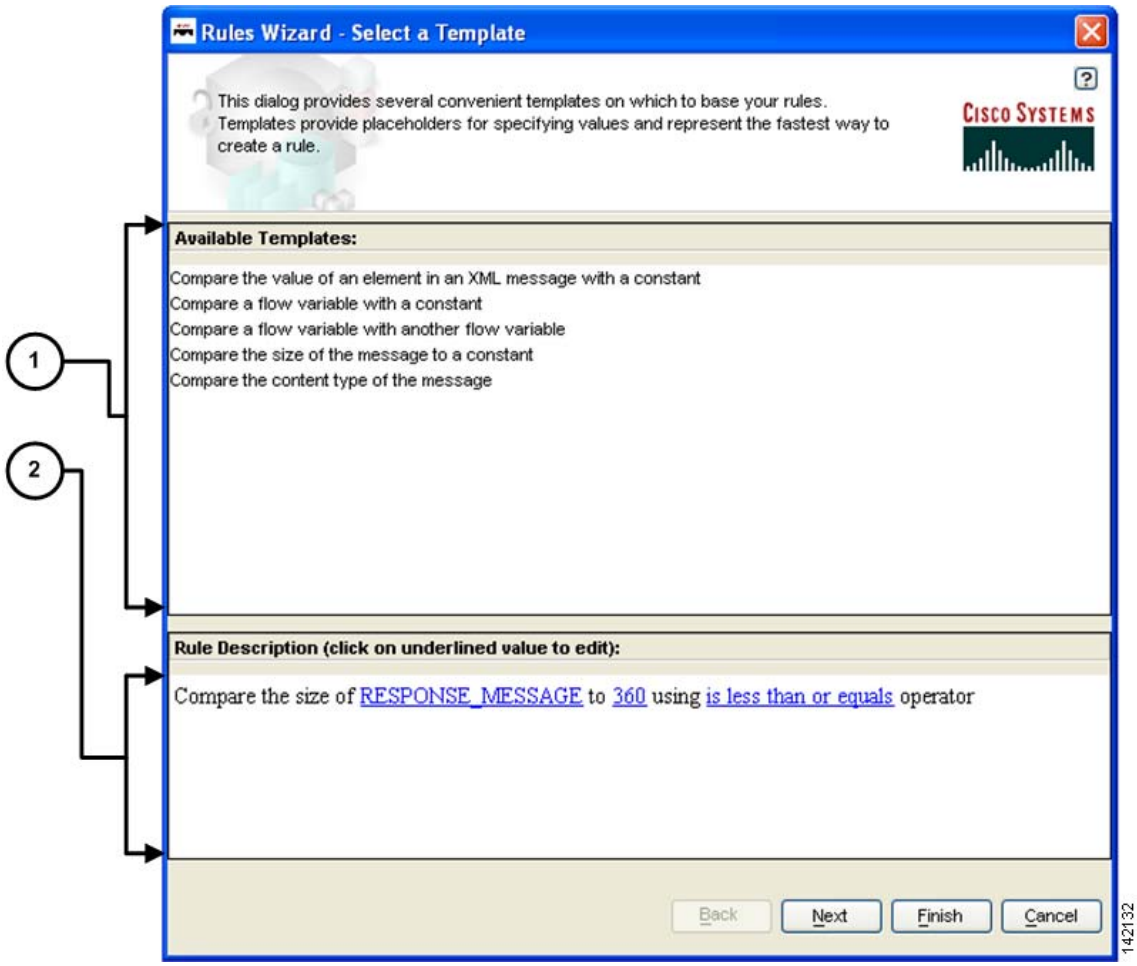
Figure 2-6 Rules Wizard—Add Conditions



1	List of available conditions from which to select (click a condition and set its values)	**2**	Conditions Description area

The Rules Wizard—Select a Template dialog box (Figure 2-7) appears when you click the **New from Template** icon in the Rules Workbench dialog box.

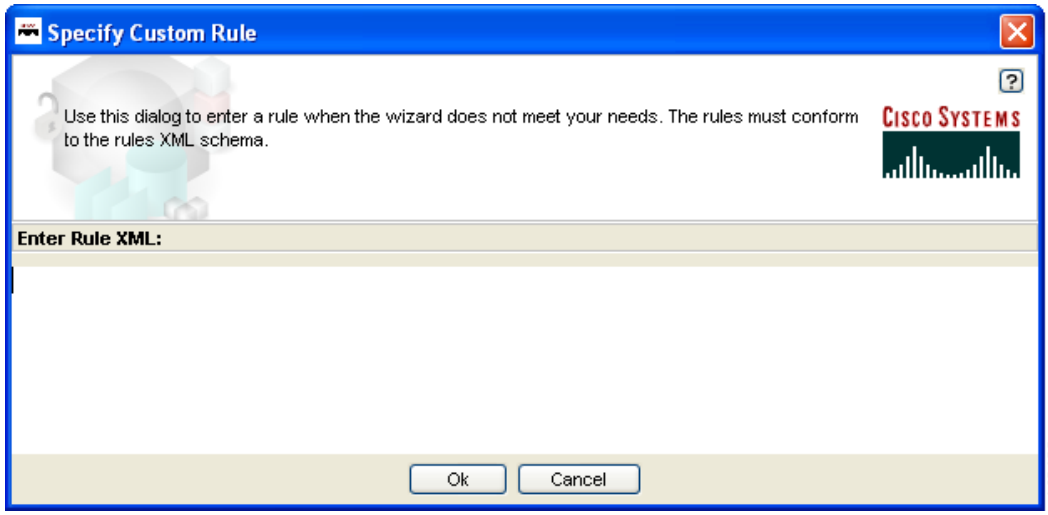
Figure 2-7 Rules Wizard—Select a Template



1	Available Templates area (click a template)	**2**	Rule Description area (click a rule and set its values)

The Rules Workbench—Specify Custom Rules dialog box (Figure 2-8) appears when you click the **New Custom Rules** icon in the Rules Workbench dialog box. Manually enter values that you want to include in a condition.

Figure 2-8 Rules Workbench—Specify Custom Rule



Bladelet Choices

This section describes the predefined Bladelets that ADS displays in its Repository pane. It also describes any Bladelet properties that you need to set in order for the Bladelet to function properly.

ADS provides the following Bladelet categories:

- PEP Markers Category, page 2-10
- External Access Category, page 2-10
- General Category, page 2-20
- Logic Category, page 2-30
- Message Handling Category, page 2-37
- Routing Category, page 2-65
- Security Category, page 2-82
- Transformation Category, page 2-131
- Miscellaneous Category, page 2-133

Note Many of the following windows allow you to specify values in one or more of the following ways:

- By typing them in directly
- By selecting them from a drop-down list
- By binding the parameter to a specific value

PEP Markers Category

In the PEP Markers category, there are two markers:

- Exception-PEP Marker
- Response Marker

Exception-PEP Marker

Use the Exception-PEP marker for tracking and recording exceptions in the PEP. It is a good way to create instances that you can store as database records to audit exceptions as information is routed through the PEP.

There are no properties to set for this marker.

Response Marker

Use the Response marker to create a marker for responding to a particular action that a message has undergone in the PEP. It is a useful way to record information based on actual actions within a PEP.

There are no properties to set for this marker.

External Access Category

In the External Access Category, there are two Bladelets:

- Access HTTP, page 2-10
- Access DB, page 2-17

Access HTTP

Summary

The Access HTTP Bladelet makes an outgoing HTTP call using the GET or POST method in either the Componentized or Normal URL Configuration groups.

Prerequisites and Dependencies

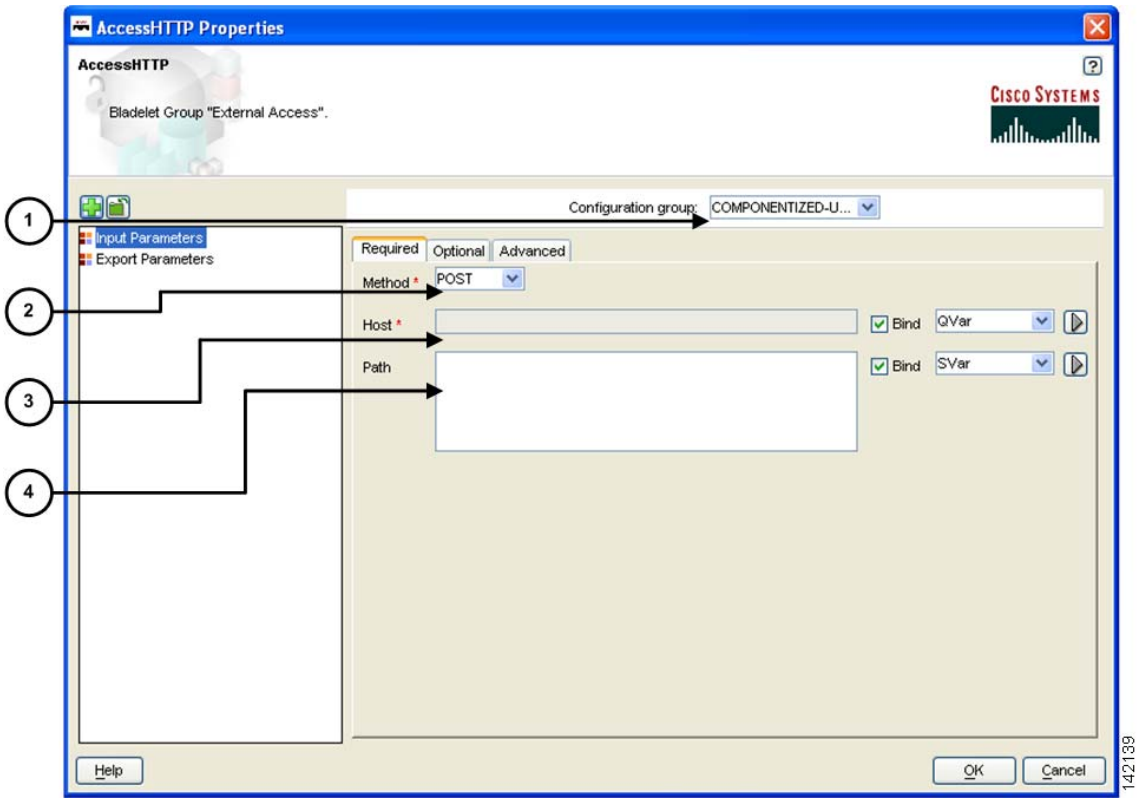
None.

Details

Figure 2-9 to Figure 2-11 show required, optional, and advanced settings for the Componentized URL Configuration group.

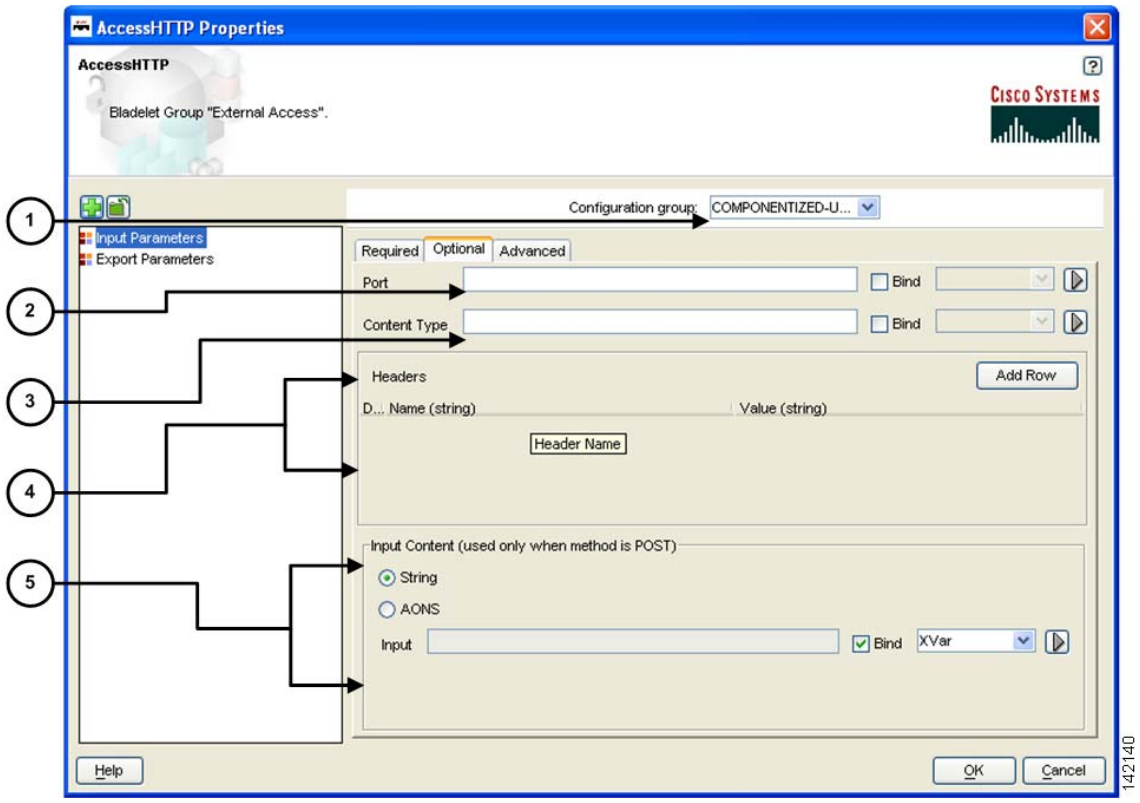
Figure 2-12 to Figure 2-14 show required, optional, and advanced settings for the Normal URL Configuration group.

Figure 2-9 Access HTTP Properties Window—Input Parameters, Componentized URL, Required Tab



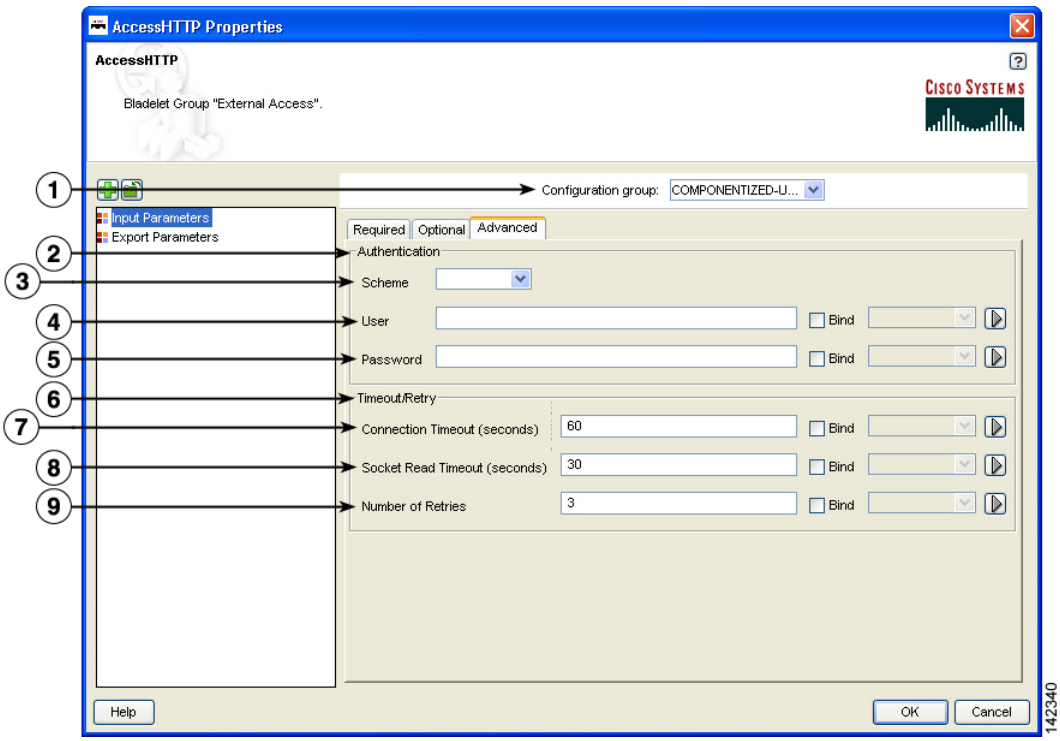
1	Configuration group	Configuration group, set here to Componentized URL.
2	Method	Method. Choices: Post and Get.
3	Host	Hostname or IP address of the HTTP server.
4	Path	Path portion of the URL (/index.jsp).

Figure 2-10 Access HTTP Properties Window—Input Parameters, Componentized URL, Optional Tab



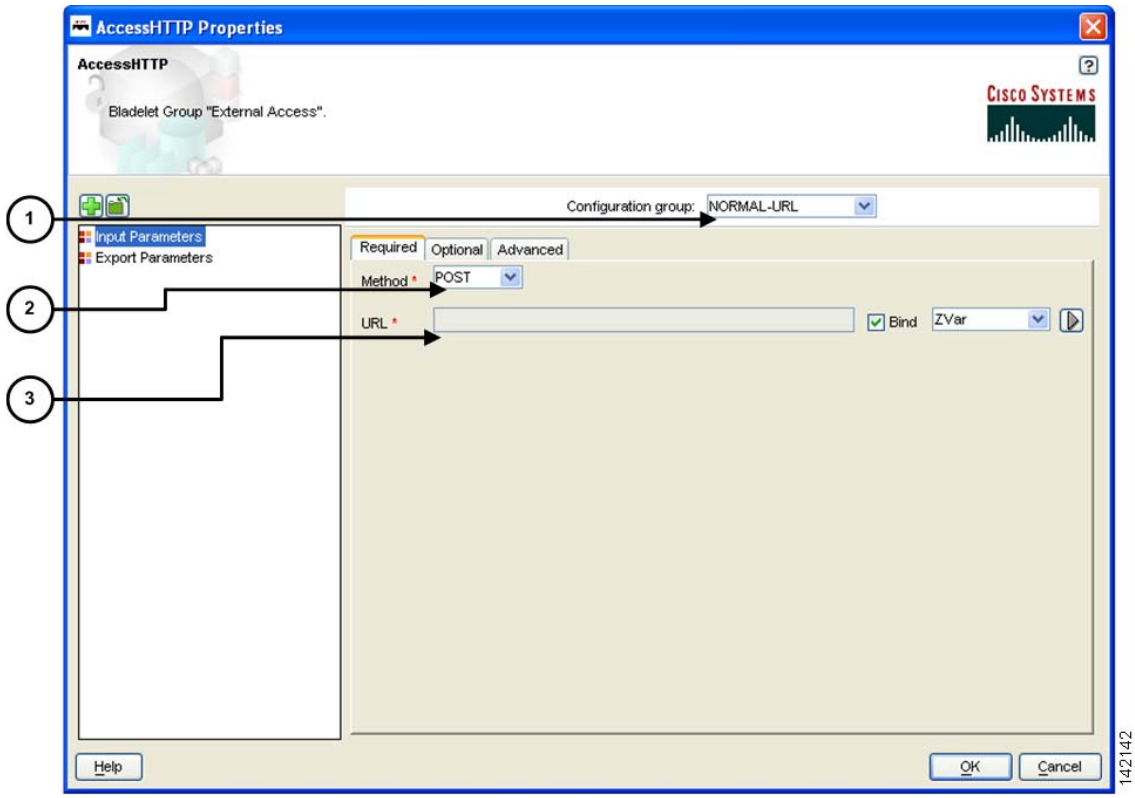
1	Configuration Group	Configuration group, set here to Componentized URL.
2	Port	Port number to be used. Defaults to 80.
3	Content Type	MIME type of the content.
4	Headers	Header name and corresponding value (string types).
5	Input Content	Payload of the HTTP call. Required only in case of POST.

Figure 2-11 Access HTTP Properties Window—Input Parameters, Componentized URL, Advanced Tab



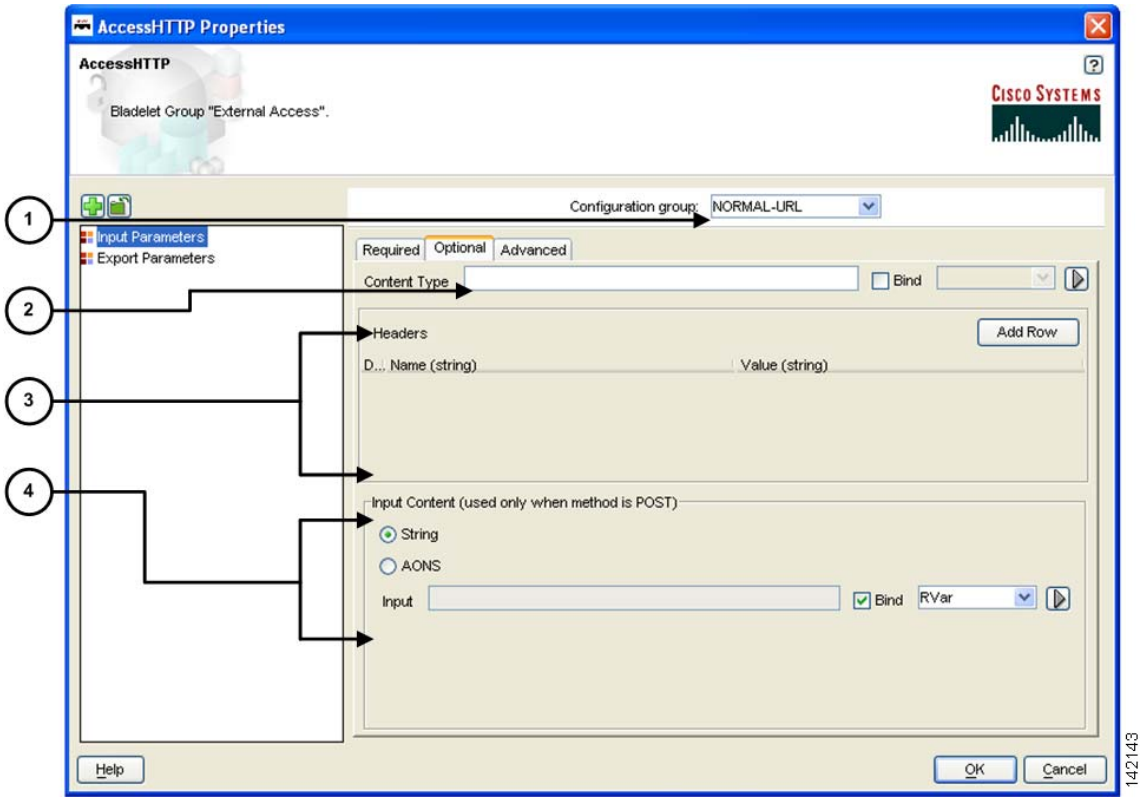
1	Configuration Group	Configuration group, set here to Componentized URL
2	Authentication	Basic HTTP is the only authentication scheme supported today.
3	Scheme	Basic HTTP.
4	User	User ID.
5	Password	Password.
6	Timeout/Retires	Timeout and retries to establish a connection.
7	Connection Timeout (seconds)	The maximum amount of time in seconds, for which AccessHttp waits to open a connection. Default is 60 seconds.
8	Socket Read Timeout (seconds)	The maximum amount of time in seconds for which AccessHttp waits to read from the socket after a connection is established. Default is 30 seconds.
9	Number of Retries	The number of times a connection is attempted. Default is 3.

Figure 2-12 Access HTTP Properties Window—Input Parameters, Normal URL, Required Tab



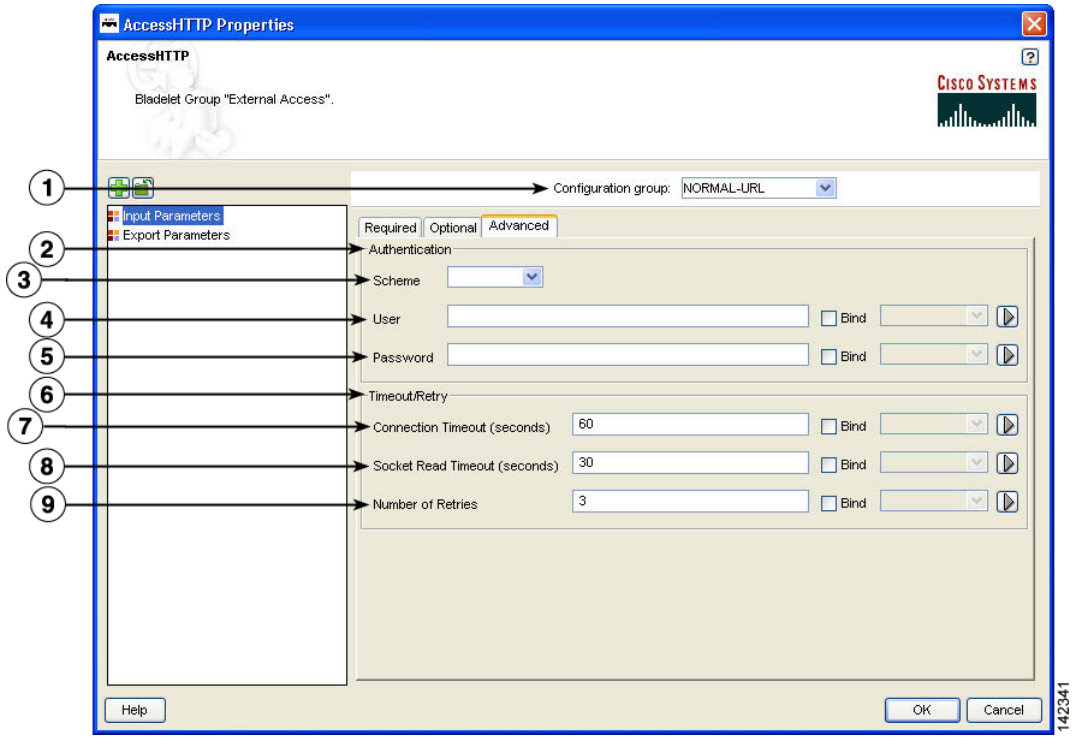
1	Configuration Group	Configuration group, set here to Normal URL.
2	Method	Method. Choices: POST or Get.
3	URL	Complete URL.

Figure 2-13 Access HTTP Properties Window—Input Parameters, Normal URL, Optional Tab



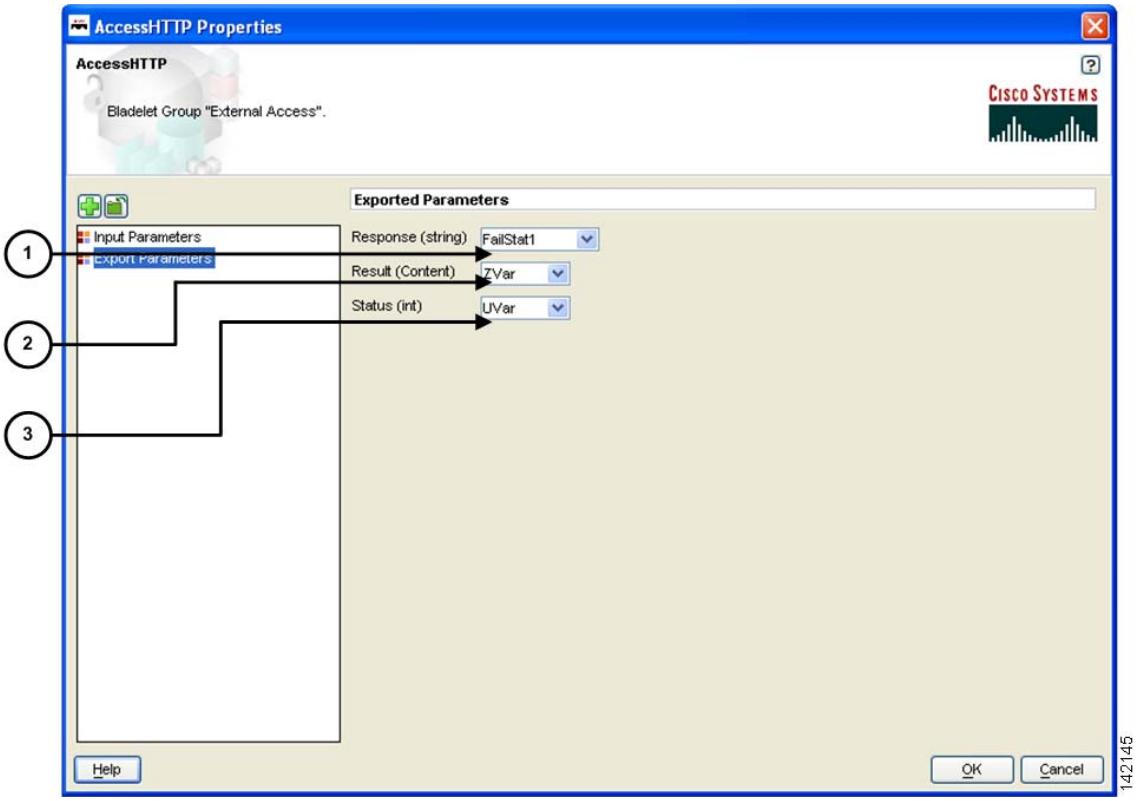
1	Configuration group	Configuration group, set here to Normal URL.
2	Content Type	MIME type of the content.
3	Headers	Header name and corresponding value.
4	Input Content	Payload of the HTTP call. Required only in case of Post.

Figure 2-14 Access HTTP Properties Window—Input Parameters, Normal URL, Advanced Tab



1	Configuration Group	Configuration group, set here to Normal URL.
2	Authentication	Authentication scheme. Basic is the only scheme supported today.
3	Scheme	Basic HTTP.
4	User	User ID.
5	Password	Password.
6	Timeout/Retires	Timeout and retries to establish a connection.
7	Connection Timeout (seconds)	The maximum amount of time in seconds, for which AccessHttp waits to open a connection. Default is 60 seconds.
8	Socket Read Timeout (seconds)	The maximum amount of time in seconds for which AccessHttp waits to read from the socket after a connection is established. Default is 30 seconds.
9	Number of Retries	The number of times a connection is attempted. Default is 3.

Figure 2-15 Access HTTP Properties Window—Export Parameters



1	Response	Response from the HTTP call (string type).
2	Result	Response from the HTTP call (AON content type).
3	Status	Status HTTP call (integer type m).

Outcome

None.

Exceptions

- Malformed URL: Connection cannot be established to the HTTP server host.
- Host Inaccessible: The URL (composed URL in case componentized URL is specified) is not correct.

Access DB

Summary

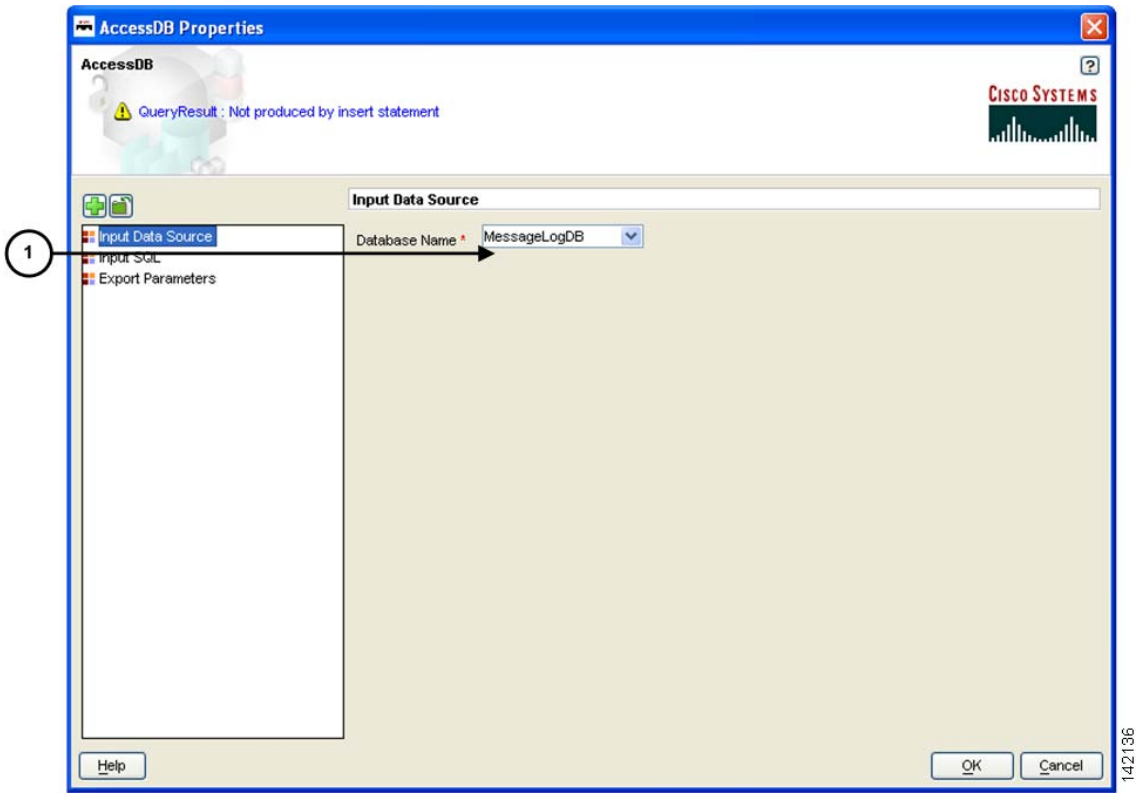
Use this Bladelet to make a SQL call out to a database.

Prerequisites and Dependencies

None.

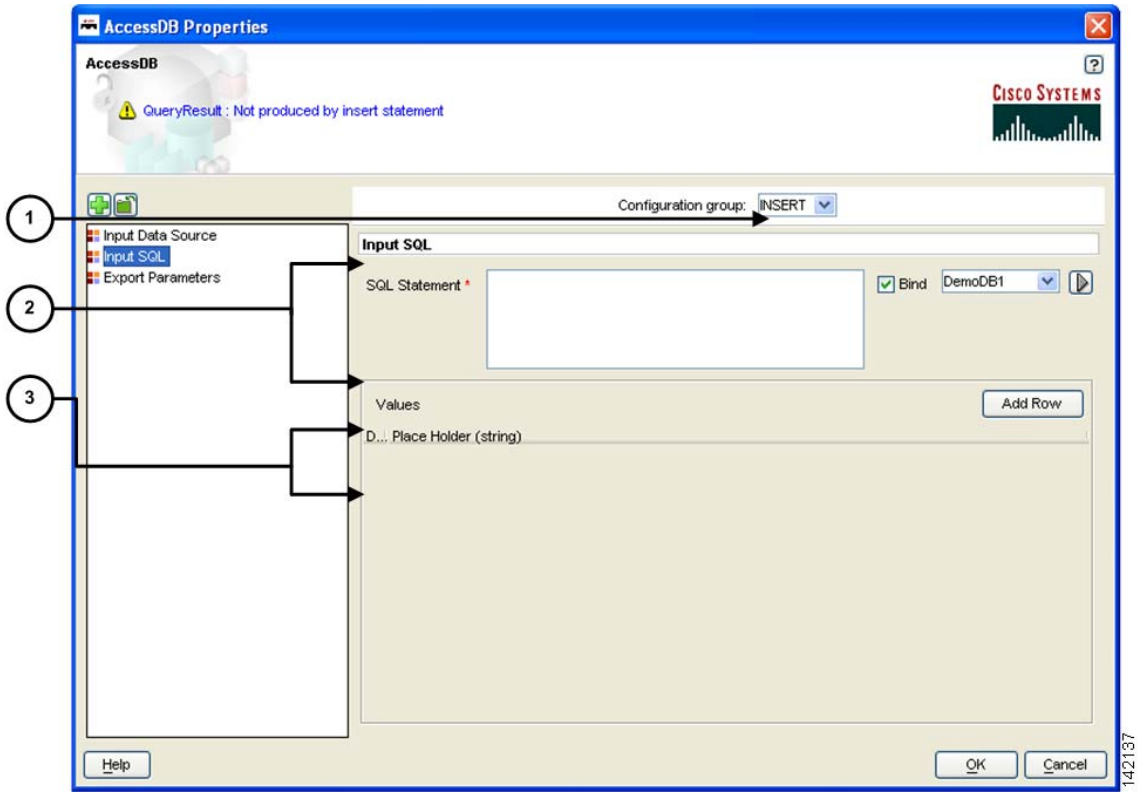
Details

Figure 2-16 Access DB Properties Window—Input Data Source



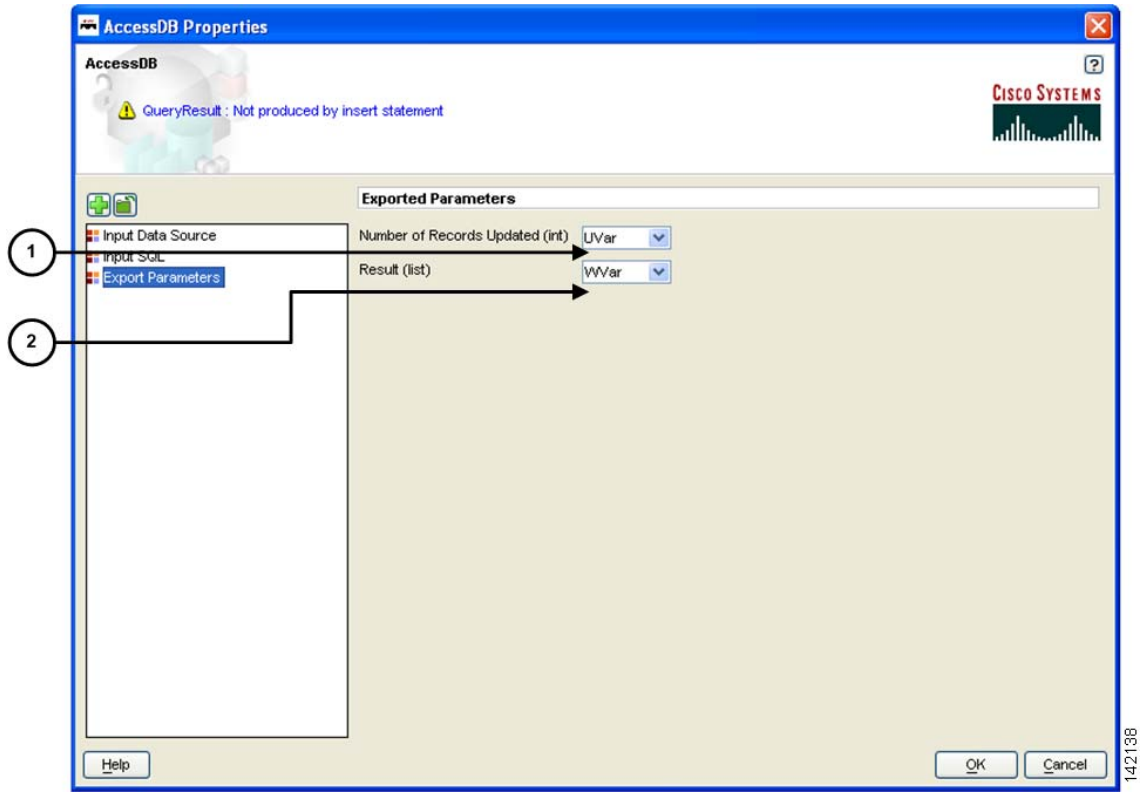
1	Database Name	Property set. Must already be configured on the AMC server. Required. Full path in AMC is **Properties > Application > Global > Databases**.

Figure 2-17 Access DB Properties Window—Input SQL



1	Configuration Group	Configuration group, set here to Insert. Choices: Insert, Update, Delete, and Query.
2	SQL Statement	The SQL statement in the Java SQL prepared statement syntax. Use ? for place holders. Do not put ? in quotes in case of string-type parameters.
3	Values	One or more values (string types). Each string corresponds to the placeholder in the SQL statement. There should be as many entries in this list as there are placeholders in the SQL statement.

Figure 2-18 Access DB Properties Window—Export Parameters



1	Number of Records Updated	Number of records updated in case of non-query type of SQL statements.
2	Result	Result set in case type of SQL statements is Query. There are as many maps in the list as there are records retrieved. Each map has name-value pairs, where name is the column name and value is the column value in the record.

Outcome

None.

Exceptions

- Database Failure: Database cannot be connected to.
- SQL Failure: The input SQL statement could not be executed properly.

Note The SQL interface does not support stored procedures.

General Category

In the General Category, there are three Bladelets:

- Log, page 2-21
- Retrieve Cache, page 2-24

- Cache Data, page 2-27

Log

Summary

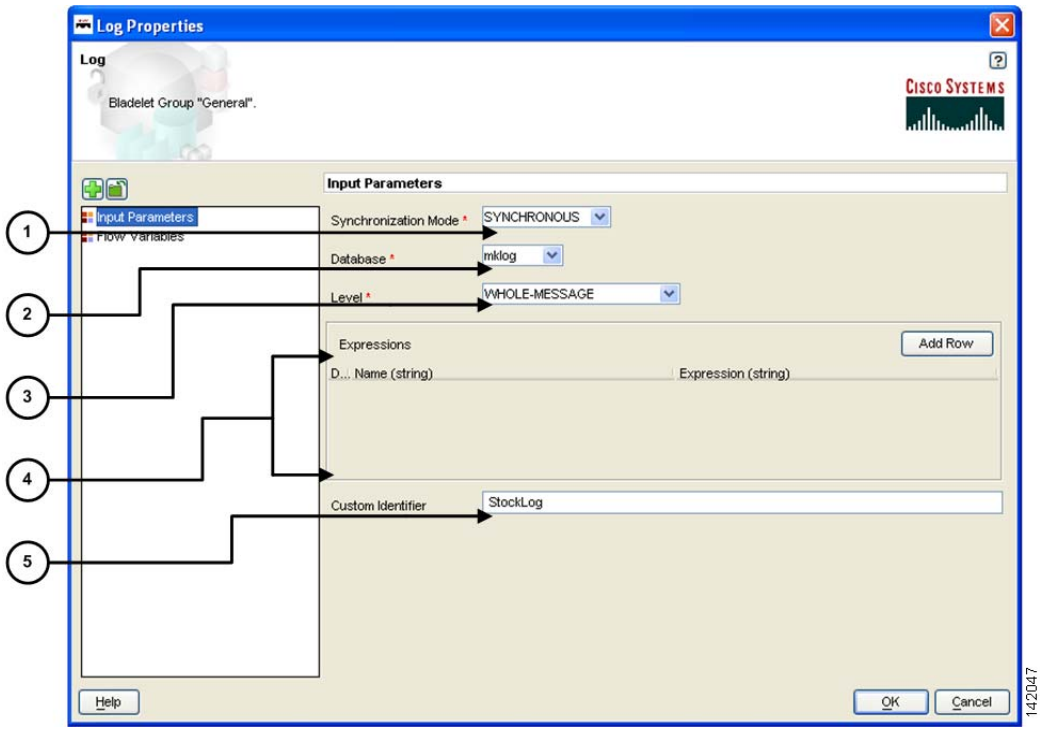
The Log Bladelet allows you to log message contents, PEP variables and other important data related to the message, message class, source, destination, time stamps, and PEP name.

Prerequisites and Dependencies

None.

Details

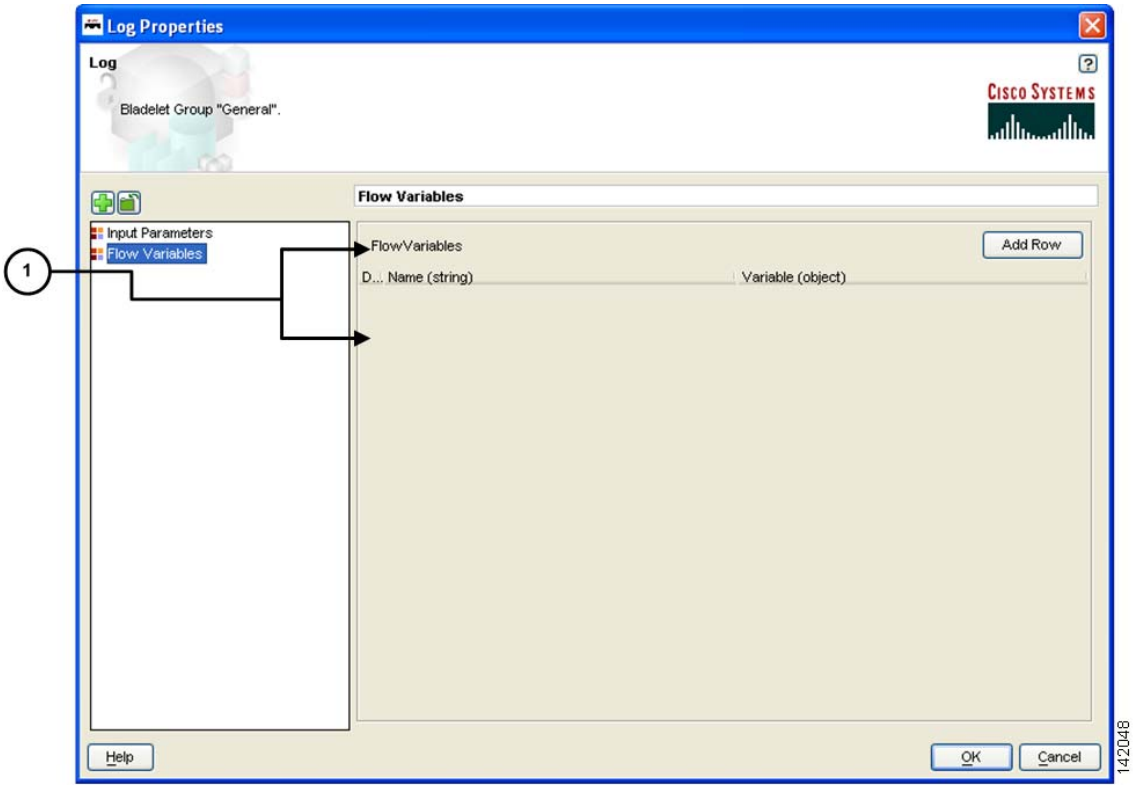
Figure 2-19 Log Properties Window—Input Parameters



1	Synchronization Mode	Mode of operation: • Asynchronous—Bladelet executes (in the foreground) while the database writes (in the background). • Synchronous—Bladelet waits while the database writes. Use to ensure that data is entered into the database properly before the PEP goes to the next step.
2	Database	Property set names for Message Log Policy. Must already be configured on the AMC server. Full path in AMC is **Properties > Application > Node > Message Log Domain**.

3	Level	Level of logging. Allowed values for the ENUM are the following: • Basic—Only metadata about the message is logged: entry time, message type, PEP name, and so on. • Header—Basic plus SOAP header. For non-SOAP messages, it is the same as Basic. • Body—Basic plus SOAP body. For non-SOAP messages, it is the whole message. • Whole-Message—Entire message without attachment. For non-SOAP message, it is the same as Body. • Specify by XPath Expressions—Contents to be logged are specified by a list of XPath expressions. (See descriptions for the Expressions parameter.)
4	Expressions	Optional. One or more XPaths specifying what needs to be logged. Applies only if level is set to Specify by XPath Expressions. Each XPath contains two values: • Name—String that provides a unique identifier for the contents specified by the expression • Expression—Valid XPath expression specifying the contents that need to be extracted and logged
5	Custom Identifier	Optional. String to identify this message log entry.

Figure 2-20 Log Properties Window—PEP Variables



1	PEP Variables	List of variables to be logged. Each list element contains two values: • Name—Unique identifier for the contents of the variable • Variable—Top-level variable or valid variable expression (select from the drop-down list or use the variable picker)

Outcome

None.

Exceptions

- Log Write Failure: A failure occurred during the database write. These are failures that are typically not recoverable. For example, data does not conform to the log schema, or the log policy is disabled for the database.
- Timeout: Timeout occurred. This applies only to synchronous mode. For example, this can happen when the database is not available or is extremely slow.

Retrieve Cache

Summary

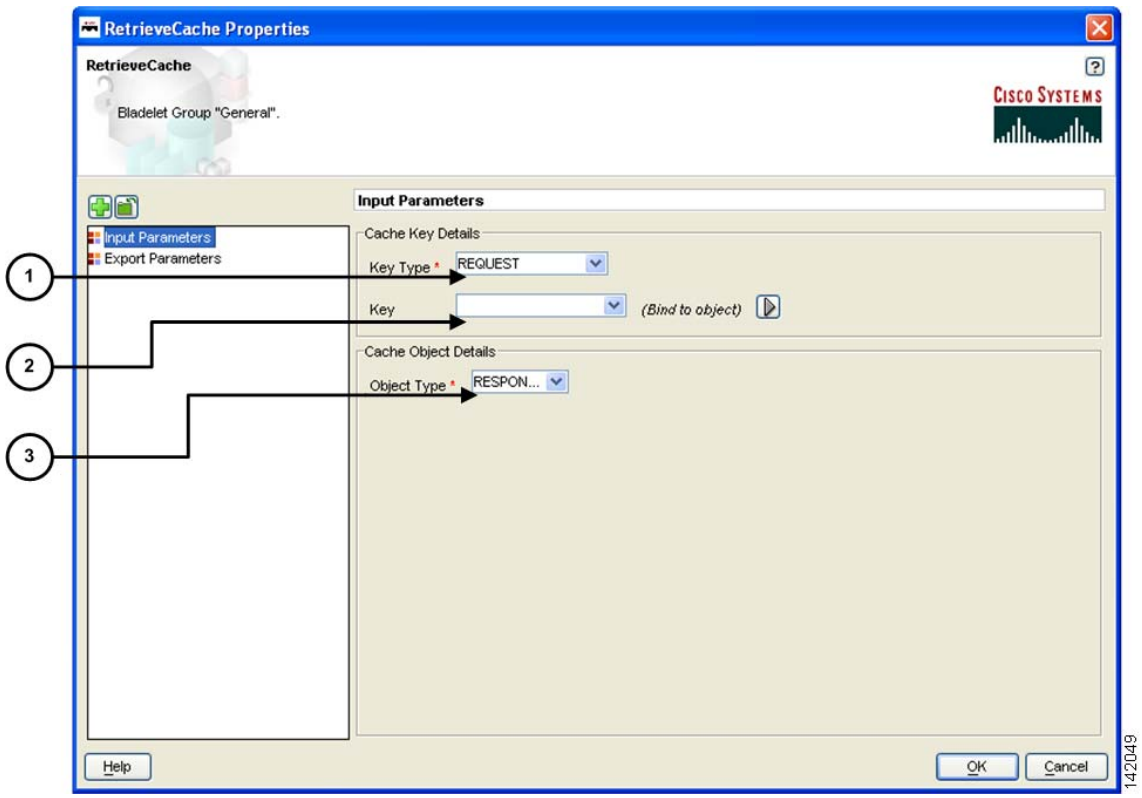
This Bladelet retrieves data from two named caches configured on an AON node. The named caches are response and variable. The response cache caches server responses. The variable cache caches PEP variables. Populate these named caches by using the CacheData Bladelet. Populate the variable cache by using the Caching Service API exposed to custom Bladelets.

Prerequisites and Dependencies

- Ensure that the cache on the AON node on which the PEP executes has bootstrapped without errors.

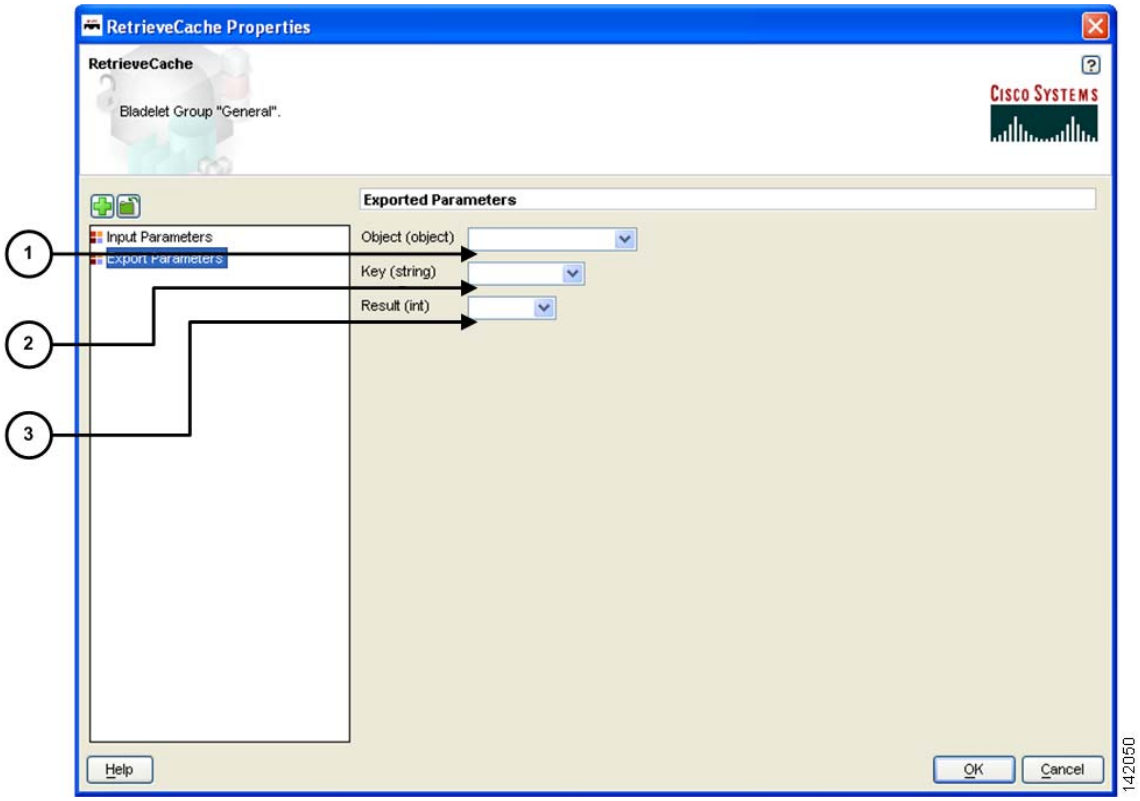
Details

Figure 2-21 Retrieve Cache Properties Window—Input Parameters



1	Key Type	Hint to the Bladelet to determine the Key to be used for retrieving the object from the cache. • Request—Bladelet computes the cache key from the payload of the current request message. • HTTP-Request-URI—Bladelet inspects the HTTP request and uses the request uniform resource identifier (URI) as the cache key. Use only if the request message is HTTP. • Variable—Bladelet uses a PEP variable as the cache key.
2	Key	Key. Required if the key type is VARIABLE. Bind to this input parameter. Can be one of the following data types: string, FindResult, or any numeric type. For Request and HTTP-Request-URI, the key is ignored.
3	Object Type	Where the Bladelet should go to fetch the data: • Response—Response cache • Variable—Variable cache

Figure 2-22 Retrieve Cache Properties Window—Export Parameters



1	Object	Exported parameter object. Bind the object retrieved from the cache to this object.
2	Key	Exported parameter key. Required if the key type is Variable. Bind the PEP variable to be used as the key to this input parameter. The variable can be one of the following data types: string, FindResult, or any numeric type.
3	Result	Expected result of export parameter. Bind the result of the cache lookup to this variable.

Outcome

- A cache hit or "Success" path indicates that the requested data was found in the cache.
- A cache miss or "Fail" path indicates that the requested data was not found in the cache.

The Bladelet exports the cache key that was used for the lookup operation, the result of the operation (0 indicates a miss; 1 indicates a hit) as follows:

- On success, it also exports the cached object, which can be bound to a PEP variable of the appropriate data type. For any object retrieved from the "response" named cache, the Bladelet also binds the object to the "RESPONSE_MESSAGE" PEP variable.
- On miss, the exported cache key can be used by a CacheData Bladelet further in the PEP execution to cache data to the cache.

Exceptions

None.

Cache Data

Summary

This Bladelet should be used to set data into the named caches configured on an AON node. The named caches are "response" and "variable". The "response" cache caches server responses. The "variable" cache caches PEP variables. You can retrieve data from the named caches by using the RetrieveCache Bladelet. In addition, you can retrieve data from the "variable" cache by using the Caching Service API exposed to Custom Bladelets.

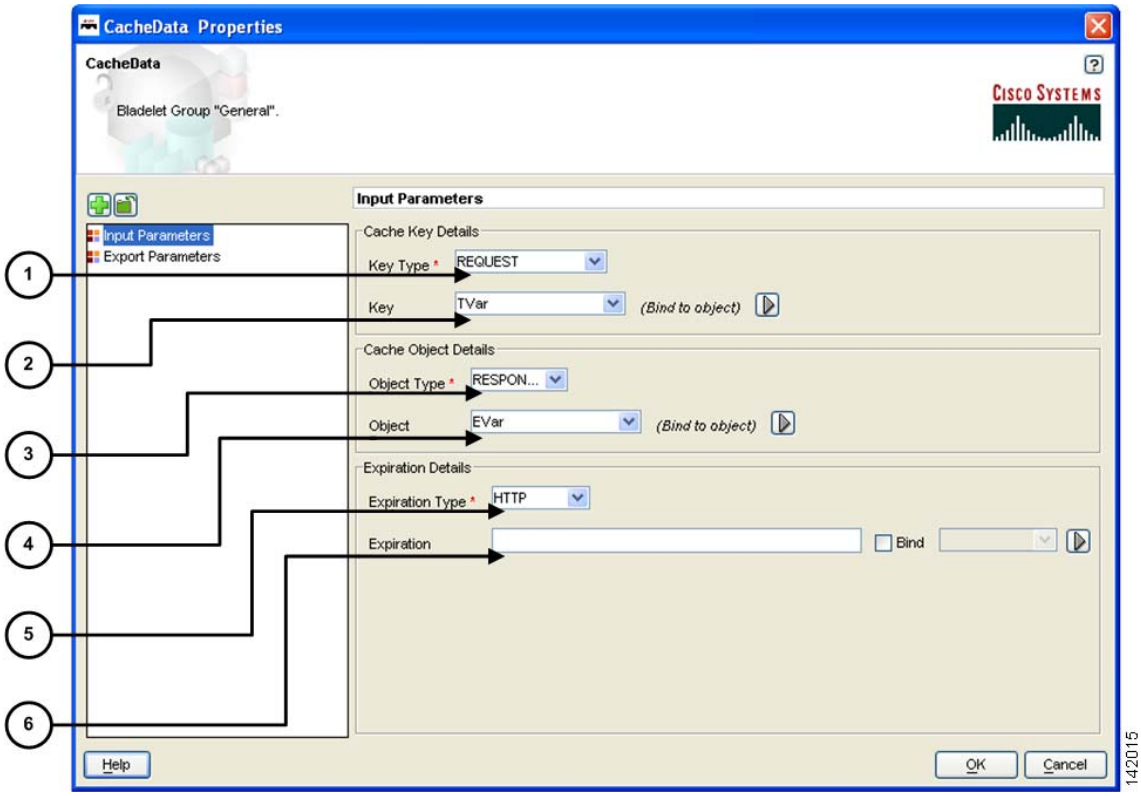
Prerequisites and Dependencies

- Ensure that the cache on the AON node on which the PEP executes has bootstrapped without errors.

Details

When it is given a cache key and optionally a PEP variable, this Bladelet caches the variable or the server response message.

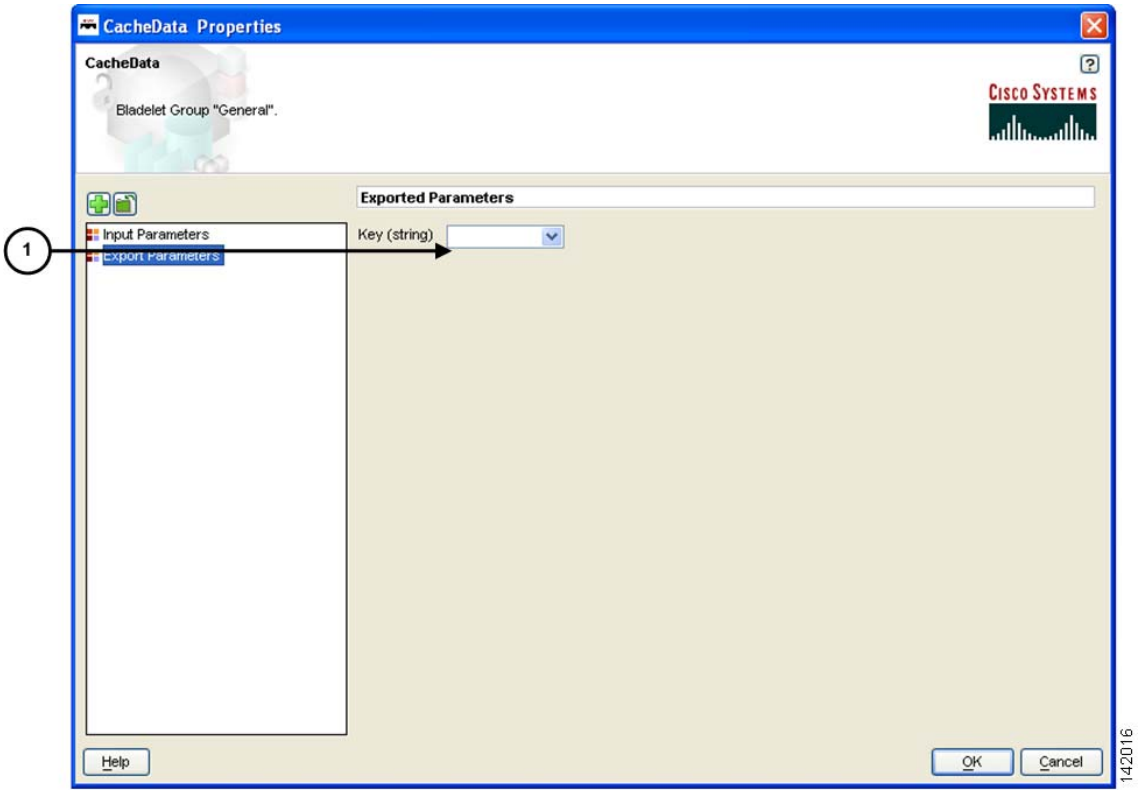
Figure 2-23 Cache Data Properties Window—Input Parameters



1	Key Type	Hint to the Bladelet to determine the key to be used for setting the object to the cache. • Request—Bladelet computes the cache key from the payload of the current request message. • HTTP-Request-URI—Bladelet inspects the HTTP request and uses the request URI as the cache key. Use only if the request message is HTTP. • Variable—Bladelet uses a PEP variable as the cache key.
2	Key	PEP variable. Required if the key type is VARIABLE. Bind the PEP variable to be used as the key to this input parameter. Can be one of the following data types: string, FindResult or any numeric type. For Request and HTTP-Request-URI the key is ignored.
3	Object Type	Whether or not the data to be cached should be the server response elicited by the request or a PEP variable. • Response—Caches the current response message in the response cache. • Variable—Caches the PEP variable specified in the Object parameter.

4	Object	Value of the PEP input variable. If the object type is VARIABLE, bind the PEP variable to be cached to this object.
5	Expiration Type	How to determine the time to live or object expiration. • Relative—Expiration time is specified as a relative integer value denoting the number of seconds for which the object should be cached. • Absolute—An absolute time for which the object should be cached. • HTTP—Use the HTTP directives and headers to compute the time to live. • Default—Use the default TTL specified in the caching policy on the AMC server. For the response cache, use the response-cache default TTL. For the variable cache, use the variable-cache default TTL.
6	Expiration	Actual time for which the object should be cached. Required only for relative and absolute expiration types. • For relative, specify an integer. –1 indicates that the object should be cached forever. • For absolute, specify a date in the following format: `EEE, dd MMM yyyy HH:mm:ss GMT'` Example: Sun, 16 Nov 2003 22:00:00 GMT Optionally, specify by binding to a PEP variable that contains the expiration value. Expiration is ignored for HTTP and Default expiration dates.

Figure 2-24 Cache Data Properties Window—Export Parameters



1	Key	Exported key parameter. Required if the key type is Variable. Bind the PEP variable to be used as the key to this input parameter. The variable can be one of the following data types: string, FindResult, or any numeric type.

Outcome

- On success, the server response elicited by the request of the PEP variable to be cached is set in the "response" and "variable" cache.

Exceptions

None.

Logic Category

In the Logic Category, there are two Bladelets:

- Find, page 2-31
- Branch, page 2-34

Find

Summary

The Find Bladelet queries an XML message and extracts all nodes identified by regular (for regular expressions, the message type does not need to be in XML format) and XPath expressions from the message currently being processed by the PEP. After regular and common XPath expressions are evaluated by this Bladelet, they are available for use by other Bladelets. Either XPath or Regex expressions can be evaluated; if both need to be evaluated, you must incorporate multiple instances of the Find Bladelet.

Prerequisites and Dependencies

None.

Details

The Find Bladelet finds multiple items from within the message using XPath expressions (for XML messages) or Regular Expressions for Non-XML messages. It works on both MIME as well as NON-MIME data. The output of the find Bladelet is placed inside a PEP variable of type FindResultMapListIterator. This data type is a complex data type that encapsulates results that are found from all parts (> 0 if MIME) of the message that is being searched. The structure of the data type is as follows:

FindResultMapListIterator:

List of parts of the message on which the Find Bladelet operates (List of size 1 containing the results if it is Non-MIME; List of size > 1 if more than 1 MIME part is in the message)

Map of all the different expressions that were searched (recall that you enter a value on the left-hand list box in the Find Bladelet and for each of these you specify a list of expressions on the right-hand table. The map contains key-value pairs with the key being the entries on the left-hand box and the value being a list (size of this list = number of expressions entered for each key). The elements in this list are the actual search results.

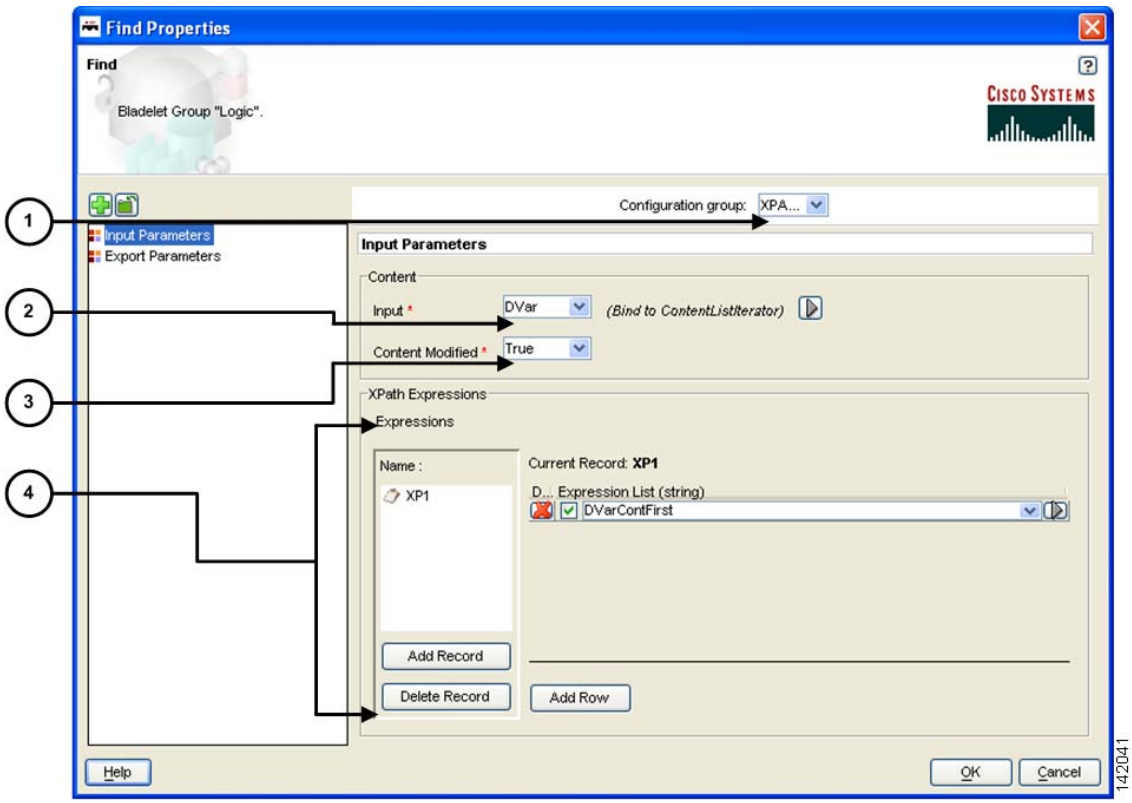
Note IMPORTANT: Today it is not possible to use the PEP variable-picker dialog to select values from the tree view. **You must enter a specific value to extract the returned results.**

```
Example:
Key (Left hand side list box)           XPath Expression list
k1                                                   e1
                                                       e2
k2                                                   e3

Assume a regular input message (NON-MIME).  The way to extract the results are (assume
that the output of Find is bound to a PEP variable called findResults) in the Specify
Value text box of the PEP Variable Picker dialog type:
    findResults.elementAt(0).elementAt(k1).elementAt(0).value()
This expression returns the value of the search result using expression e1 on the message
while
    findResults.elementAt(0).elementAt(k1).elementAt(1).value()
gives the value of the search results for e2.
The value() function is used if you know your xpath result is of type boolean, string,
integer; or if you want only the string value of the first node in the XPath Result (which
is a nodeSet)
If the XPath result of e3 is known to be a nodeset, then to get e3 result's 2nd node's
string value:
    findResults.elementAt(0).elementAt(k2).elementAt(2).nodeValue(1).
```

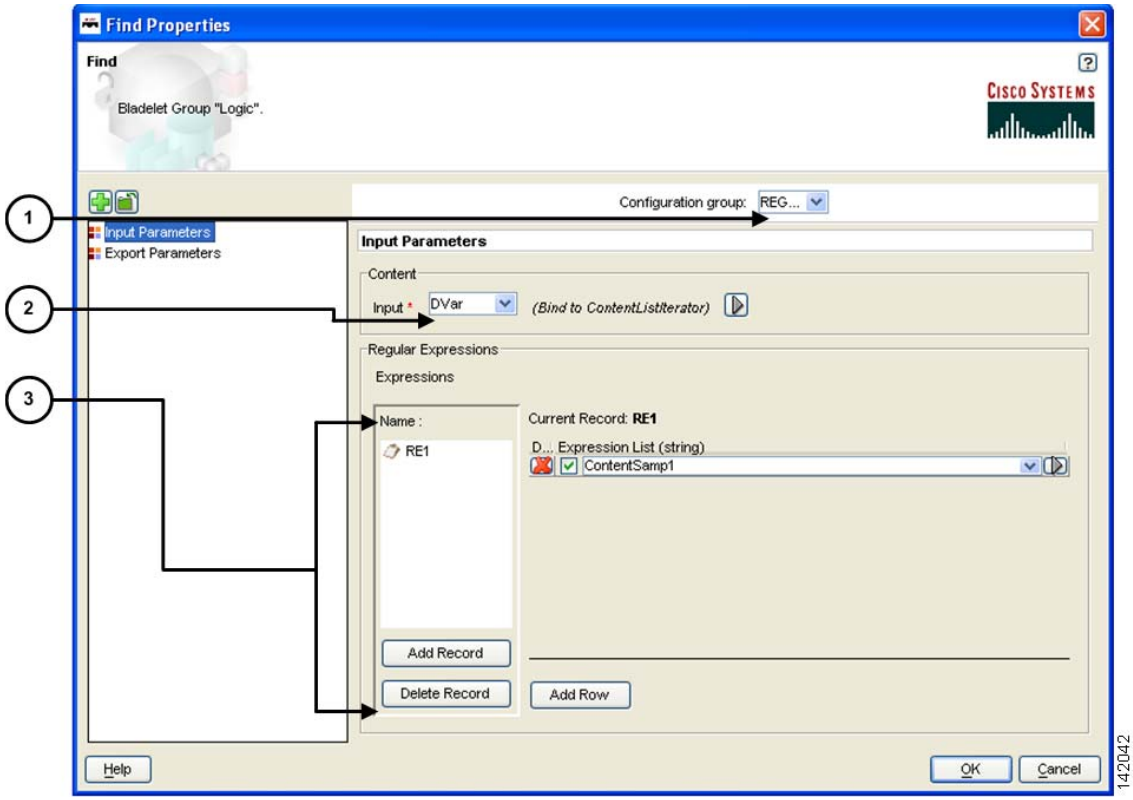
The input parameters for this Bladelet (configuration group is set to XPath) are shown in Figure 2-25. Input parameters for a Bladelet whose configuration group is set to Regex are shown in Figure 2-26.

Figure 2-25 Find Properties Window—Input Parameters, XPath



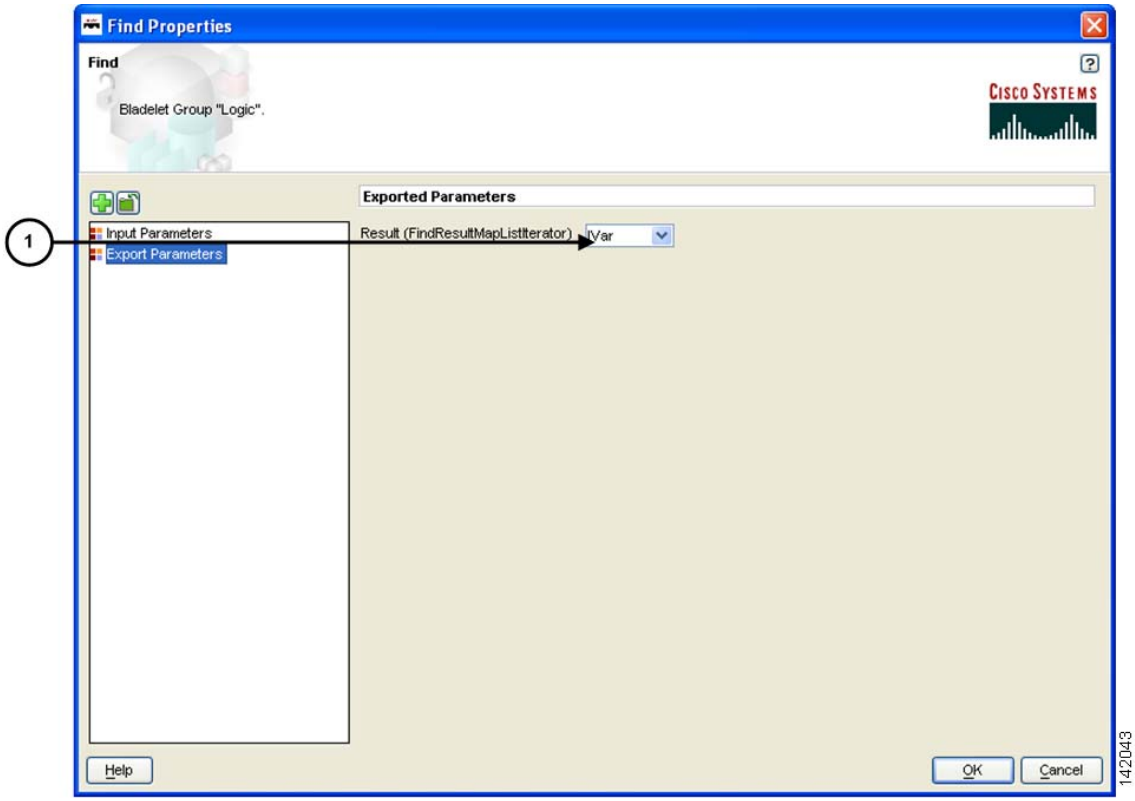
1	Configuration Group	Configuration group, set here to XPath. Valid values are XPath and Regex.
2	Input	Input, such as DVar.
3	Content Modified	Whether or not message content has been modified—for example, by a preceding encryption Bladelet or transformation Bladelet. If this is the first Find Bladelet in the PEP, then this parameter is always true because, to this Bladelet, every message is a new message.
4	Xpath Expressions	XPath expressions under which the condition is evaluated. Add one or more records and at least one row for each record added with an expression list in string format.

Figure 2-26 Find Properties Window—Input Parameters, Regex



1	Configuration Group	Configuration group, set here to Regex. Valid values are XPath and Regex.
2	Input	Content input parameter such as DVar.
3	Regular Expressions	Any number of regular expressions, such as a sample. Add records with one or more rows of expression lists to be evaluated.

Figure 2-27 Find Properties Window—Export Parameters



1	Result	Result to be exported. Export parameter result to a variable such as IVar. If no PEP variable is available in the list, add one without exiting the properties window as described in the "Bladelet Properties Window and Dialog Boxes" section on page 2-3.

Outcome

- If all expressions in the Find Bladelet are evaluated to null, the output path is set to Fail.
- If any expression is evaluated to other than null, the output path is set to success. On success, a PEP variable of type FindResultMapListIterator is exported for use by other Bladelets in the PEP.

Exceptions

- Invalid Content Type: The content type is invalid for evaluation. This happens when expression type is XPath while the message is NOT XML documents.

Branch

Summary

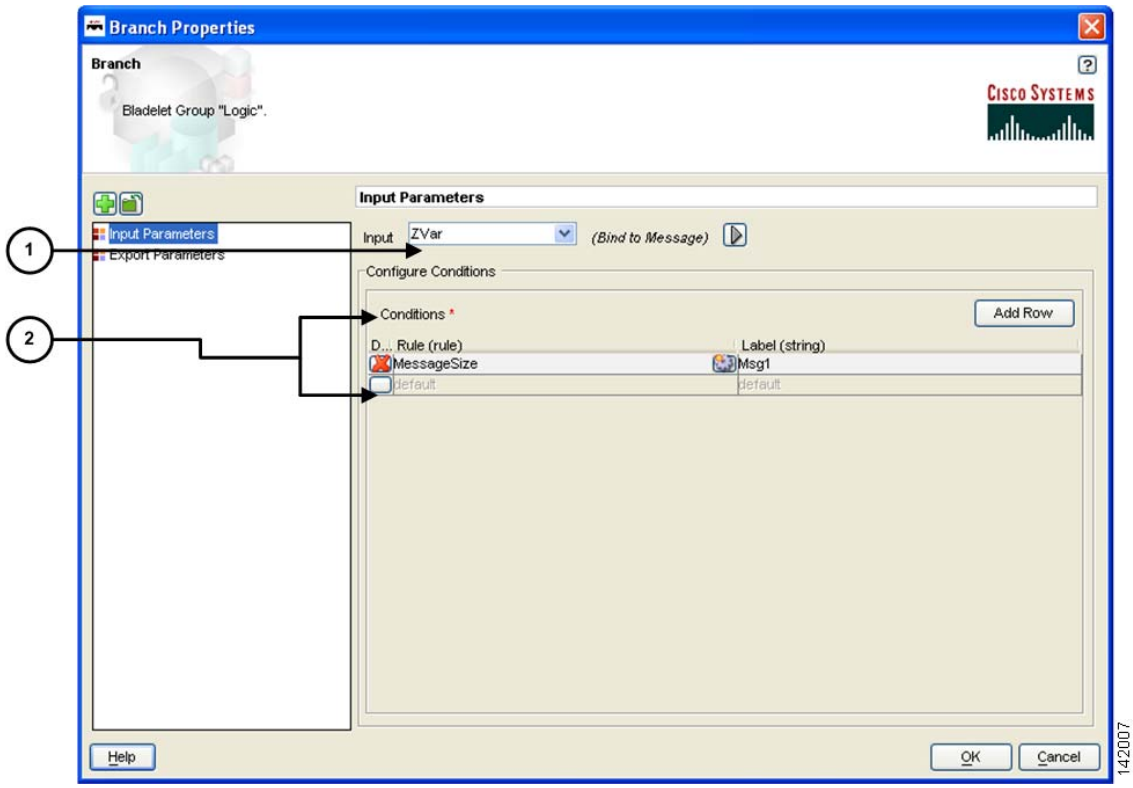
This Bladelet establishes conditions for message route branching based on rules and message labels. There are two main sections in the Branch Properties window.

Prerequisites and Dependencies

None.

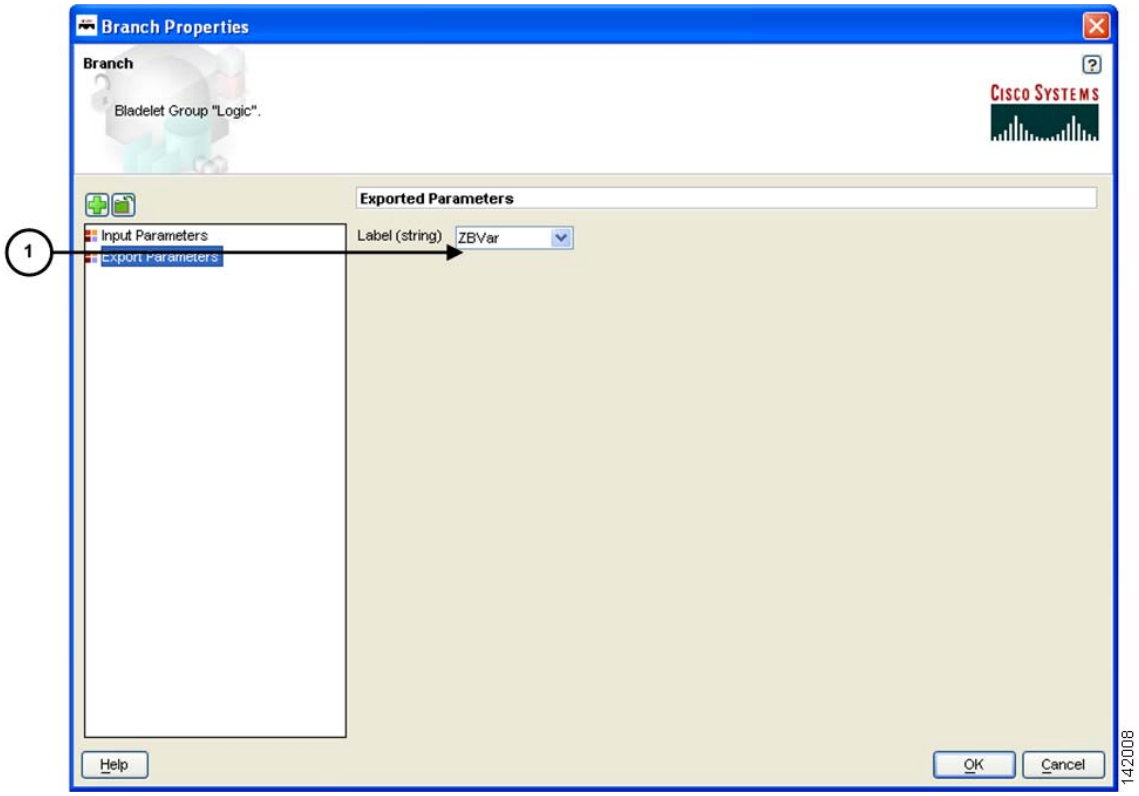
Details

Figure 2-28 Branch Properties Window—Input Parameters



1	Input	Input message. If not specified, the message used is based on the position of the Bladelet. If the Bladelet is placed before the response marker, then REQUEST_MESSAGE is used. If the Bladelet is placed after response marker, then RESPONSE_MESSAGE is used.
2	Conditions	Rules and labels. Each rule is evaluated in the order it is specified; evaluation stops at the first rule that evaluates to true. The label corresponding to that particular rule is set as the output path of this Bladelet. If none of the rules evaluates to true, the default output port is activated.

Figure 2-29 Branch Properties Window—Export Parameters



1	Label	Label that is chosen as the output path.

Outcome

- On success, the output port activated is the same as the one corresponding to the rule that evaluates to true.
- If none of the rules evaluate to true, the default output port is activated.

Exceptions

None.

Message Handling Category

In the Message Handling Category, there are eight Bladelets:

Validate

Summary

The purpose of this Bladelet is to validate XML messages based on a schema (XSD) or DTD. The schema referred by the XML message must already be loaded into AON in an appropriate Schema Extension package using the AMC server.

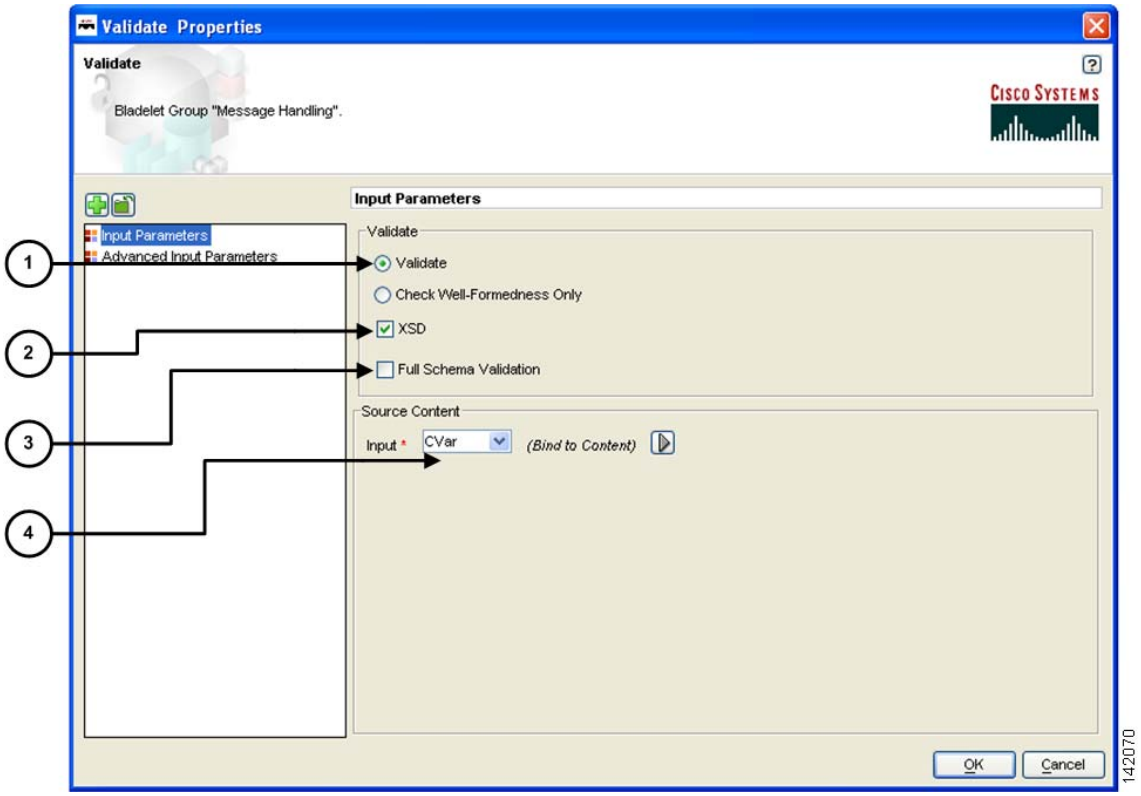
Prerequisites and Dependencies

- Load all schemas including XSD and DTD files that can be referred to by incoming XML messages into AON using the AMC server's Extension-Uploading and Deployment mechanism.
- Configure any Schema Validation policies, if required, and deploy them from the AMC server.

Details

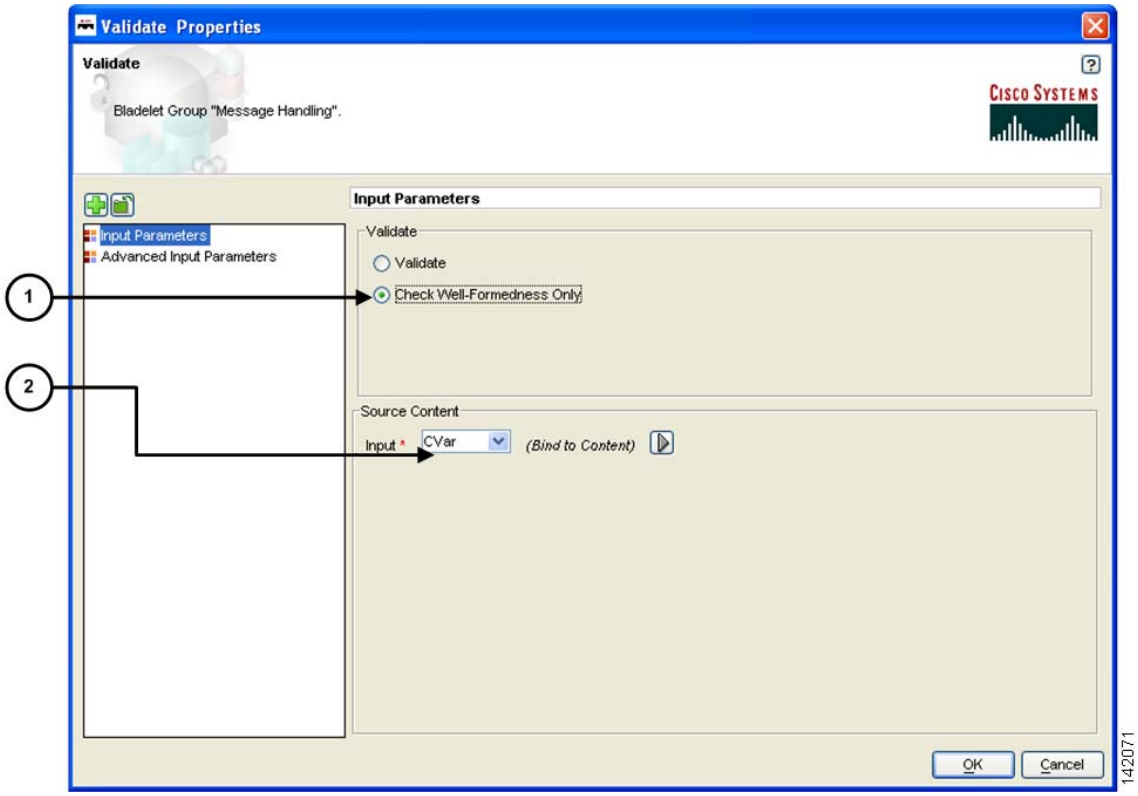
The Validate Bladelet has two main parts in its properties window: input parameters and advanced input parameters.

Figure 2-30 Validate Properties Window—Input Parameters, Validate



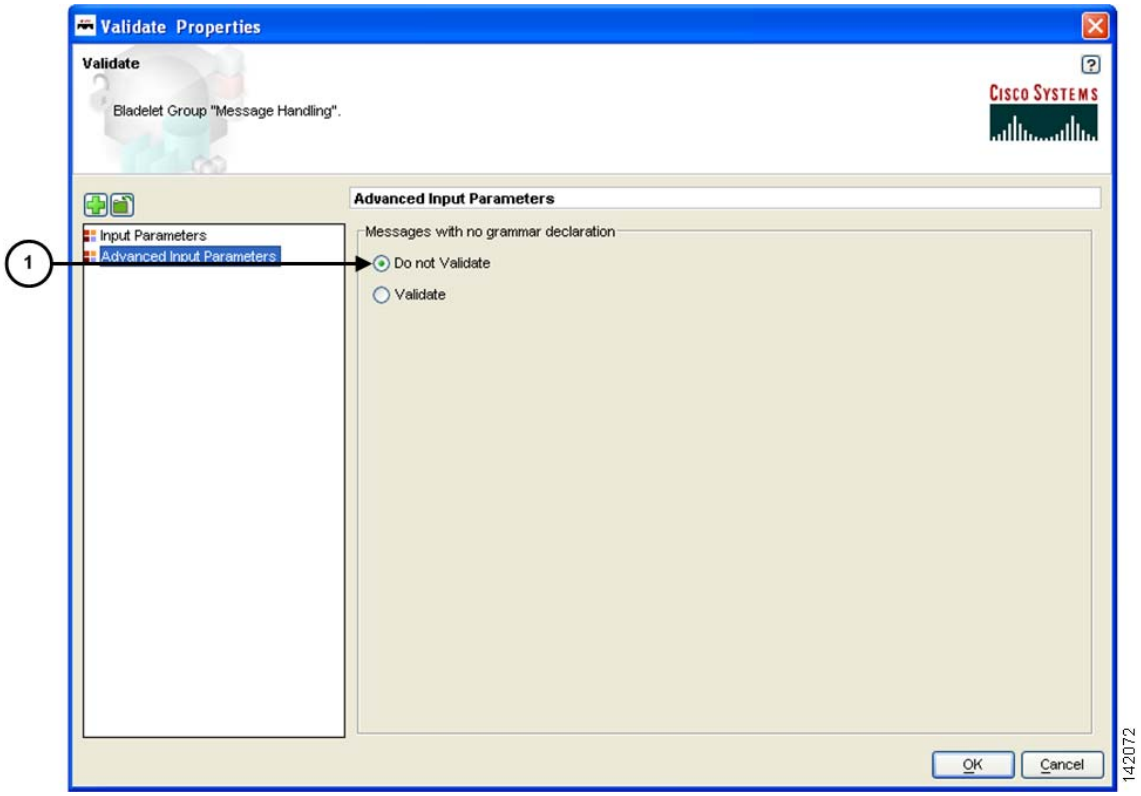
1	Validate	Whether or not to validate messages.
2	XSD	Whether or not to validate XSD in addition to DTD. Check if you expect incoming messages to contain XSD references that need to be validated. If unchecked, XML messages that refer to XSD references are not validated.
3	Full Schema Validation	Whether or not to employ advanced validation features, such as particle derivation restriction checking, that constitute the more time-consuming and memory-intensive aspects of schema validation. If this check box is checked, AON validation occurs. If it is unchecked, AON does not report validation failures. If this and the previous check box are checked, AON reports validation failures against an XSD. If this box is unchecked, it does not validate a message against its XSD.
4	Input	Source-content input. XML message content to be validated by the Bladelet.

Figure 2-31 Validate Properties Window— Input Parameters, Check Well-Formedness Only



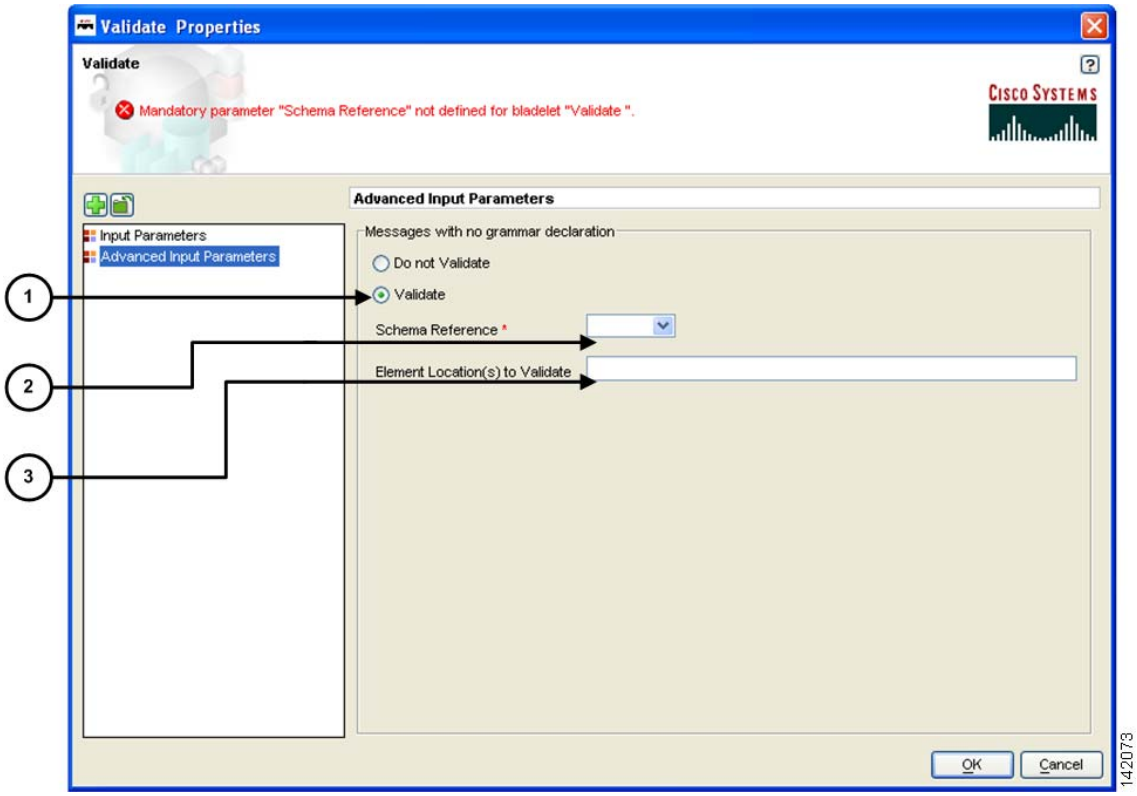
1	Check Well-Formedness Only	Whether or not to ensure the well-formedness of the input XML message. The XML message is not validated using any XSD or DTD, even if it refers to one.
2	Input	Source-content input. XML message content to be validated by the Bladelet.

Figure 2-32 Validate Properties Window—Advanced Input Parameters, Do Not Validate



1	Do Not Validate	Whether or not to refrain from validating input XML messages that do not contain any schema (DTD or XSD) reference.

Figure 2-33 Validate Properties Window—Advanced Input Parameters, Validate



1	Validate	Whether or not to validate input XML messages that do not contain any schema (DTD or XSD) reference. If checked, the following two parameters are mandatory.
2	Schema Reference	Schema validation policy to be enforced on the XML message, if no schema is internally referred in it. Must already be configured on the AMC server.
3	Element Location(s) to Validate	XPath location of any elements that need to be validated. If the whole XML need not be validated, specify an XPath location to validate only elements that are present at the XPath location.

Outcome

- The Success output path is taken when the XML message is found to be valid—that is, it conforms to the XSD or DTD used to validate the message.
- The Failure output path is taken in the following cases:
  - The XML message is found to be invalid—that is, it does not conform to the XSD or DTD used to validate the message.
  - The input message is not a well-formed XML message and therefore could not be validated using any schema.
  - The schema referred by the XML message does not exist in AON.

Exceptions

None.

Build Composite Content

Summary

Creates multipart content from the given input message and the parts that need to be added/deleted/overwritten.

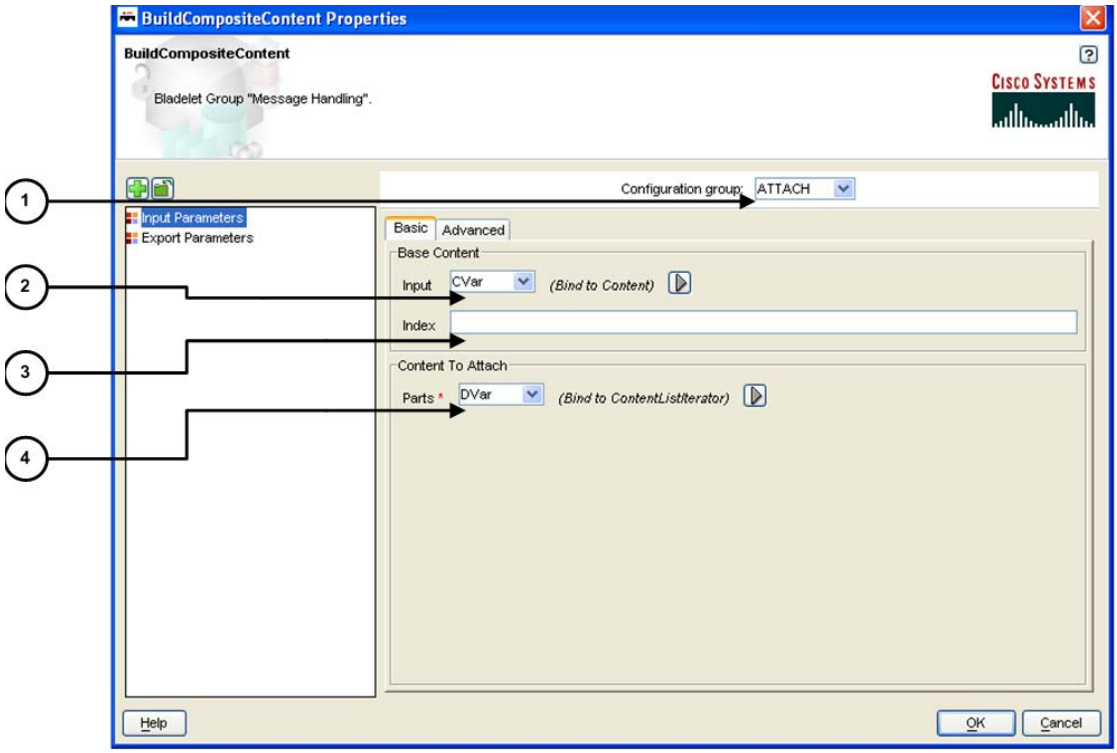
Prerequisites and Dependencies

None.

Details

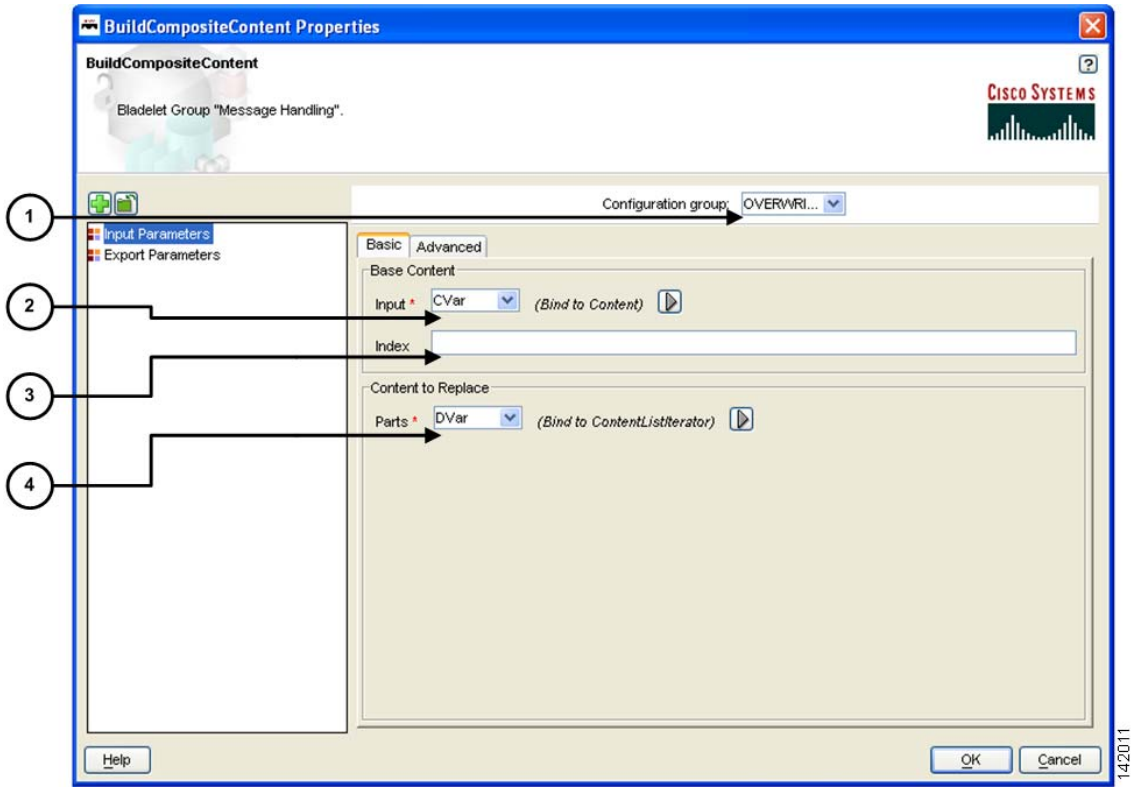
The Build Composite Content Bladelet's properties are, as with some other Bladelets, dependent on the type of configuration group that is used. If the index in the configuration group Attach is null, the Bladelet attaches the parts to the end of the input content. If the index specified is blank in the configuration group Overwrite, it overwrites the Input Content based on the Content-Id of the parts. In configuration group Delete, index and parts are mutually exclusive. Both cannot be specified. If the index is blank, the parts are deleted based on the Content-Id.

Figure 2-34 Build Composite Content Properties Window—Input Parameters, Attach



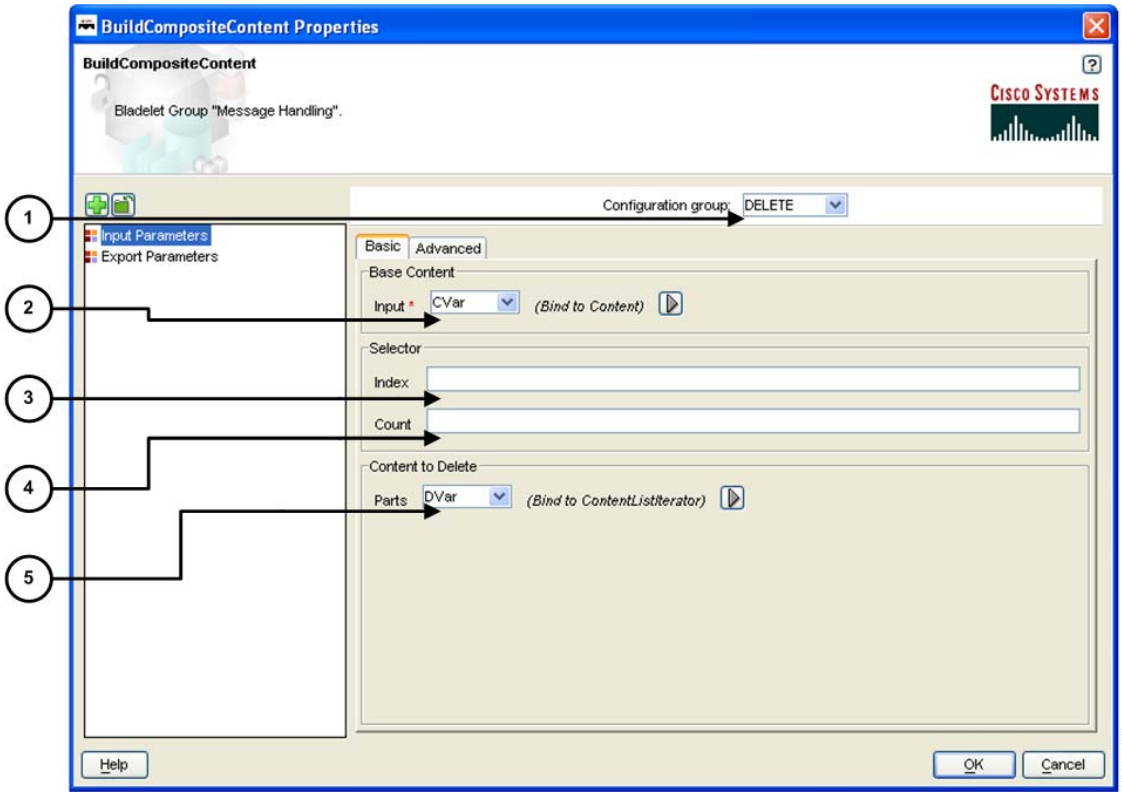
1	Configuration Group	Configuration group, set here to Attach.
2	Input	Base-content input message. Base content to which parts are attached and it has to be a multipart.
3	Index	Optional. Index to attach. If blank, attaches to the end.
4	Parts	List of contents to attach.

Figure 2-35 Build Composite Content Properties Window—Input Parameters, Overwrite



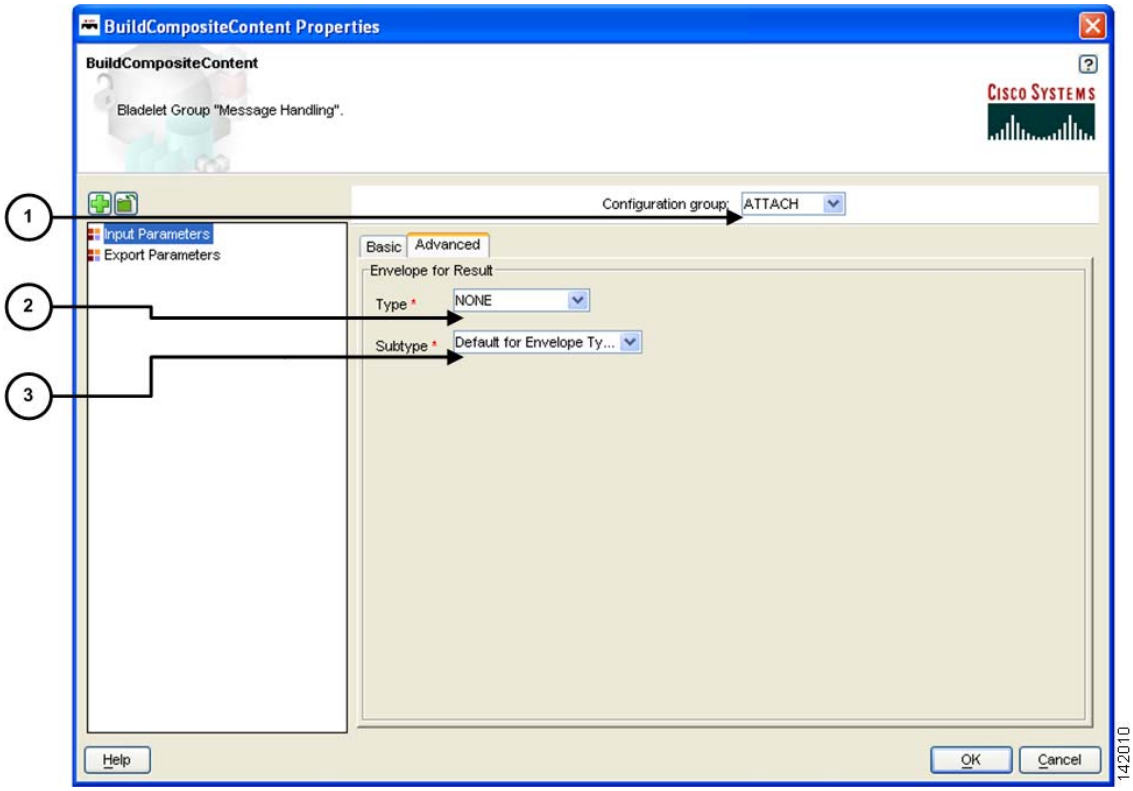
1	Configuration Group	Configuration group, set here to Overwrite.
2	Input	Base-content input message. Base content to which parts are overwritten. Must be a multipart content.
3	Index	Optional. Index to overwrite.
4	Parts	List of contents to Overwrite. Use to overwrite existing contents at the index specified. If blank, it overwrites the input content based on the Content-Id of the parts.

Figure 2-36 Build Composite Content Properties Window—Input Parameters, Delete



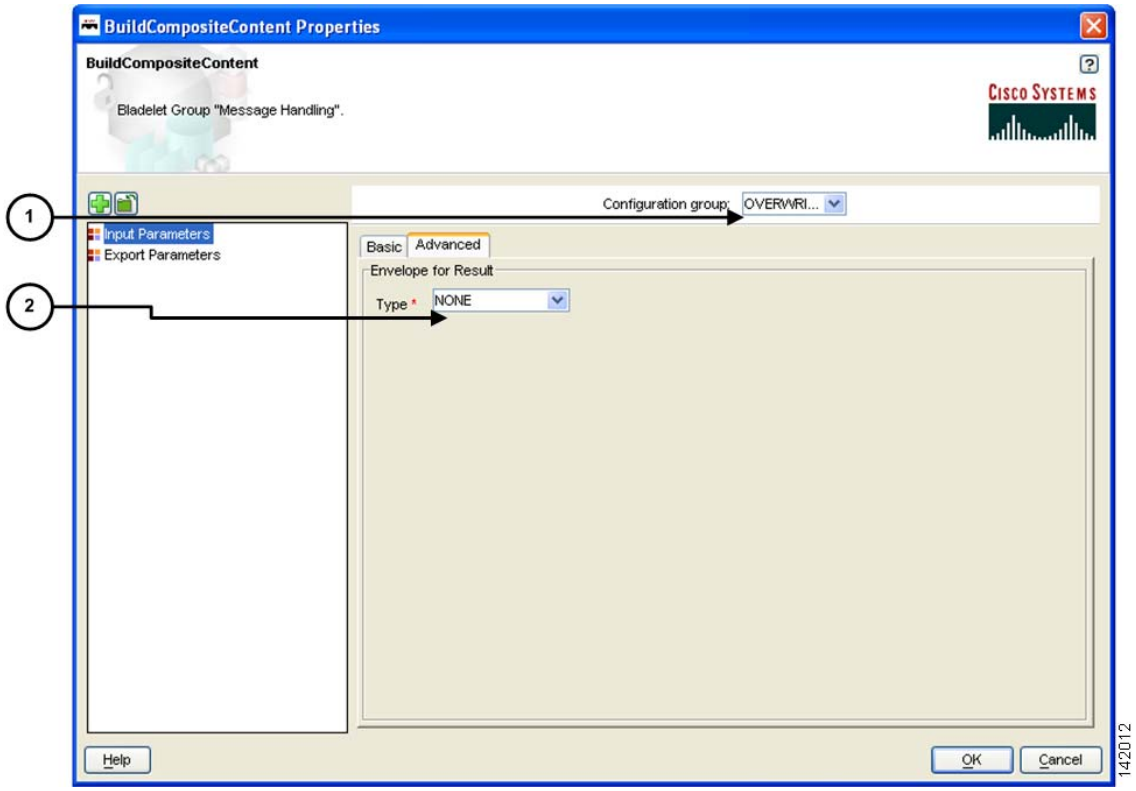
1	Configuration Group	Configuration group, set here to Delete.
2	Input	Base-content input message. Base content to which parts are overwritten. Must be a multipart content.
3	Index	Optional. Index to overwrite.
4	Count	Number of parts that need to be deleted from the index specified.
5	Parts	List of contents to delete. Select from the drop-down list or bind to a specific value. Use to delete existing contents from the Input Content. Mutually exclusive with Index.

Figure 2-37 Build Composite Content Properties Window—Input Parameters, Advanced 1



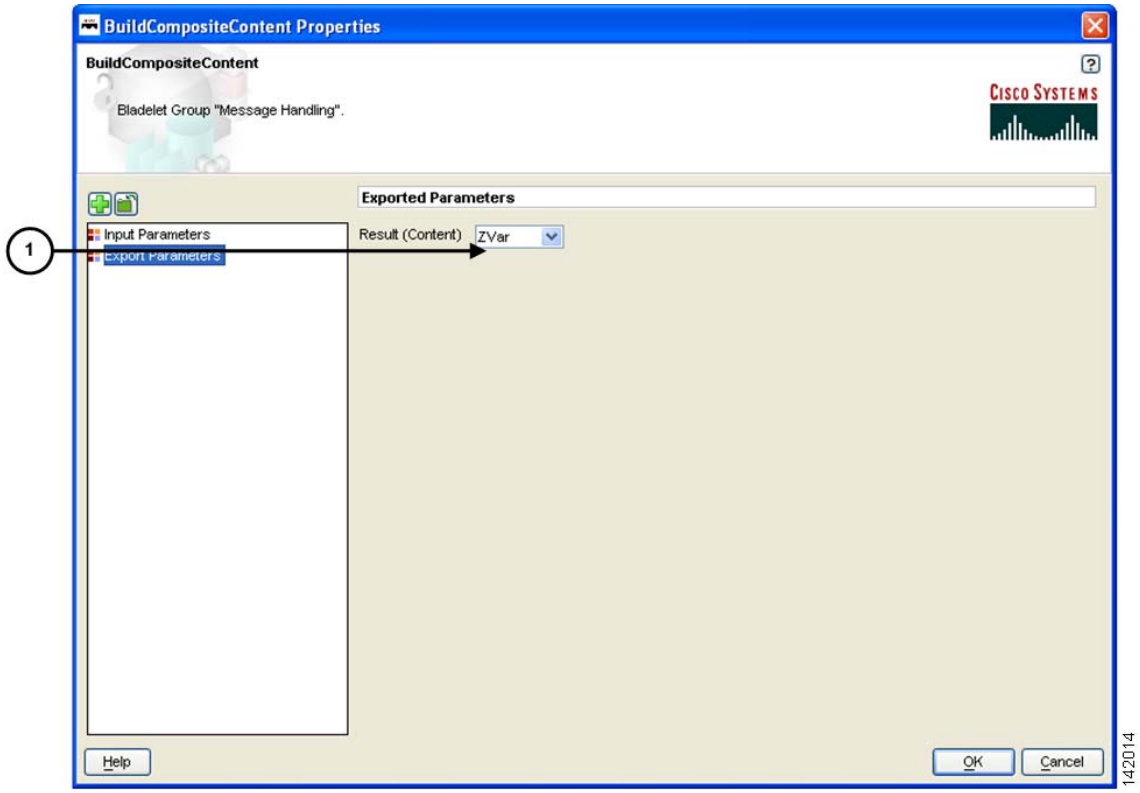
1	Configuration Group	Configuration group, set here to Attach.
2	Type	Output-message type. Default is None, which is the same as a regular MIME message. RosettaNet outputs the message in RosettaNet format.
3	Subtype	Header for subtypes when the input content is null. Can be set only when Configuration Group is set to Attach.

Figure 2-38 Build Composite Content Properties Window - Input Parameters, Advanced 2



1	Configuration Group	Configuration group, set here to Overwrite. Window looks the same if the value is set to Delete.
2	Type	Output-message type. Default is None, which is the same as a regular MIME message. If set to RosettaNet, outputs the message in RosettaNet format.

Figure 2-39 Build Composite Content Properties Window—Export Parameters



1	Result	Type of exported parameter such as ZVar.

Outcome

- On Success, the BuildCompositeContentBladelet exports a Content that is built from the inputs and other parameters specified.

Exceptions

- ParsingException: Exception thrown when input data is not MIME or when the data could not be parsed.

Discard

Summary

The Discard Bladelet discards a message based on whether it meets certain policies or message requirements established in the PEP and has no user-configurable input parameters.

Prerequisites and Dependencies

None.

Details

There are no properties windows for this Bladelet.

Outcome

- On success, PEP processing stops and connection to the client is lost. In case of Queue based messages (JMS/MQ), the adapter transfers the message to dead letter queue, if one is configured.

Exceptions

None.

Create Message

Summary

This Bladelet creates a message wihin a PEP. You can use the message body as an input parameter to this Bladelet or set as the response_message in the PEP context. You can use create message to shorten a message, request PEPs, or speed up responses.

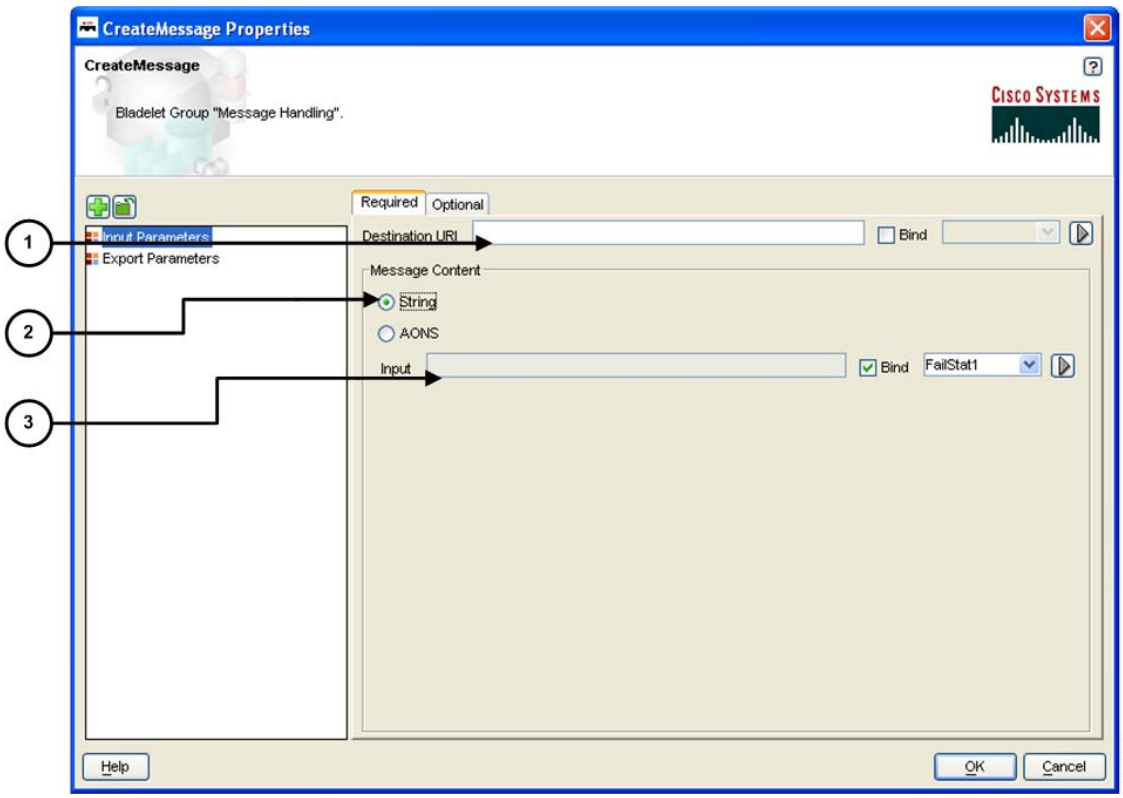
Prerequisites and Dependencies

None.

Details

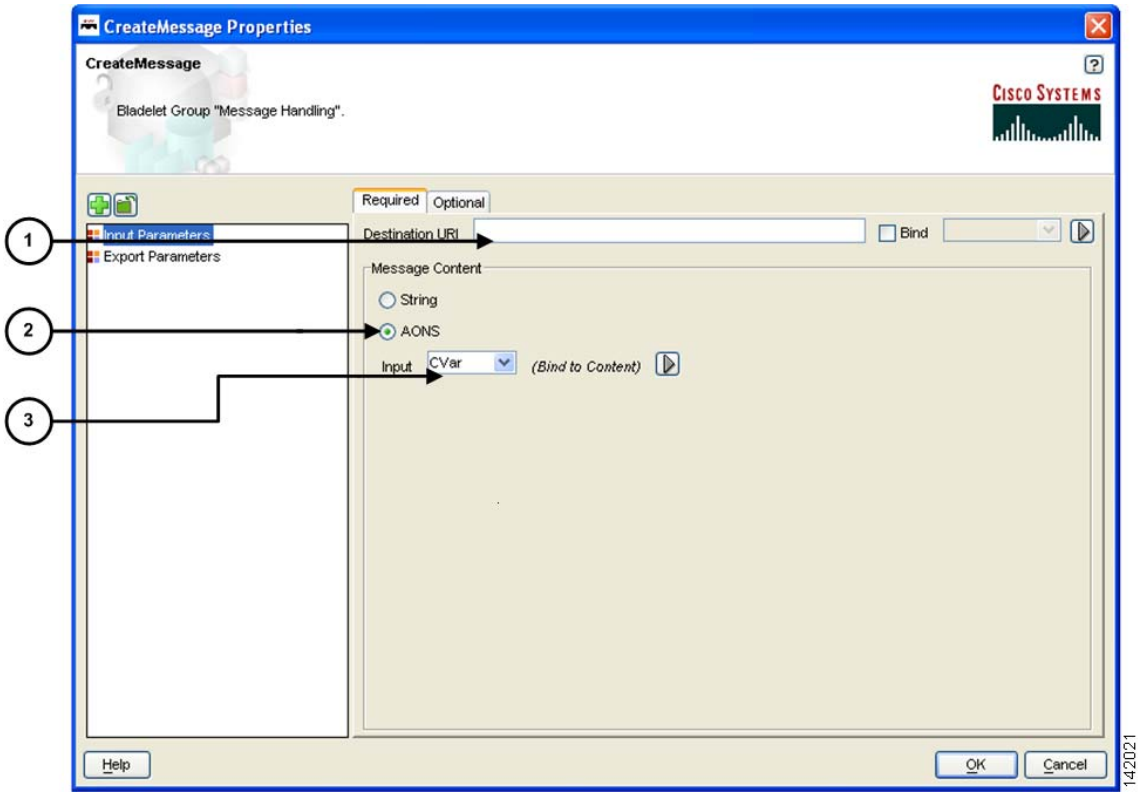
In the Create Message properties window under the Input Parameters section, tabs show required (Figure 2-40) and optional (Figure 2-42) settings.

Figure 2-40 Create Message Properties Window—Input Parameters, Required 1



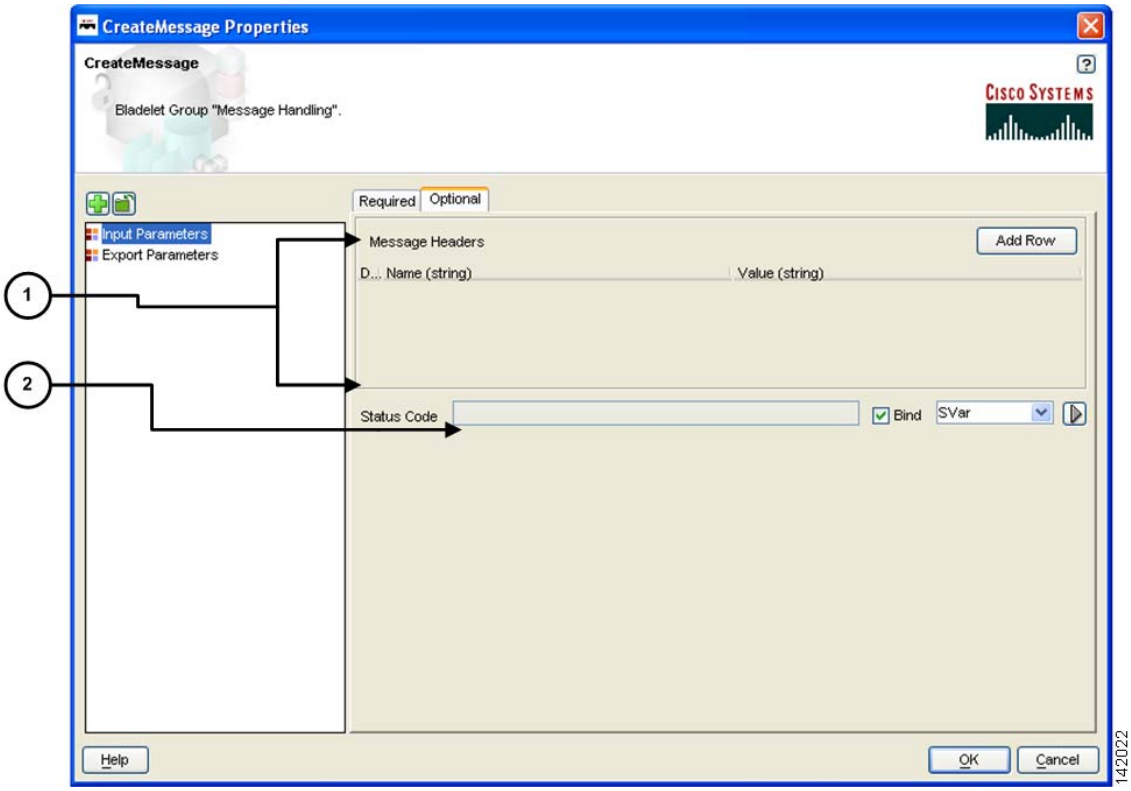
1	Destination URI	Destination of the new message. Need not be set in case of a response message or if the URI can specified in the Bladelets that work on this message (example: Send).
2	String	Message content (string type). Required.
3	Input	Input content mentioned in 2 above.

Figure 2-41 Create Message Properties Window—Input Parameters, Required 2



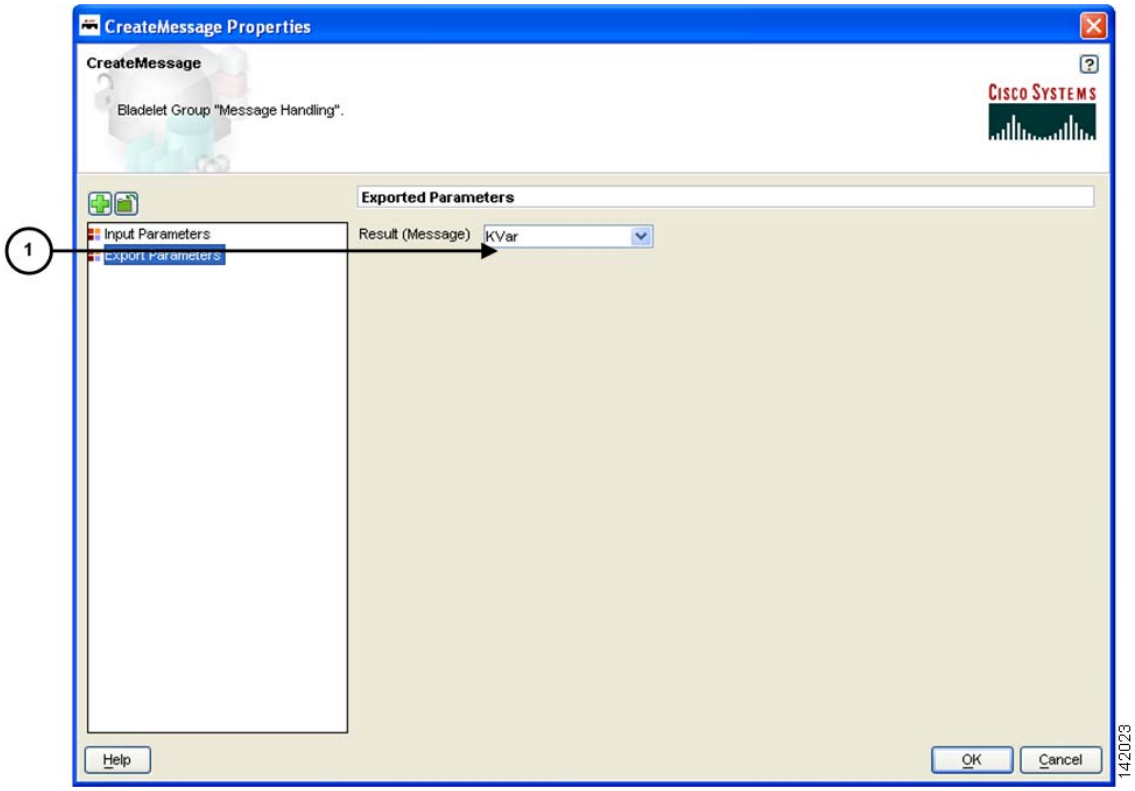
1	Destination URI	Destination URI for the new message. Need not be specified in case of a response message or if the URI can specified in the Bladelets that work on this message (for example, Send).
2	AONS	AON content is created using the bytes in the input. Set as the message payload. Required.
3	Input	CVar.

Figure 2-42 Create Message Properties Window—Input Parameters, Optional



1	Message Headers	Optional. One or more headers of the created message. Add rows as needed and enter a header name and value (string types).
2	Status Code	Optional. Status code of the created message. Useful if you have to create an error response message with a certain status code such as 500.

Figure 2-43 Create Message Properties Window—Export Parameters



1	Result	Resulting created message.

Outcome

- On success, a new AON message is produced that can be consumed via a variable and used in Bladelets such as Send, BalanceLoad, Distribute, SetDestination, and Branch.

Exceptions

None.

Update Message

Summary

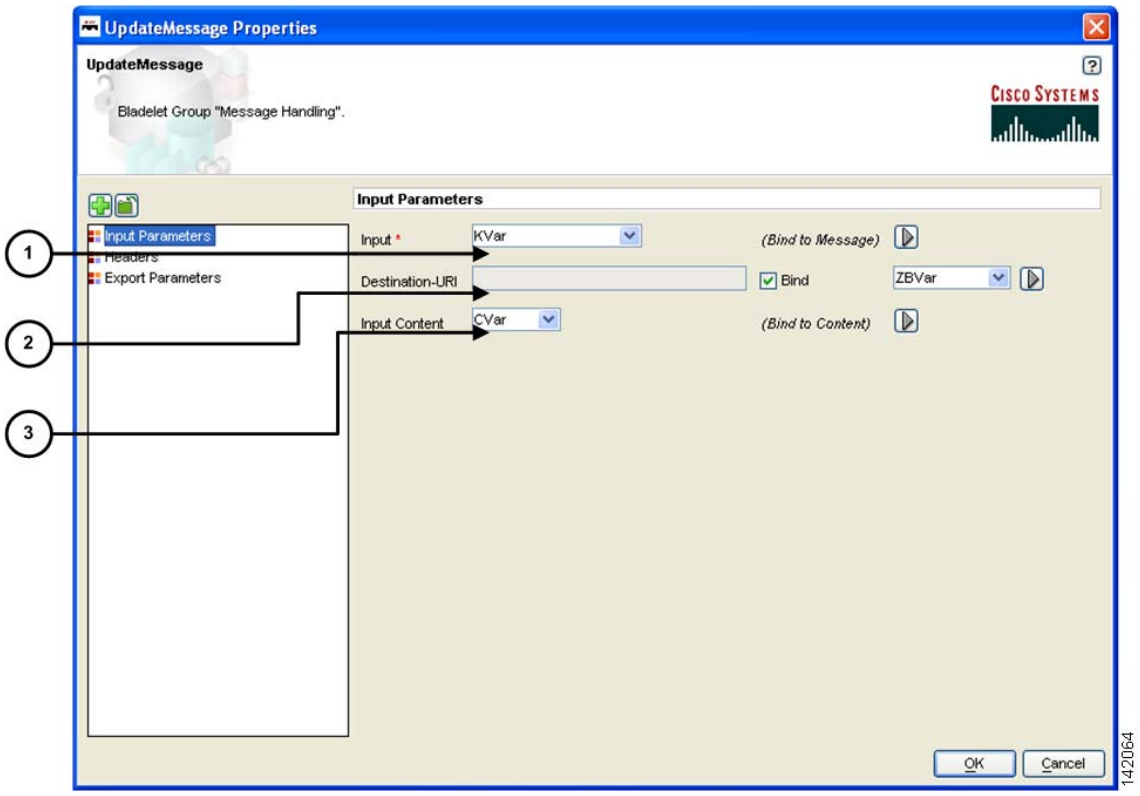
The UpdateMessage Bladelet updates an existing AON message in the PEP. User can optionally update the destination, content or the headers of the message. You can use this Bladelet to update the payload of the incoming message or modify some header information as it forwards on to an endpoint or to the client.

Prerequisites and Dependencies

None.

Details

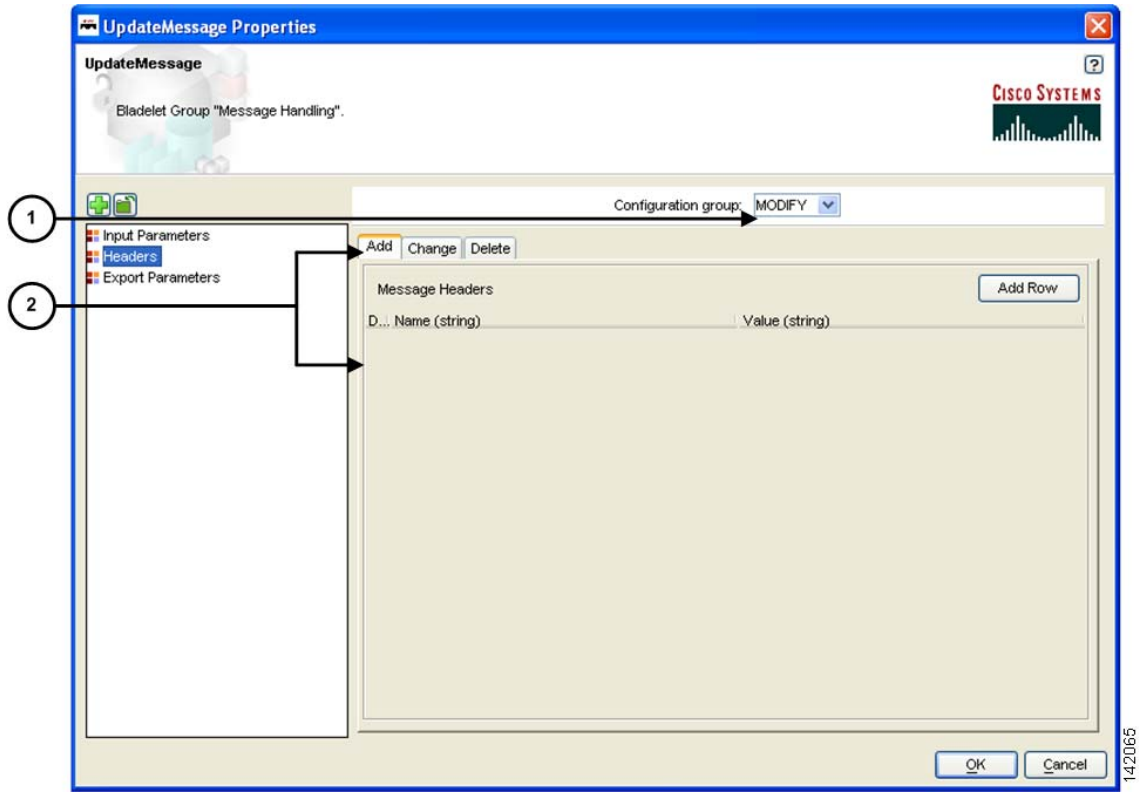
Figure 2-44 Update Message Properties—Input Parameters



1	Input	Message to be updated. Required.
2	Destination URI	URI to be set as the destination of the message being updated.
3	Input Content	Input content to be set as the content of the message being updated.

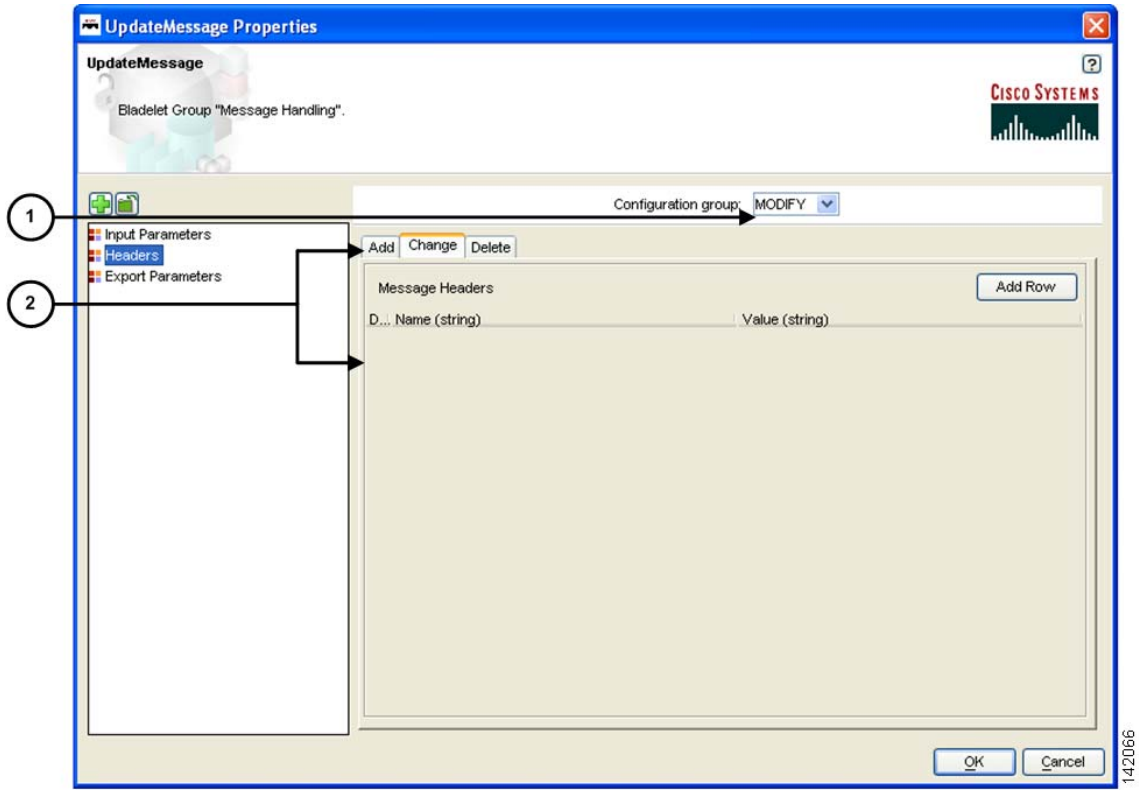
The Headers section has three tabs when the Configuration group is set to Modify (Figure 2-45 to Figure 2-47). You can set the Configuration group to Replace (Figure 2-48).

Figure 2-45 Update Message Properties Window—Headers, Modify, Add Tab



1	Configuration group	Configuration group, set here to Modify. Choices: Modify and Replace.
2	Add Message Headers	Header name-value pairs that are added to the existing set of headers of the message being updated.

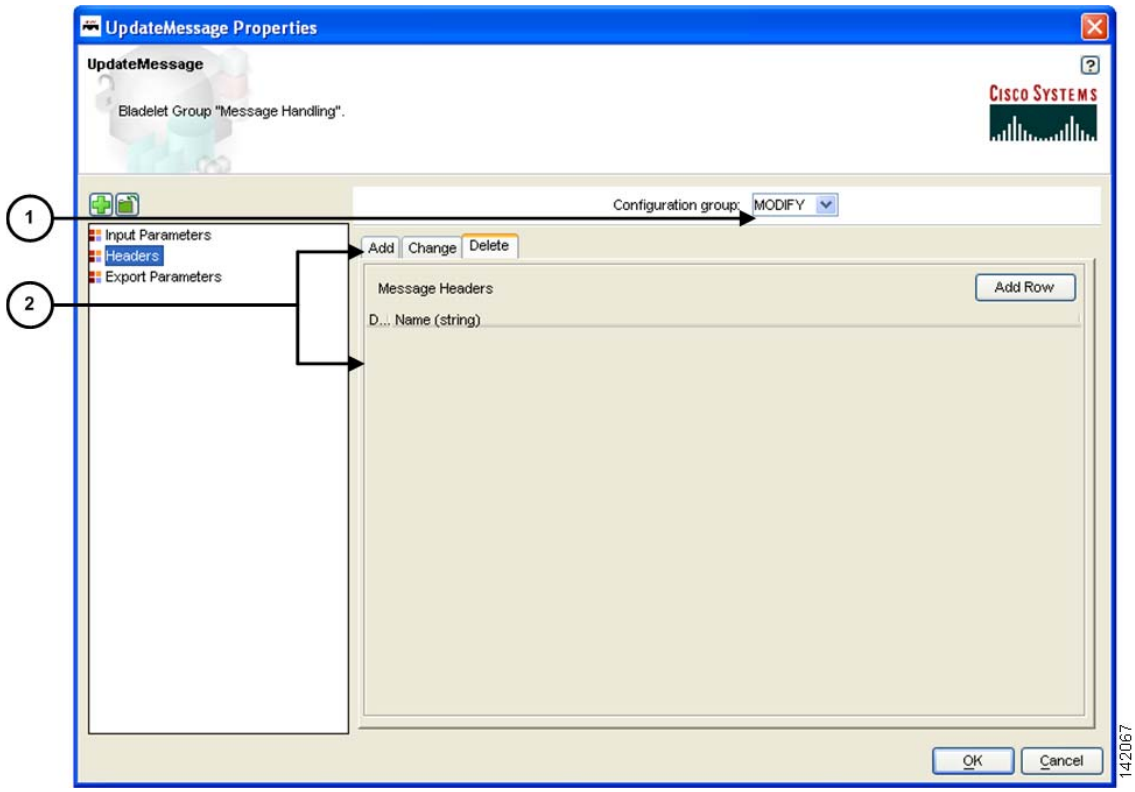
Figure 2-46 Update Message Properties Window—Headers, Modify, Change Tab



1	Modify	Configuration group, set here to to Modify.
2	Change Message Headers	Header-name values (string type), arranged in a two-column table. Use to update the headers of the message being updated. If the header does not exist, the new value is added. If it exists, the value is changed. Add a row for each instance.

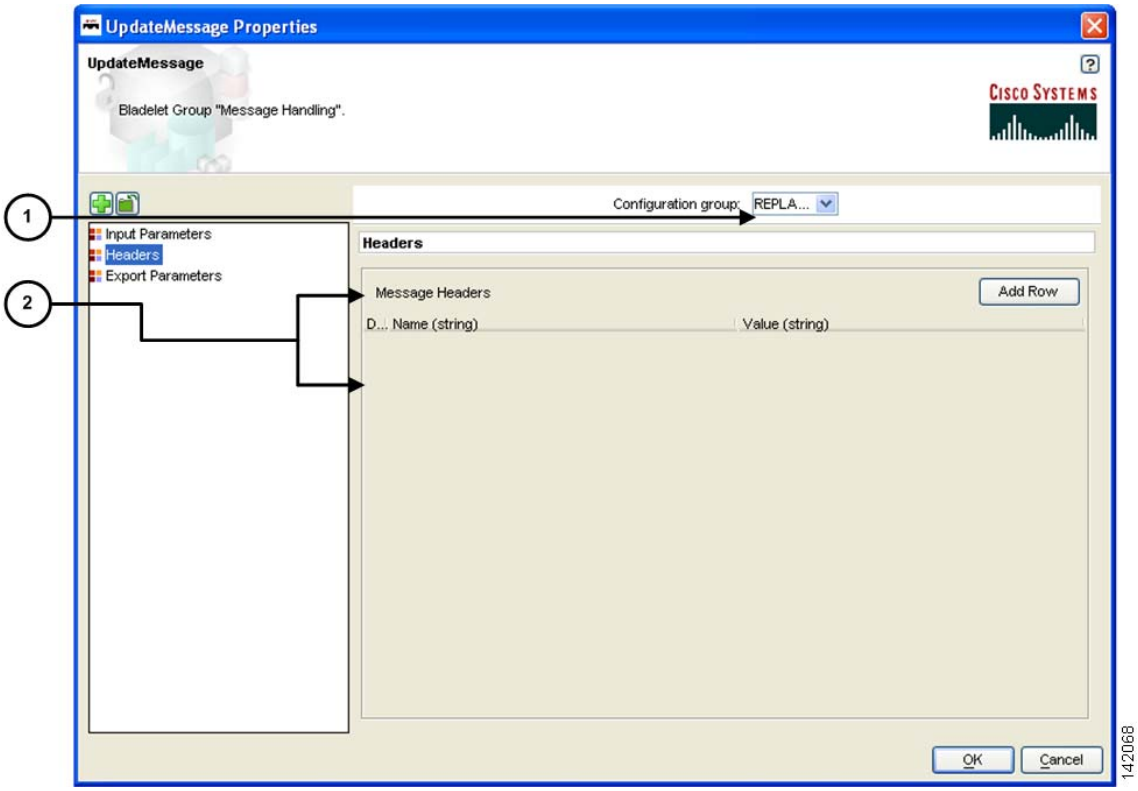
While the Configuration group is still set to Modify, delete one or more header names as necessary.

Figure 2-47 Update Message Properties Window—Headers, Modify, Delete Tab



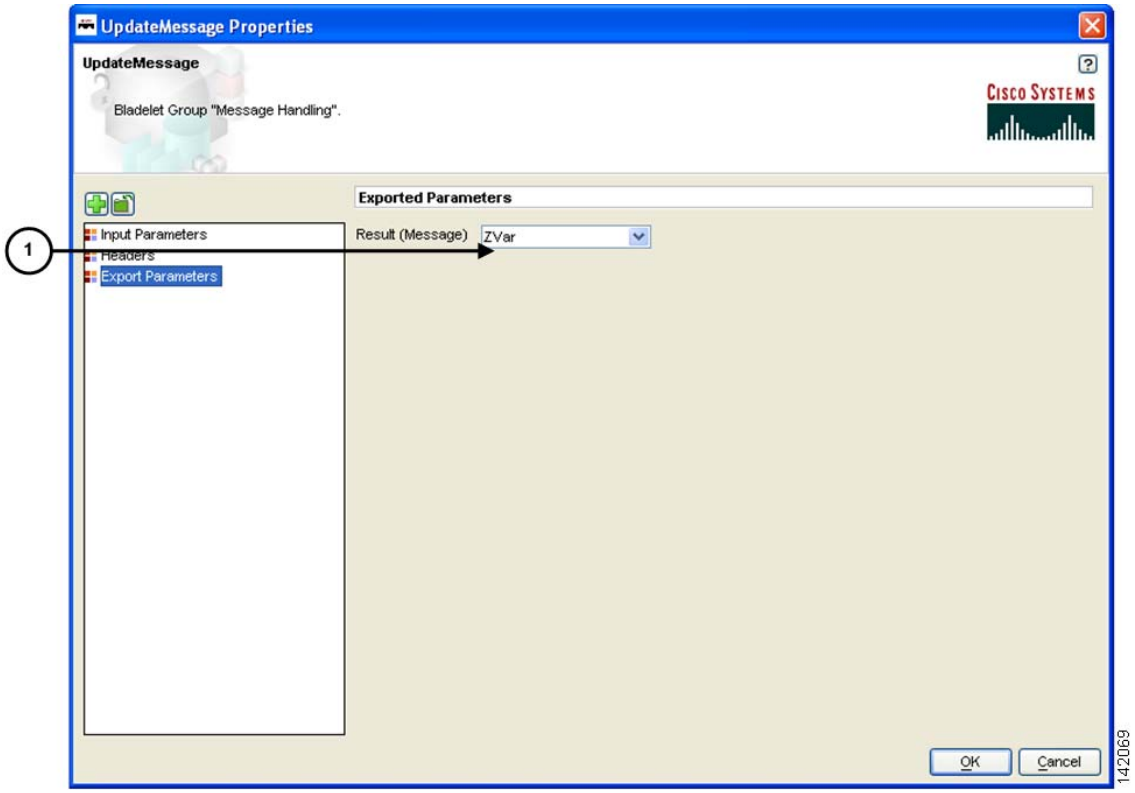
1	Modify	Configuration group, set here to Modify.
2	Delete Header Names	Names of the headers to be deleted from the message being updated (string type). Delete as many header names as required by adding rows.

Figure 2-48 Update Message Properties Window—Headers, Replace



1	Replace	Configuration group, set here to Replace.
2	Message Headers	Same as the message that was input. No new message is created by this Bladelet. Changes only the headers/content of the input message.

Figure 2-49 Update Message Properties Window—Exported Parameters



1	Result	Same as the message that was input. No new message is created by this Bladelet. Changes only the headers/content of the input message.

Outcome

- On success, the input message is modified as specified by the parameters.

Exceptions

None.

Create Content

Summary

The CreateContent Bladelet creates AON content from a string or converts one type of AON content to the other. Bladelets such as CreateMessage, UpdateMessage, AccessHTTP and BuildMIME operate on AON content that is produced by this Bladelet.

The content headers in the optional configuration group are applicable only to content that is used as a MIME part. If you want to add headers to the message, CreateMessage or UpdateMessage should be used, based on the requirement.

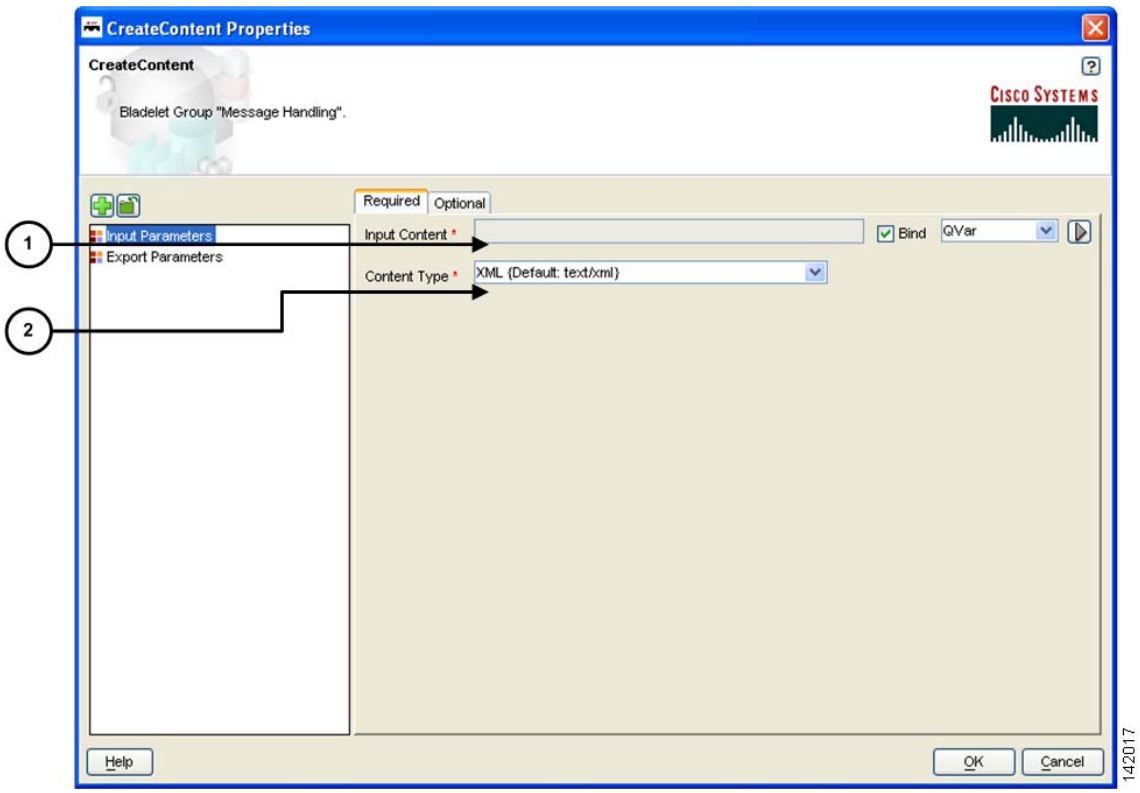
If the content type needs to be more specific than what is shown in the Required/Content-Type drop-down list, you can specify it as one of the headers. For example: Content-Type (header name) and “application/xml” (header value). The entry in the header overrides the default content type.

Prerequisites and Dependencies

None.

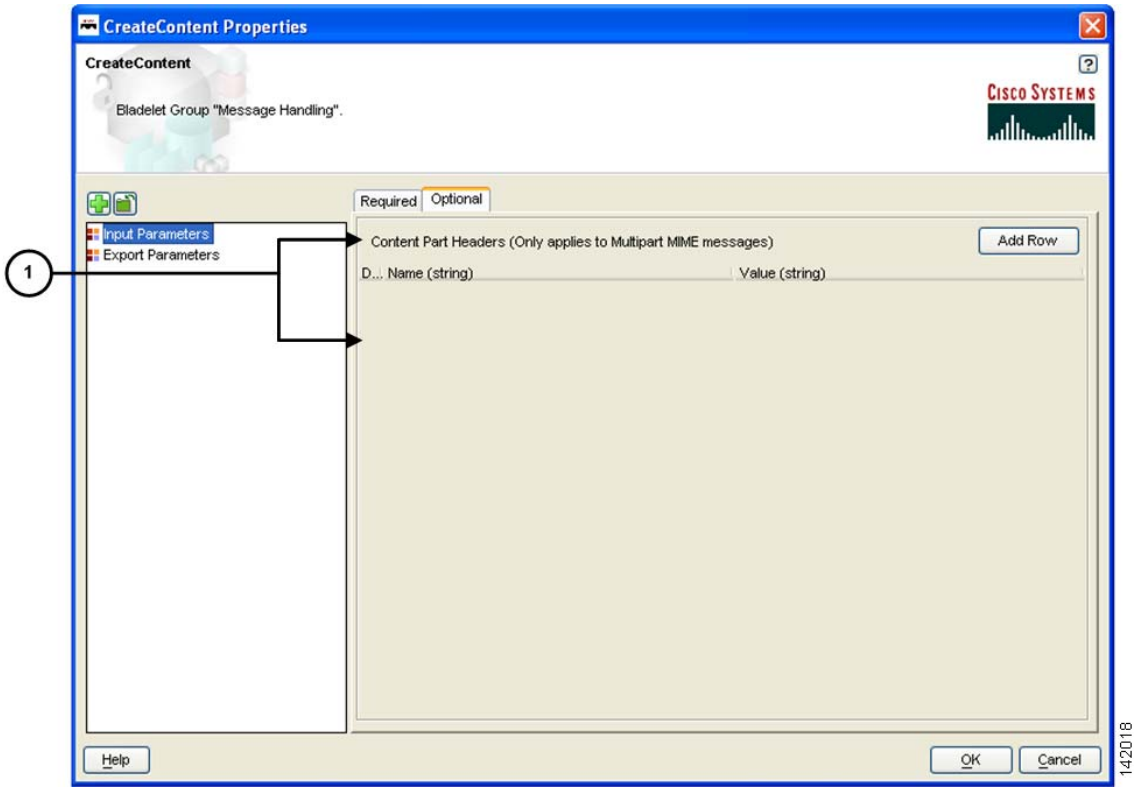
Details

Figure 2-50 Create Content Properties Window—General, Required Parameters



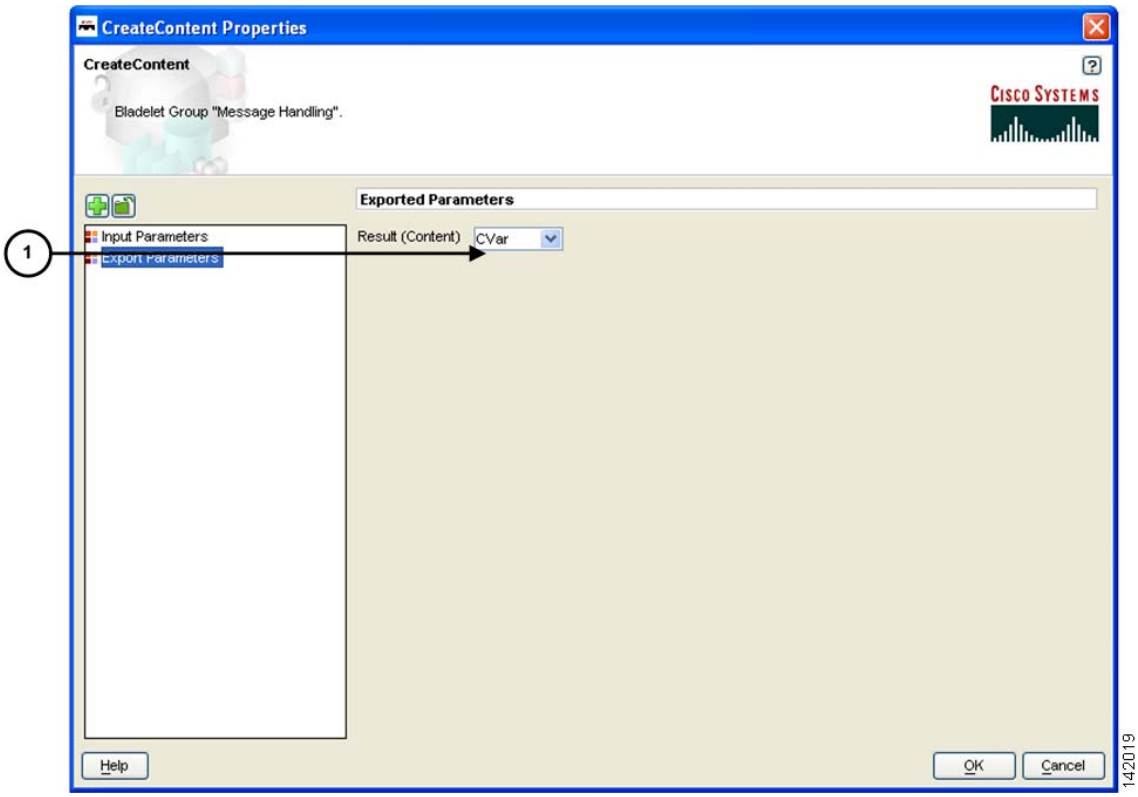
1	Input Content	Input content: • String—AON content is created from the bytes in the string. • AON content—AON content is created from the bytes in the input content. Useful to convert one type of content (say Stream) to another (say SOAP). The input content should be convertible to the desired output type.
2	Content Type	Content type. Choices: Stream Content, XML Content, SOAP Content, and Map Content.

Figure 2-51 Create Content Properties Window—General, Optional Parameters



1	Content Part Headers	Optional. Headers (name-value pairs) that apply to MIME parts only.

Figure 2-52 Create Content Properties Window—Export Parameters



1	Result	Created AON content.

Outcome

- On success, AON content is produced that can be consumed via a variable.

Exceptions

None.

Extract Composite Content

Summary

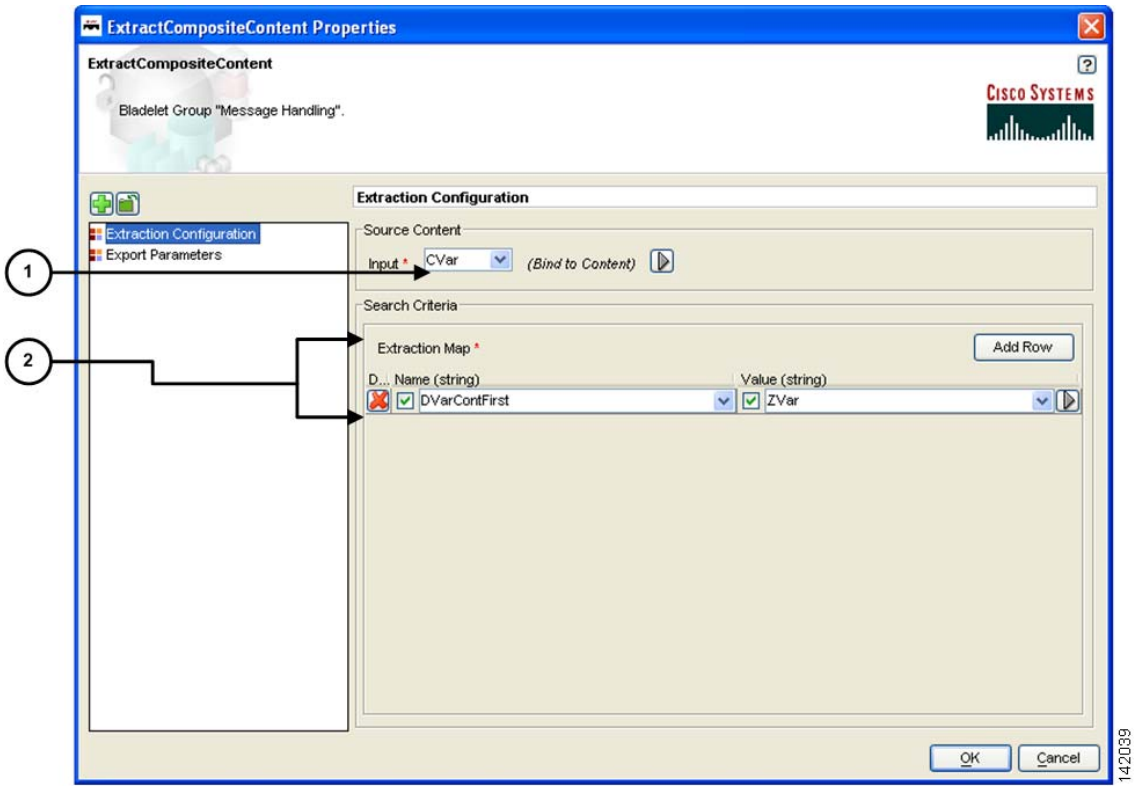
The Extract Composite Content Bladelet extracts the contents from a multipart content message.

Prerequisites and Dependencies

- Ensure that InputContent is available from the request message or create it from another Bladelet.

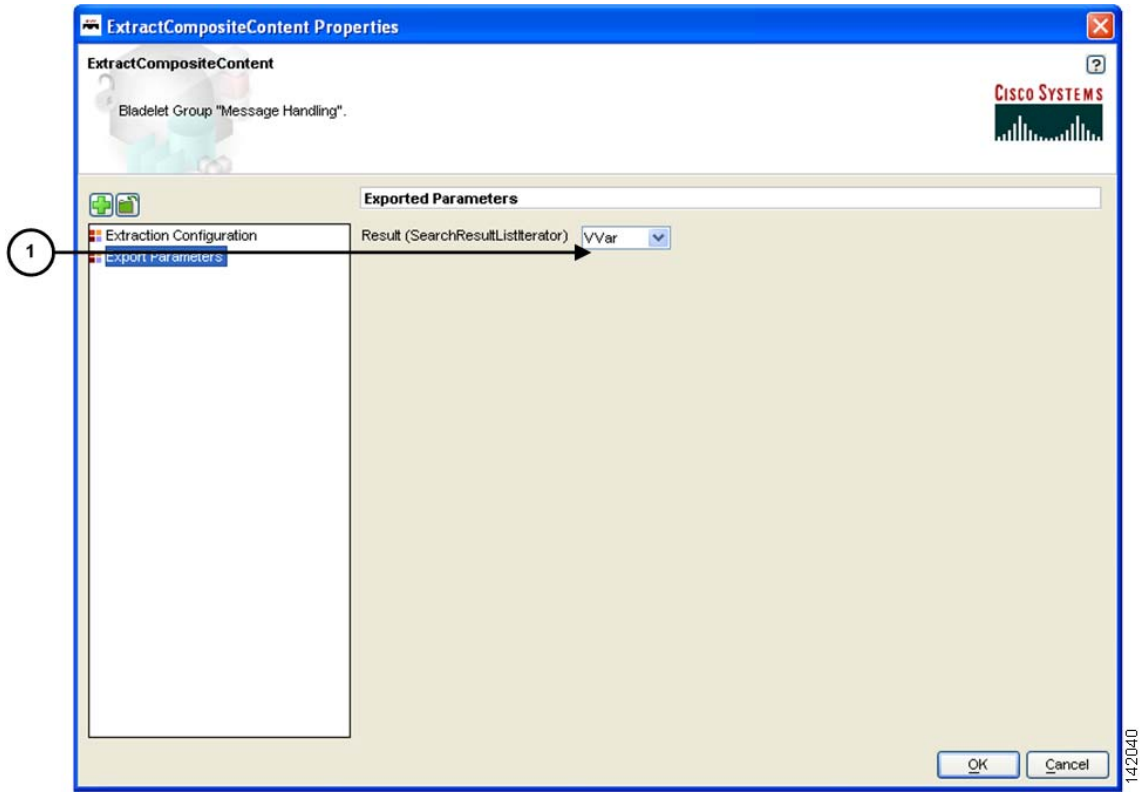
Details

Figure 2-53 Extract Composite Content Properties Window—Extraction Configuration



1	Source Content	Input content.
2	Extraction Map	Names and values for one or more extraction maps (string types).

Figure 2-54 Extract Composite Content Properties Window—Export Parameters



1	Result	Variable selected as exported parameter.

Outcome

- On success, the ExtractCompositeContentBladelet returns a SearchResultListIterator. Use this to extract specific contents as needed by other Bladelets.

Exceptions

- ParsingException: Exception thrown when input data is not MIME or when the data could not be parsed.

Create Response

Summary

The CreateResponse Bladelet tags an existing AON message in a PEP as the response message that has to be sent back to the client. Normally response messages are created in a PEP by way of a Send Bladelet or a Distribute Bladelet. The users can also handcraft response messages without involving an endpoint using CreateMessage Bladelet. RetrieveCache can put a response message into the PEP that was previously cached by the CacheData Bladelet. In the cases that do not involve Send and Distribute, CreateResponse has to be used to mark a particular message as the response message.

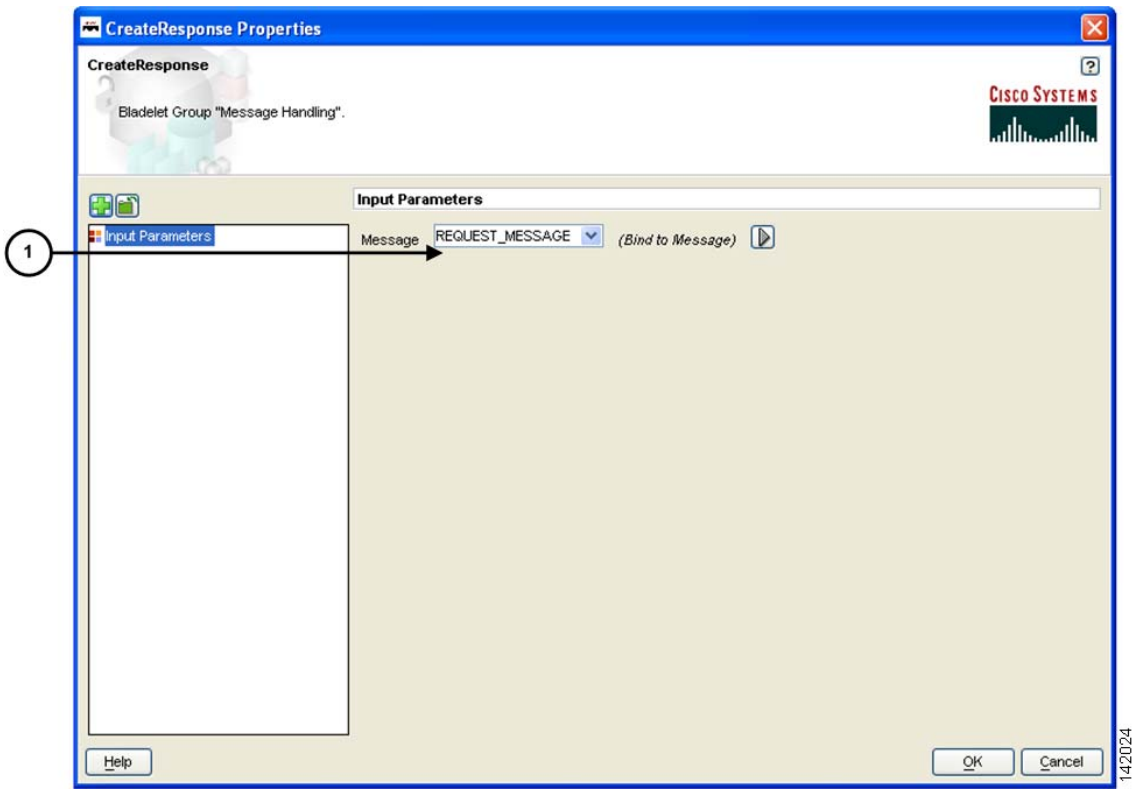
This Bladelet does not have to be used after Send or Distribute Bladelets. This Bladelet is typically used in conjunction with CreateMessage or RetrieveCache. It may also be used when the PEP has multiple Sends and based on some logic, one of the replies needs to be picked.

Prerequisites and Dependencies

None.

Details

Figure 2-55 Create Response Properties Window—Input Parameters



1	Message	Optional. Input message to be tagged as the Response message. If no input is specified, the Bladelet picks the current value of RESPONSE_MESSAGE variable. (Send/Distribute and RetrieveCache set their output to this variable).

Outcome

- On success, the specified input message is tagged as the response message of the PEP and is updated with necessary internal header information so that it can be sent back to the client at the end of PEP processing.

Exceptions

None.

Routing Category

In the Routing Category, there are four Bladelets:

Distribute

Summary

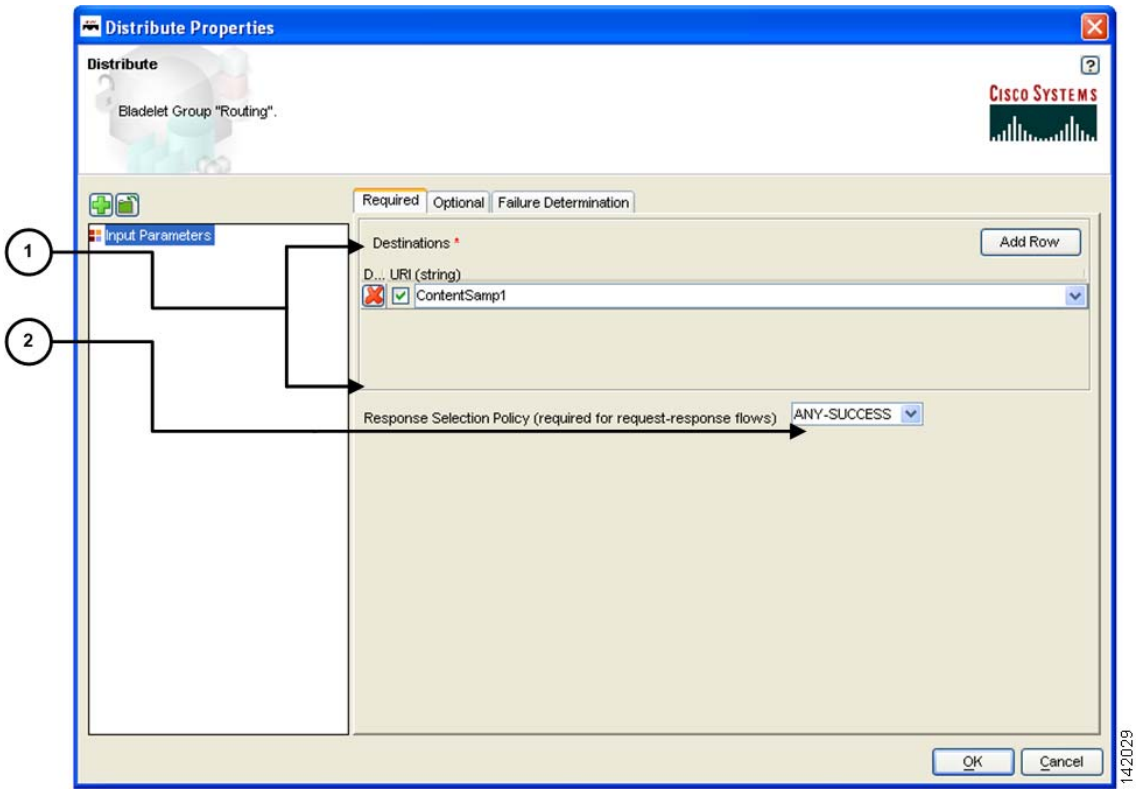
Distribute sends the same message over to multiple endpoints. In case of request-response PEPs, it gathers the responses, chooses one based on a selection criteria, and sets it as the response message. Distribute sets the response message of the PEP to the one chosen. It is a terminal Bladelet, so no Bladelet can follow this Bladelet.

Prerequisites and Dependencies

None.

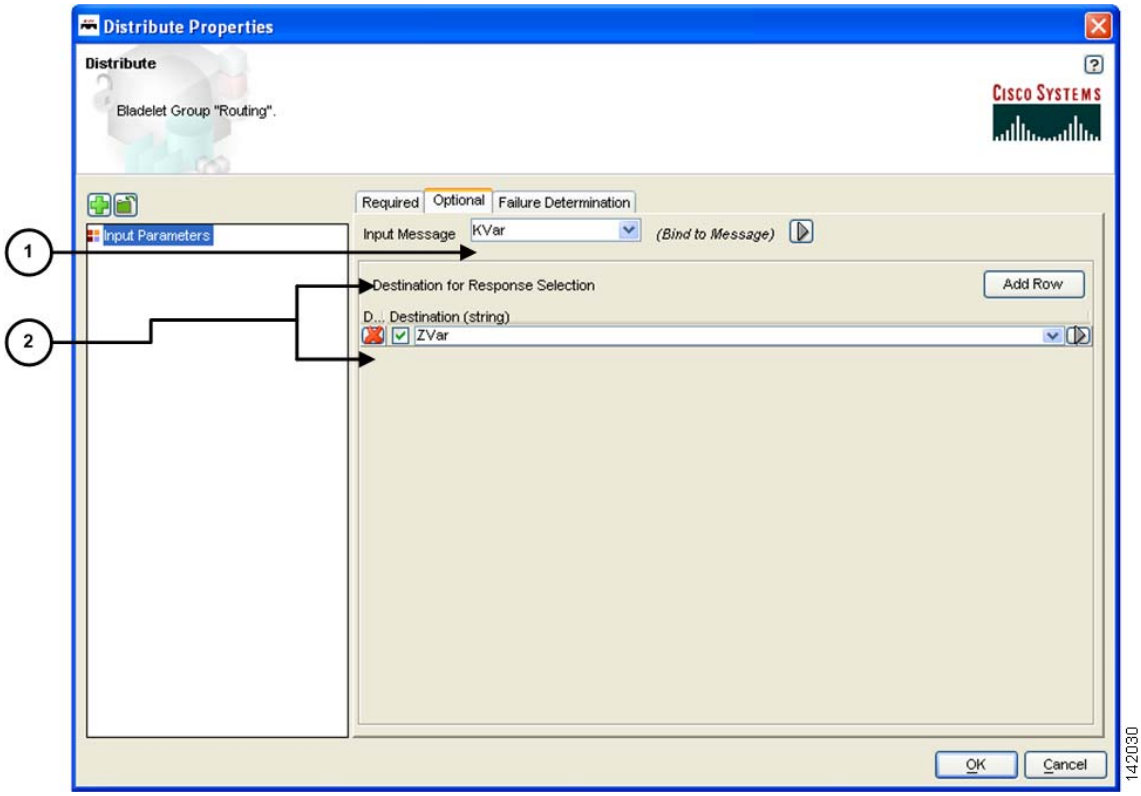
Details

Figure 2-56 Distribute Properties Window—Input Parameters, Required



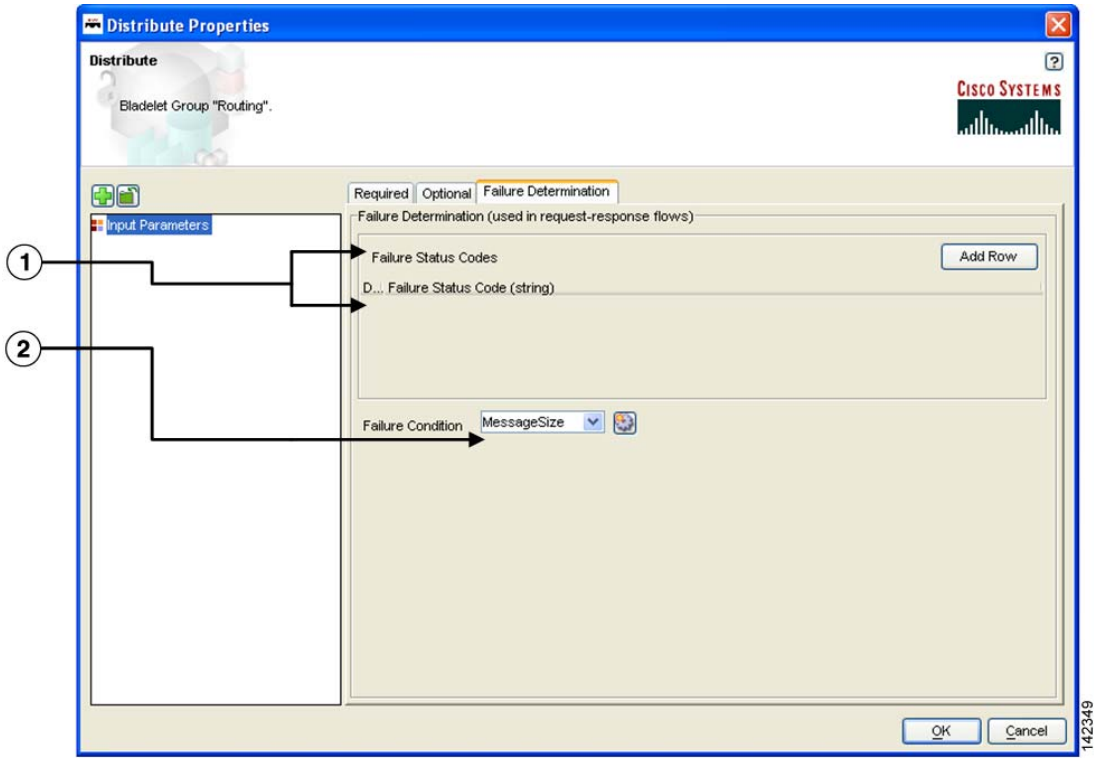
1	Destination	One or more destination URIs (string type) to which the message should be distributed. Required.
2	Response Selection Policy	Message to be considered as the response message. Required in request response PEPs. Not required for request-only PEPs. • Any-Success—First successful message to come back from the endpoints. • Any-Failure—First failed message to come back from the endpoints. • Any-Response—Any message to come back from the endpoints. Can be an error message or a proper response. If none of the messages matches the selected criteria, an error message is returned to the client.

Figure 2-57 Distribute Properties Window—Input Parameters, Optional



1	Input Message	Optional. The message that should be distributed. If not specified, the message used is based on the position of the Bladelet. If the Bladelet is placed before the response marker, then REQUEST_MESSAGE is used. If the Bladelet is placed after response marker, then RESPONSE_MESSAGE is used.
2	Destination for Response Selection	Optional. One or more destination URIs (string types) that form a subset of the list of destinations to which the message is distributed. Use only in case of request-response PEPs, to filter certain destinations whose responses are not of interest. If the URIs specified here are not in the list of destinations specified, validation errors result. If variables are involved, runtime exceptions can result if this is not a proper subset of the original set of destinations.

Figure 2-58 Distribute Properties Window—Input Parameters, Failure Determination



1	Failure Status Codes	Optional. Failure error codes (examples: 404, 500) (string types) that indicate a failure message. If none specified, any error code in the 400-600 range is considered to be a failure. Specifying particular error code helps if only some of these errors should be considered fatal. If the requirement is to treat a couple of error codes as non-fatal, instead of specifying the whole list, use Failure Condition (below) and specify a rule accordingly (use RESPONSE_MESSAGE.status() as the variable to compare against).
2	Failure Condition	If the condition evaluates to true, the response message is considered to be a failed message. Typically, the condition should be evaluated against a field/body of the RESPONSE_MESSAGE. Select a displayed choice or add a rule by clicking the Rules Wizard icon.

Outcome

- On success, all destinations receive the input message. In case of request-response PEPs, a message that matches the criteria is set as the response message of the PEP.

Exceptions

None.

Set Destination

Summary

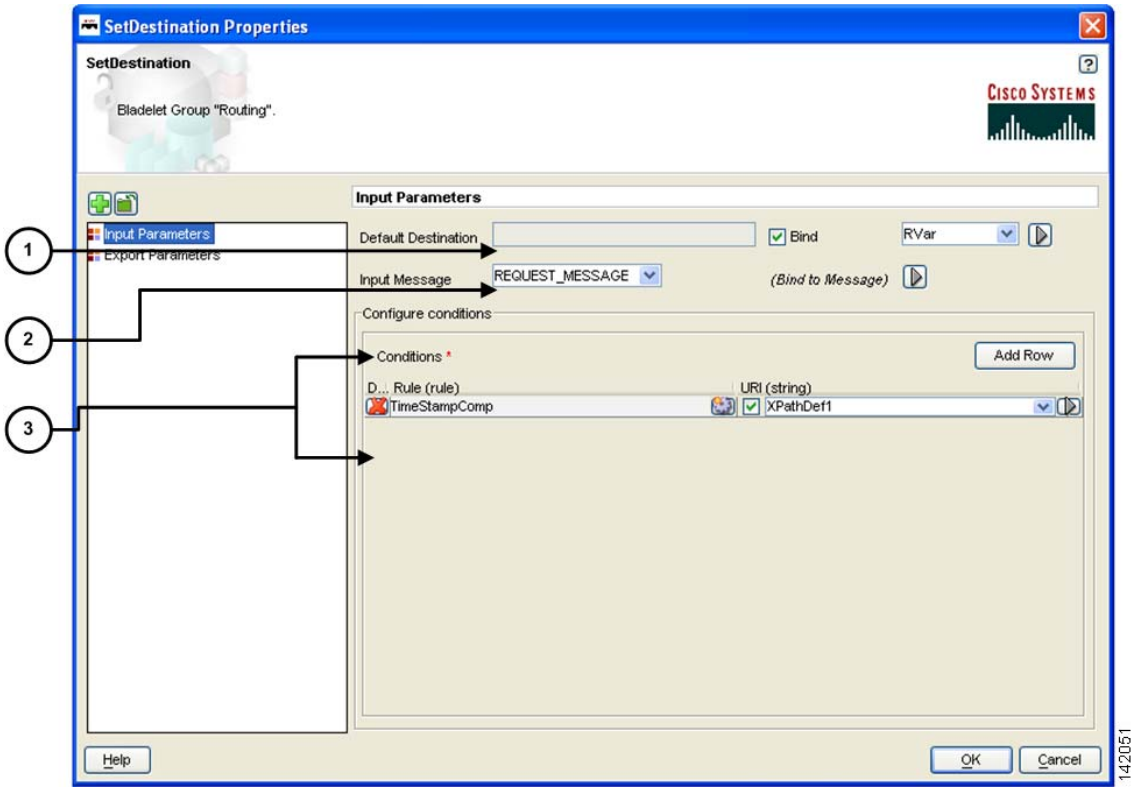
This Bladelet routes messages to destinations based on rules. It determines the final endpoint (URI) destination of the message being processed by the PEP.

Prerequisites and Dependencies

None.

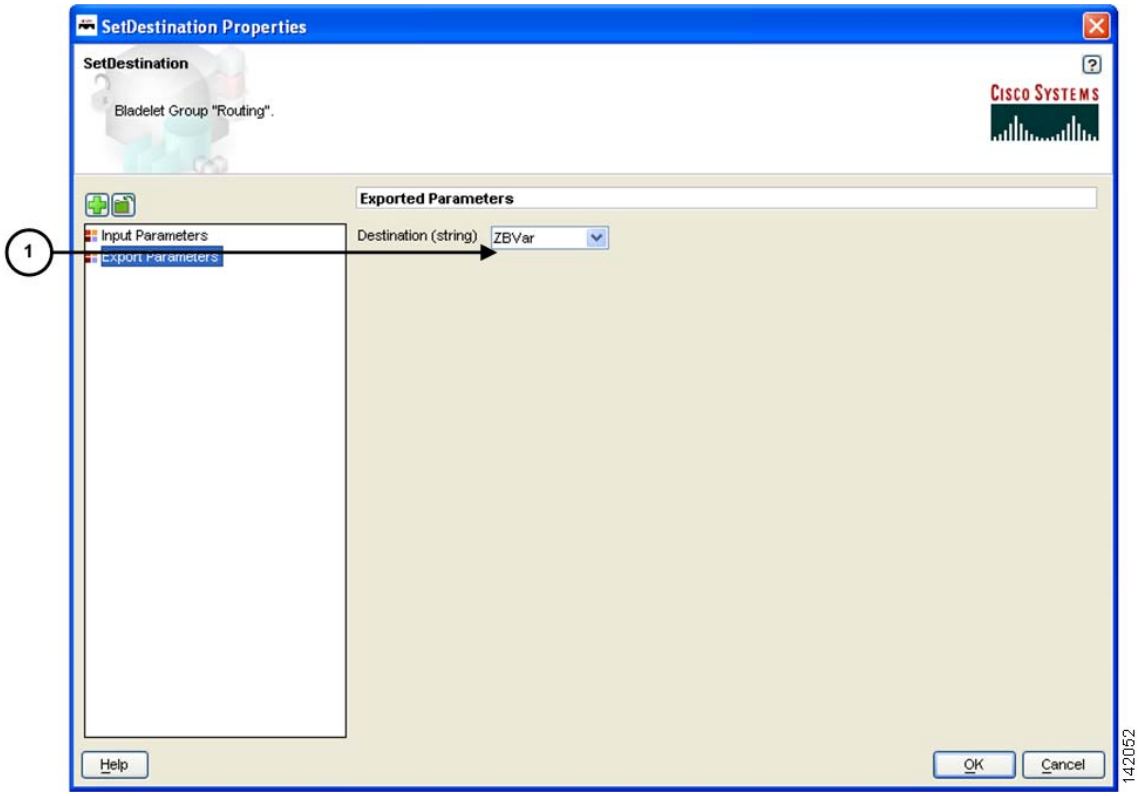
Details

Figure 2-59 Set Destination Properties Window—Input Parameters



1	Default Destination	Destination to be used if none of the rules evaluates to true.
2	Input Message	Message to be routed (whose destination should be updated). If not specified, the message used is based on the position of the Bladelet. If the Bladelet is placed before the response marker, then REQUEST_MESSAGE is used. If the Bladelet is placed after response marker, then RESPONSE_MESSAGE is used.
3	Conditions	Required. Rules and URIs (string type). Each rule is evaluated in the order it is specified and the evaluation stops at the first rule that evaluates to true. The URI corresponding to that particular rule is set as the destination URI of the message. If none of the rules evaluates to true, the URI specified as the default destination is used. Select a displayed choice or add a rule by clicking the Rules Wizard icon.

Figure 2-60 Set Destination Properties Window—Export Parameters



1	Destination	Destination that was set on the input message (string type).

Outcome

- On success, the destination of the input message is updated and set to the one corresponding to the rule that evaluates to true.
- If none of the rules evaluates to true, the URI given by Default Destination is set as the message destination. If a default destination is not specified, the original destination is not modified.

Exceptions

None.

Send

Summary

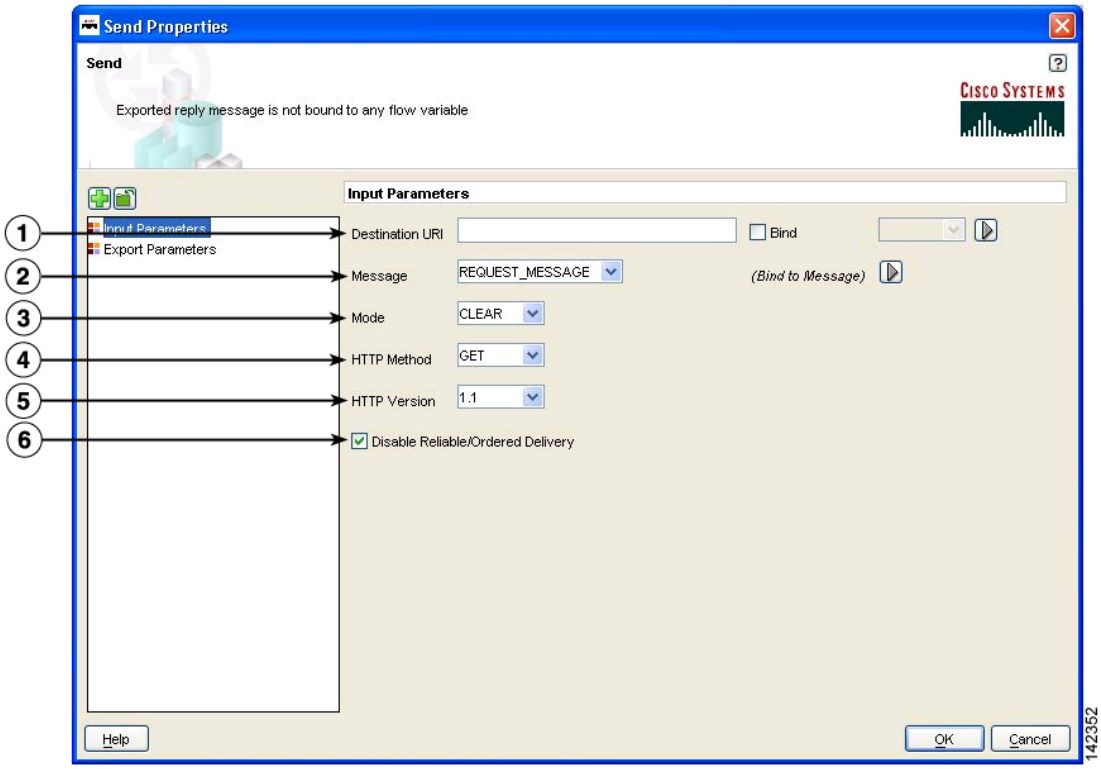
The Send Bladelet is the last item in a message request PEP and sends a message to a selected destination. The Bladelet performs protocol translation if the destination URI of the message to be sent out has to go out via an adapter that is different from the one that received the message.

Prerequisites and Dependencies

None.

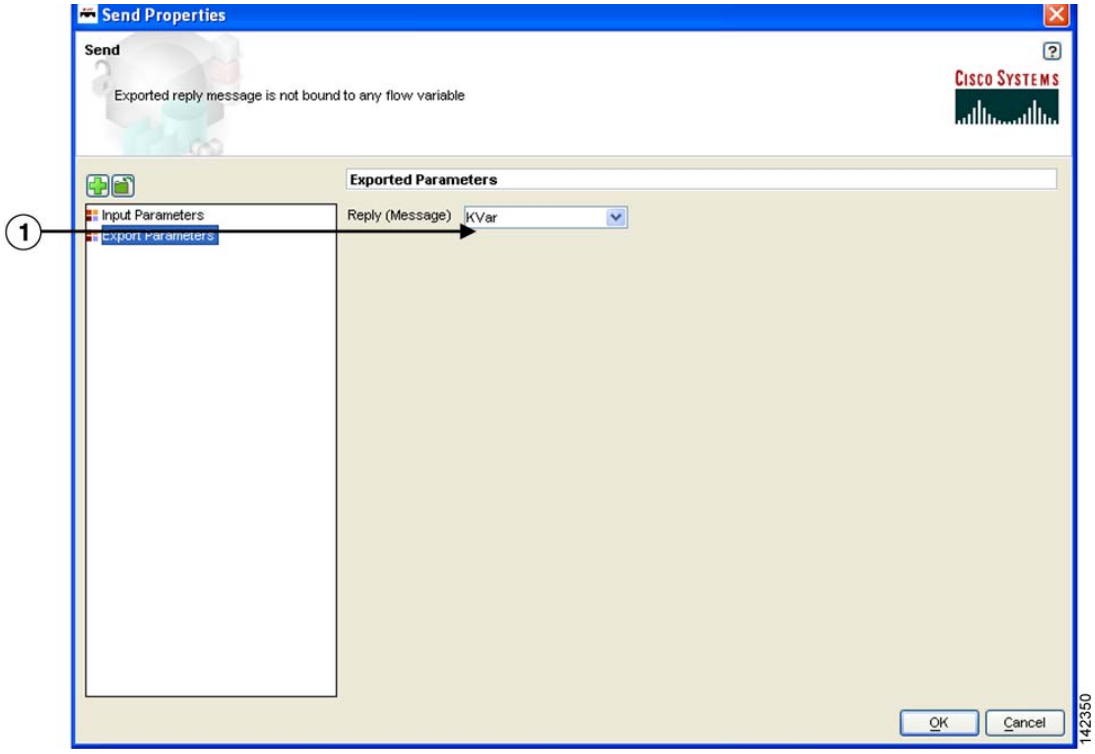
Details

Figure 2-61 Send Properties Window—Input Parameters



1	Destination URI	Destination URI to be set as the destination of the message being sent out. Overwrites the destination set in the input message. If this field is not set and the message does not have a valid URI, send fails and the client is sent an error message.
2	Message	Message to be sent. If none is specified, the message associate to the REQUEST_MESSAGE variable is used as the input.
3	Mode	Mode. Choices: Clear or SSL.
4	HTTP Method	Method. Choices: Get or Post. Used only if the message is sent out via HTTP.
5	HTTP Version	1.0 or 1.1. Used only if the message is sent out via HTTP.
6	Disable Reliable/Ordered Delivery	Disables the Reliable/Ordered semantics for this send. **Note** If Reliable/Ordered messaging is enabled, exactly one send in a flow of execution must have Reliable/Ordered semantics enabled. If more than one Send bladelet uses Reliable/Ordered semantics, AON will throw a runtime exception and the message will abort processing.

Figure 2-62 Send Properties Window—Export Parameters



1	Reply	Message from the endpoint.

Outcome

- On success, the response from the endpoint is output as the reply of this Bladelet that can be used via variables. If this is the terminal Bladelet in the PEP, the response returned by the endpoint is sent back to the client.

Exceptions

None.

Balance Load

Summary

This Bladelet distributes the message load to multiple devices for improved throughput service. BalanceLoad employs one of four different algorithms to decide which particular endpoint should receive the next message. It updates the destination URI of the message based on the algorithm. The Send Bladelet that follows the BalanceLoad Bladelet sends the message to the chosen destination.

BalanceLoad does not send the message out to the destination, but just updates the destination of the input message. A send Bladelet that follows BalanceLoad and is given the same input message uses the decision made by the BalanceLoad.

In case of a failure in send, BalanceLoad and Send work together to go through the remaining live destinations to try and find a valid destination to send the message through. Failover is not optional.

The following algorithms mentioned above are used as different approaches for load balancing:

- Round-robin (Figure 2-67 on page 2-78)
- Weighted round-robin (Figure 2-68 on page 2-79)
- Adaptive (Figure 2-69 on page 2-80)
- Highest Priority (Figure 2-70 on page 2-81)

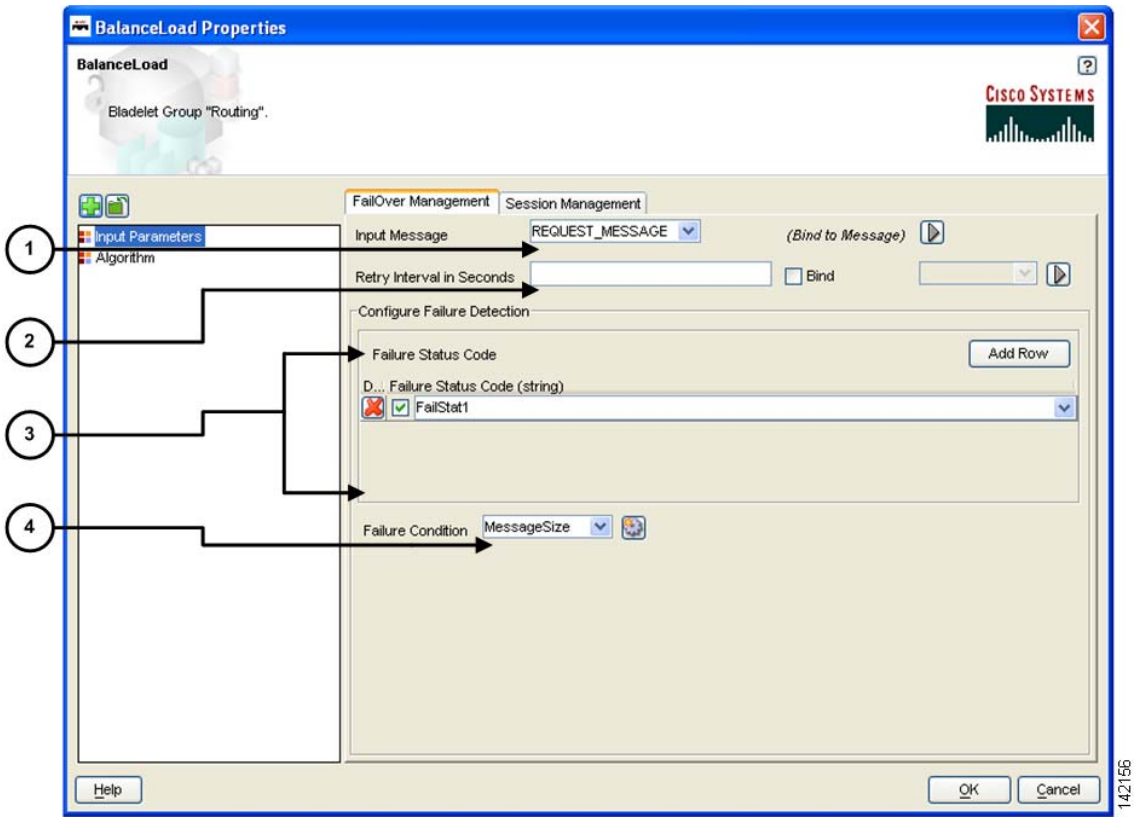
Prerequisites and Dependencies

None.

Details

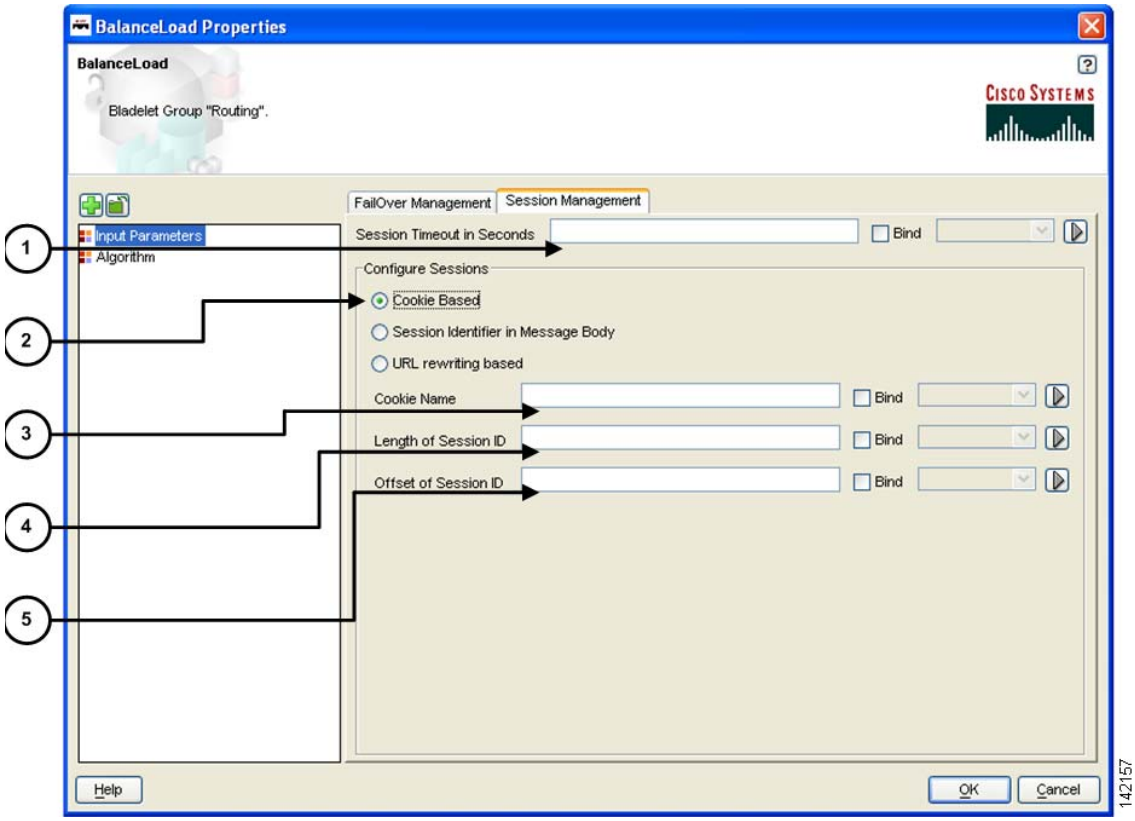
Two tabs, FailOver Management and Session Management, are under the Input Parameters section (Figure 2-63 to Figure 2-66).

Figure 2-63 Balance Load Properties Window—Configure Parameters, FailOver Management



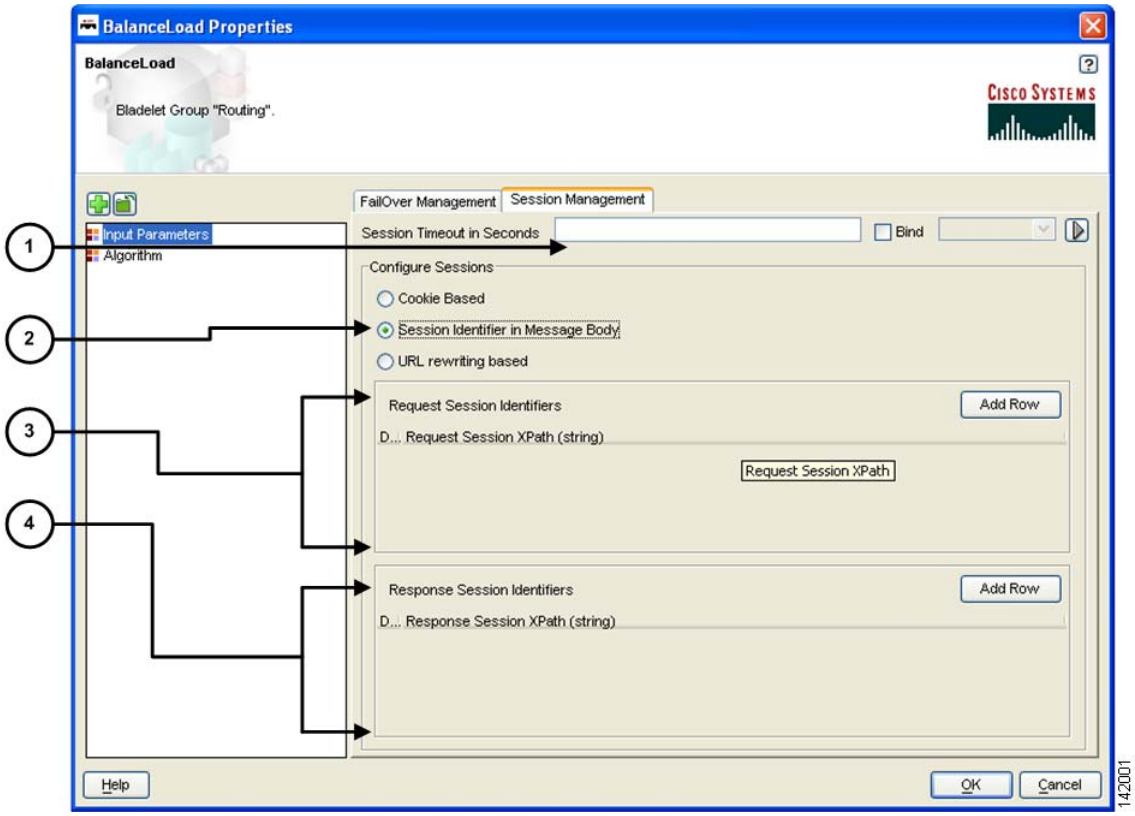
1	Input Message	Message type for the message whose destination is to be updated. If not specified, the message used is based on the position of the Bladelet. If the Bladelet is placed before the response marker, then REQUEST_MESSAGE is used. If the Bladelet is placed after response marker, then RESPONSE_MESSAGE is used.
2	Retry Interval in Seconds	Time for which a destination is not used again once it is considered to have experienced a failure.
3	Failure Status Code	Optional. One or more failure error codes (examples: 404, 500) that indicate a failed endpoint. If none specified, any error code in the 400-600 range is considered to be a failure. Specifying particular error codes helps if only some of these errors should be considered fatal. If the requirement is to treat a couple of error codes as non-fatal, instead of specifying the whole list, use Failure Condition (below) and specify a rule accordingly (use RESPONSE_MESSAGE.status() as the variable to compare against).
4	Failure Condition	Failure condition. If the condition evaluates to true, the destination is considered to have failed. Typically, the condition is evaluated against a field/body of the RESPONSE_MESSAGE. Select a displayed choice or add a rule by clicking the Rules Wizard icon.

Figure 2-64 Balance Load Properties Window—Input Parameters, Session Management 1



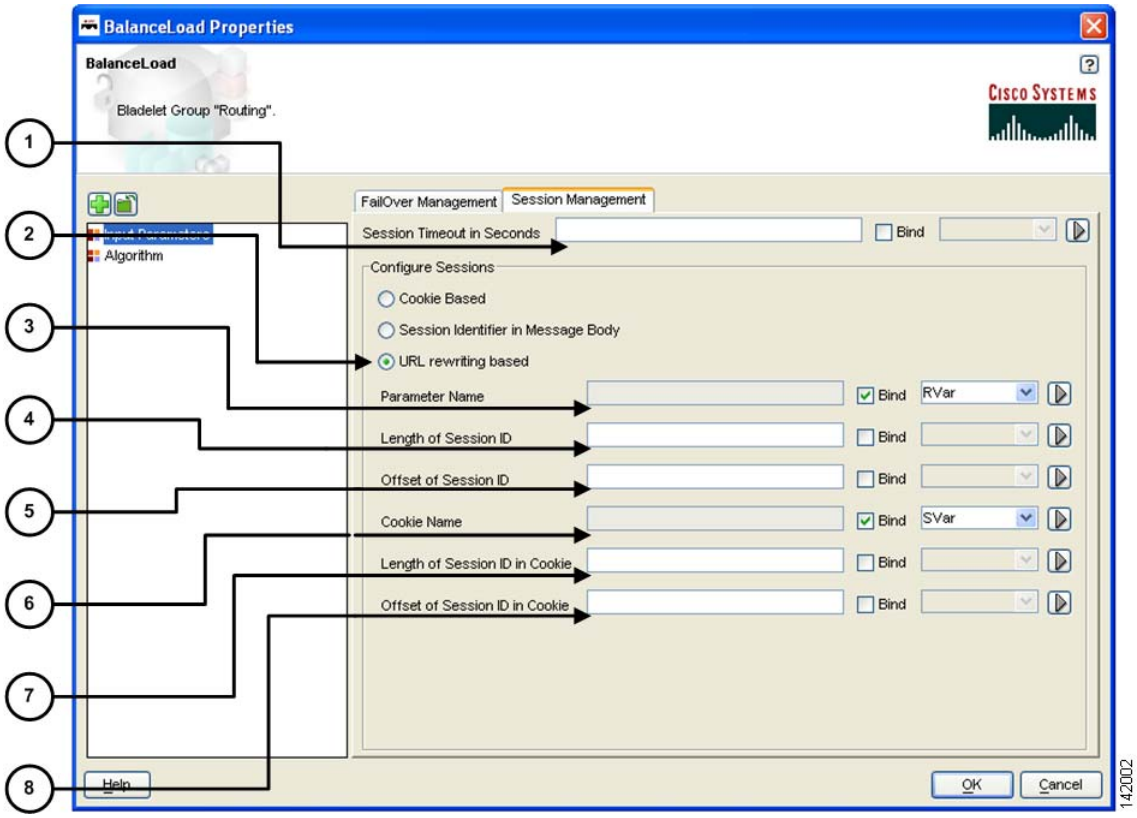
1	Session Timeout in Seconds	After a session is established, time for which it is considered active. Should equal the session timeout on the endpoint for predictable behavior.
2	Cookie Based	Select the session type. In the figure, Cookie Based is selected.
3	Cookie Name	Name of the cookie that carries the session information (example: in Unit3 this is JSESSIONID) in both request and response.
4	Length of Session ID	If the session ID is only a part of the cookie value (as opposed to being the whole cookie value), length of the session ID within the cookie value. Need not be specified if the session ID is the entire cookie value (example: Unit3).
5	Offset of Session ID	If the session ID is only a part of the cookie value (as opposed to being the whole cookie value), offset from where the session ID starts in the cookie value. Need not be specified if the session ID is the entire cookie value (example: Unit3).

Figure 2-65 Balance Load Properties Window—Input Parameters, Session Management 2



1	Session Timeout in Seconds	After a session is established, time for which it is considered active. Should equal the session timeout on the endpoint for predictable behavior.
2	Session Identifier in Message Body	Configuration session type. In the figure, Session Identifier in Message Body is chosen.
3	Request Session Identifiers	Request-session IDs. Each string is an XPath, which is evaluated against the Input message body and the resultant value is treated as the session ID. The first XPath evaluation that results in a non-null value is treated as the session ID.
4	Response Session Identifiers	Response-session IDs. Each string is an XPath, which is evaluated against the response message body and the resultant value is treated as the session ID. The first XPath evaluation that results in a non-null value is treated as the session ID.

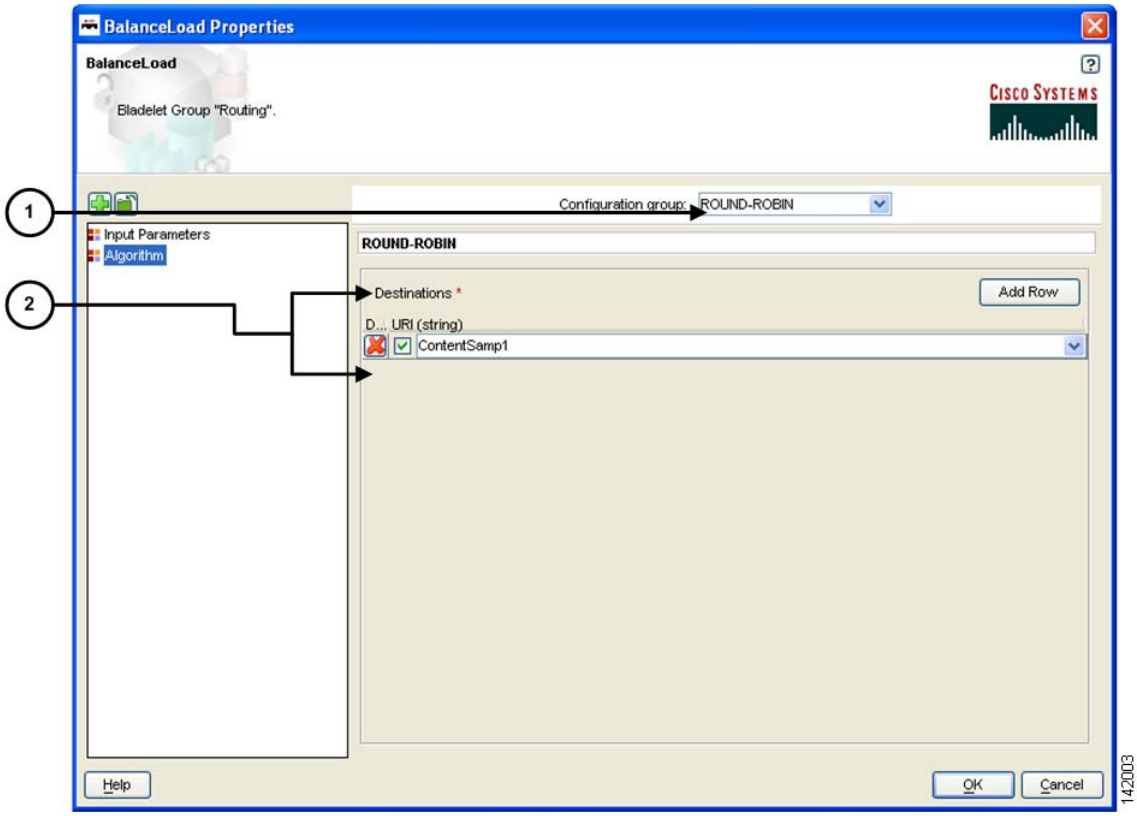
Figure 2-66 Balance Load Properties Window—Input Parameters, Session Management 3



1	Session Timeout in Seconds	After a session is established, time for which it is considered active. Should equal the session timeout on the endpoint for predictable behavior.
2	URL Rewriting Based	Configuration session type. In the figure, URL rewriting based is chosen.
3	Parameter Name	Name of the parameter in the rewritten URL that carries the session information (for example, in Unit3 this is `jsessionid=').
4	Length of Session ID	If the session ID is only a part of the parameter value in the URL (as opposed to being the whole cookie value), length of the session ID within the parameter value. Need not be specified if the session ID is the entire parameter value such as Unit3. It is very unlikely that a rewritten URL has a parameter in which the Session ID is only a part of the whole parameter.
5	Offset of Session ID	If the session ID is only a part of the parameter value in the URL (as opposed to being the whole cookie value), offset from where the session ID starts in the parameter value. Need not be specified if the session ID is the entire parameter value such as Unit3. It is very unlikely that a rewritten URL has a parameter in which the Session ID is only a part of the parameter value.

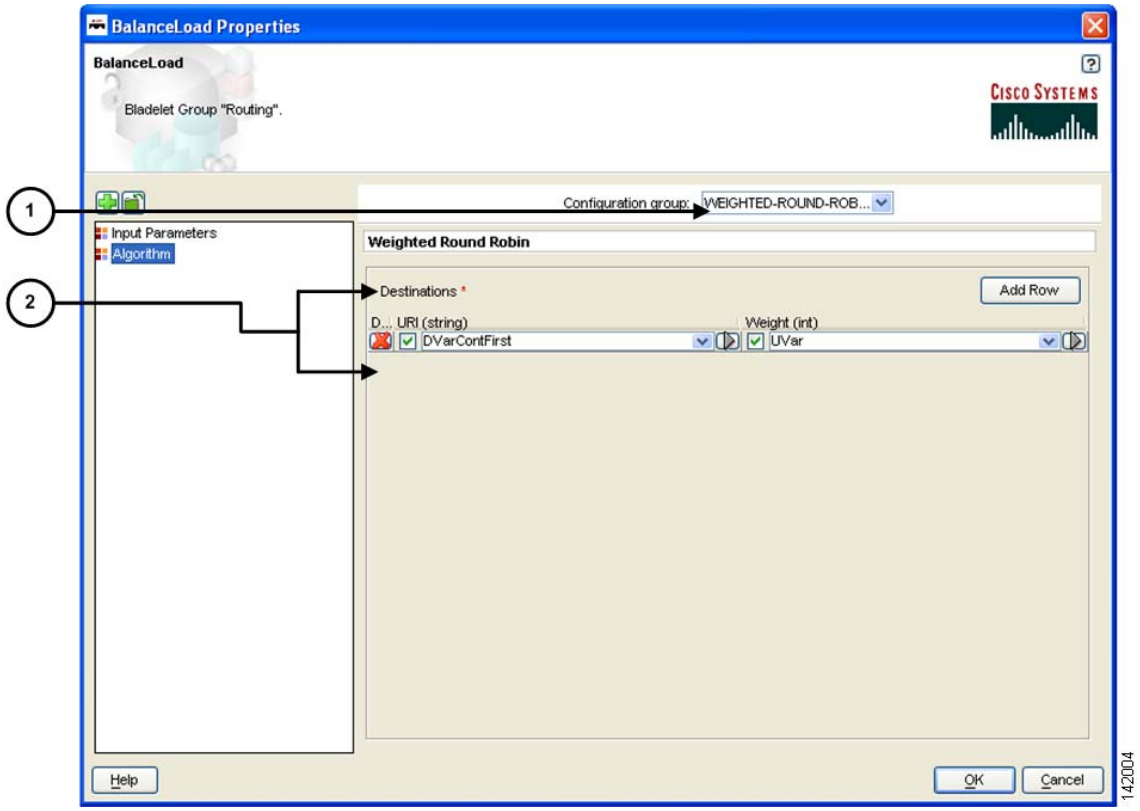
6	Cookie Name	Name of the cookie that carries the session information (example: in Unit3 this is JSESSIONID) in the response message headers.
7	Length of Session ID in Cookie	If the session ID is only a part of the cookie value (as opposed to being the whole cookie value), length of the session ID within the cookie value. Need not be specified if the session ID is the entire cookie value such as Unit3. Applies only to response message headers, associated with the Cookie Name parameter.
8	Offset of Session ID in Cookie	If the session ID is only a part of the cookie value (as opposed to being the whole cookie value), offset from where the session ID starts in the cookie value. Need not be specified if the session ID is the entire cookie value such as Unit3. Applies only to response message headers, associated with the Cookie Name parameter.

Figure 2-67 LoadBalancing Properties Window—Algorithm, Round-Robin



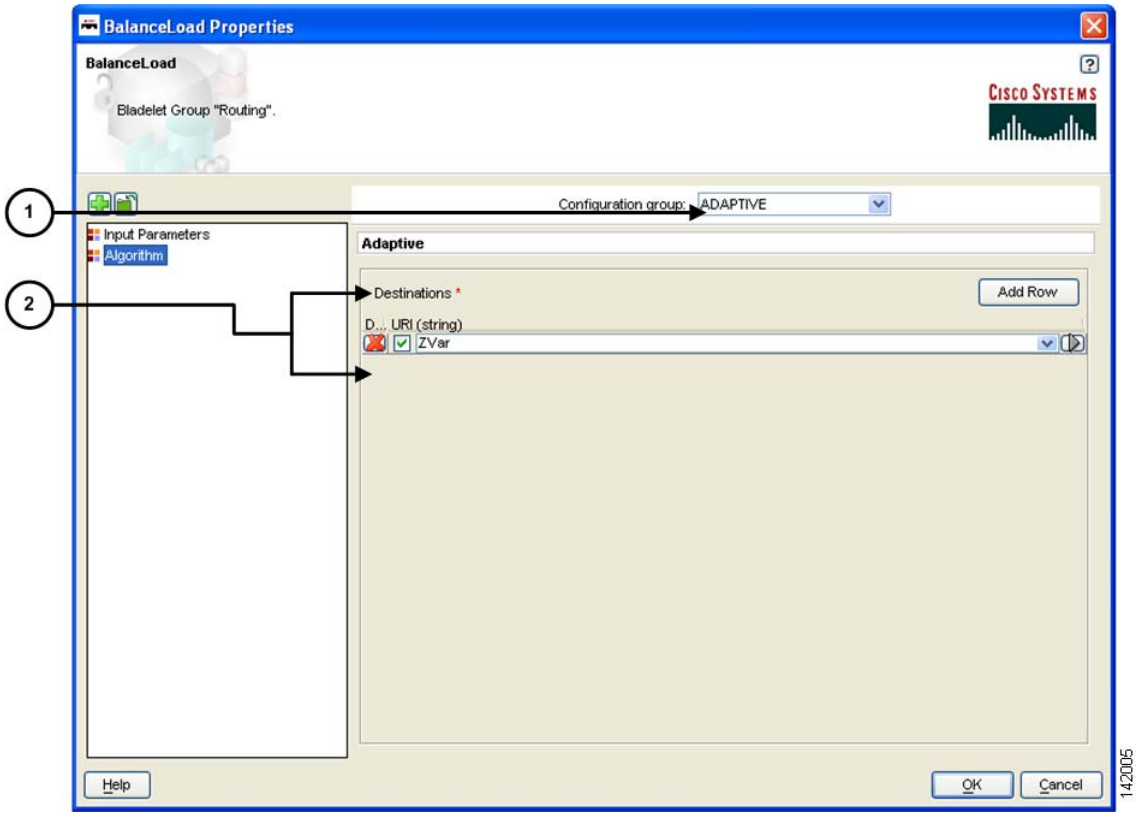
1	Configuration Group	Configuration group, set here to Round-Robin.
2	Destinations	One or more destination URIs to be load-balanced, based on the following: • Endpoint with least response time • Endpoint with least average wait time (when concurrency > 1)

Figure 2-68 LoadBalancing Properties Window—Algorithm, Weighted Round-Robin



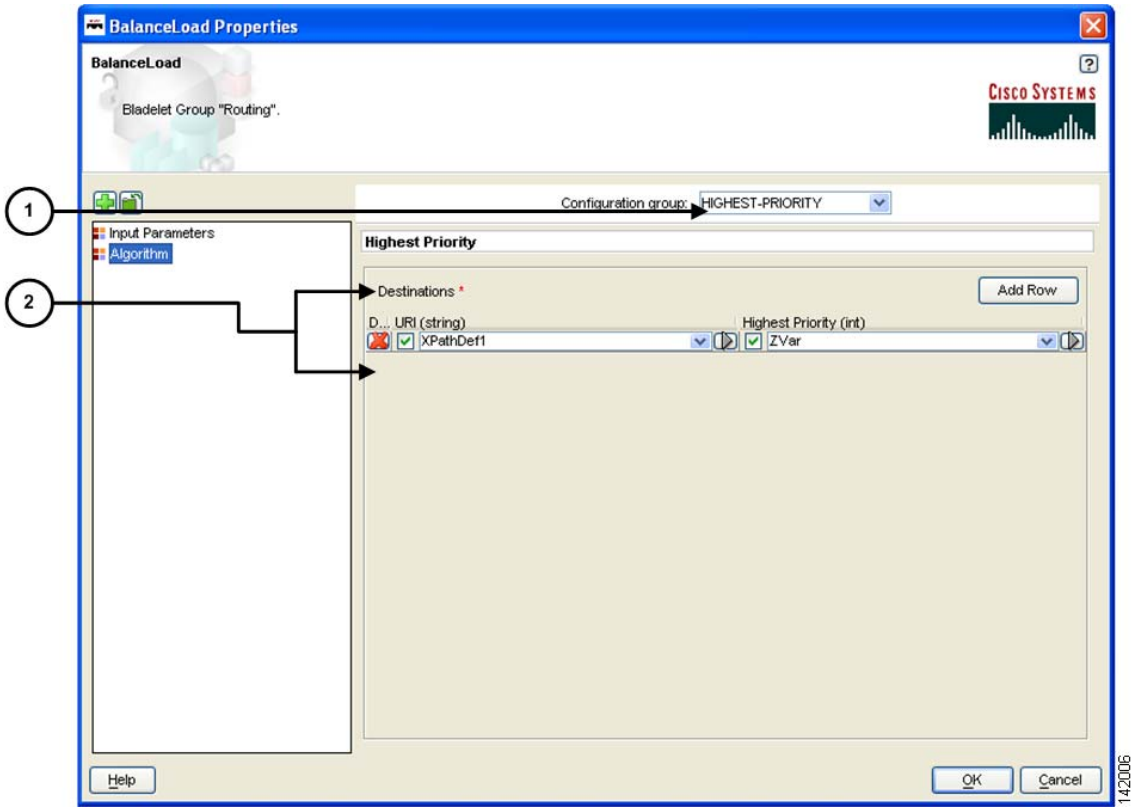
1	Configuration Group	Configuration group, set here to Weighted-Round-Robin.
2	Destinations	One or more URIs (string) and weight (int). This algorithm picks the destination based the corresponding weights. Distribution of messages to the destination is in proportion to the corresponding weight. So, if the weights of two destinations are 1 and 2, the destination with weight 2 gets twice as many requests as the one with weight 1.

Figure 2-69 Balance Load Properties Window—Algorithm, Adaptive



1	Configuration Group	Configuration group, set here to Adaptive.
2	Destinations	One ore more destination URIs, such as ZVar.

Figure 2-70 Balance Load Properties Window—Algorithm, Highest Priority



1	Configuration Group	Configuration group, set here to Highest-Priority.
2	Destinations	One or more URIs (string) and Highest Priority (int). This algorithm picks the destination with the highest priority (highest integer value in the priority column) that is currently available. If the destination with a higher priority is unavailable, the destination with the next-highest priority is picked.

Outcome

- On success, the destination URI of the input message is updated to be the one chosen by the BalanceLoad algorithm.
- This Bladelet also performs failover, so if the first endpoint chosen is not available to serve the request, BalanceLoad and Send work together to go through the rest of the destinations to find one that is available. If all endpoints are unavailable, the client receives an error.

Exceptions

None.

Security Category

In the Security category, there are eight Bladelets:

Authorize

Summary

The Authorize Bladelet uses access control to secure application resources in the PEP and is able to execute AON authorization procedures and other authorization-type procedures. By comparing authorization policies within the message to those within the PEP, the Authorize Bladelet is able to determine the proper course of action based on authorization.

Authorize Bladelet in AON provides the function of computing authorization decisions and enforcing authorization decisions on an incoming message. It supports three different authorization mechanisms as described in the Details section.

Prerequisites and Dependencies

- For LDAP-Based and SAML-Based authorization, provide AONSSubject. AONSSubject in case of SAML-Based authorization specifies a SAML Authorization Assertion that is verified by the Bladelet. In case of LDAP-Based, AONSSubject must specify a user object in LDAP repository. Use the Identify Bladelet to extract the identities present in the message.
- For LDAP-Based authorization, use the AMC server to define LDAP property sets that specify the LDAP configuration parameters. The full path in AMC is **Properties > Authorization & Authentication > LDAP**.
- For SAML-Based authorization, either verify the SAML assertions by using Identity Verify Bladelet in PEP before the Authorizer Bladelet or use Authorizer Bladelet to verify the signature of the assertion.

Details

SAML-Based Authorization

Identify Bladelet extracts the SAML Token containing Authorization Assertion, which can be signed or unsigned. You can configure Authorizer to accept an unsigned assertion, in which case it processes a SAML Assertion and enforces it even if it is not signed by a SAML Authority.

Once the SAML Assertion is extracted by the Identify Bladelet, it can be verified by an Identity Verify Bladelet before passing to Authorizer. However, if verification is not done at that point, it performs the signature verification of the SAML Assertion if it is needed.

Authorizer enforces the authorization decision specified in the SAML Assertion by ensuring that resource to authorize is allowed Permit access in the SAML Assertion and the Action configured in the Bladelet matches the Action in the assertion.

If it results in the Deny access then corresponding output path is set on the Authorizer Bladelet.

LDAP GROUP-Based Authorization

LDAP Group Based Policy Rules defines Authorization Policies based on the subject's group membership in an LDAP Directory. Such a Rule essentially is a Policy Rule that comprises of Rule Condition and Rule Action where Rule Action specifies one or more LDAP Groups to allow the access.

If all the conditions specified in the policy rule evaluate to true, then list of the groups specified Active Group Name parameter are allowed access. If you specified in the Subject to Authorize is a member of any of the groups that allowed access, access is allowed.

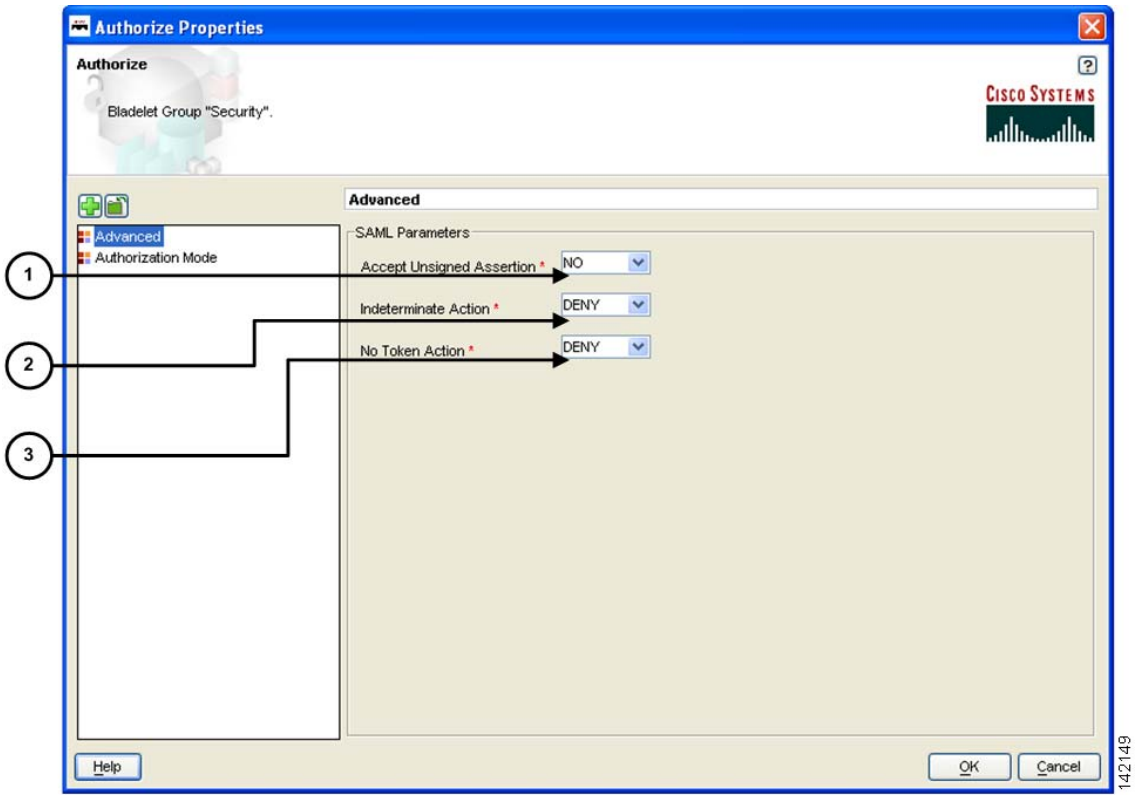
RULE-Based Authorization

Authorizer can make authorization decision based on the results of evaluation of Content Rules specified on Authorizer. Content Rule essentially is a policy rule that comprises of Rule Conditions and a Rule Action. When a Policy Rule is selected for evaluation, all its conditions are evaluated and, if all evaluate to true, Rule Action can be taken.

Rule Action may specify if the Authorize should result in PERMIT or DENY of the Authorization decision. Based on the Rule Action specified and results of Rule Condition evaluation Authorizer sets the output path of Authorizer Bladelet.

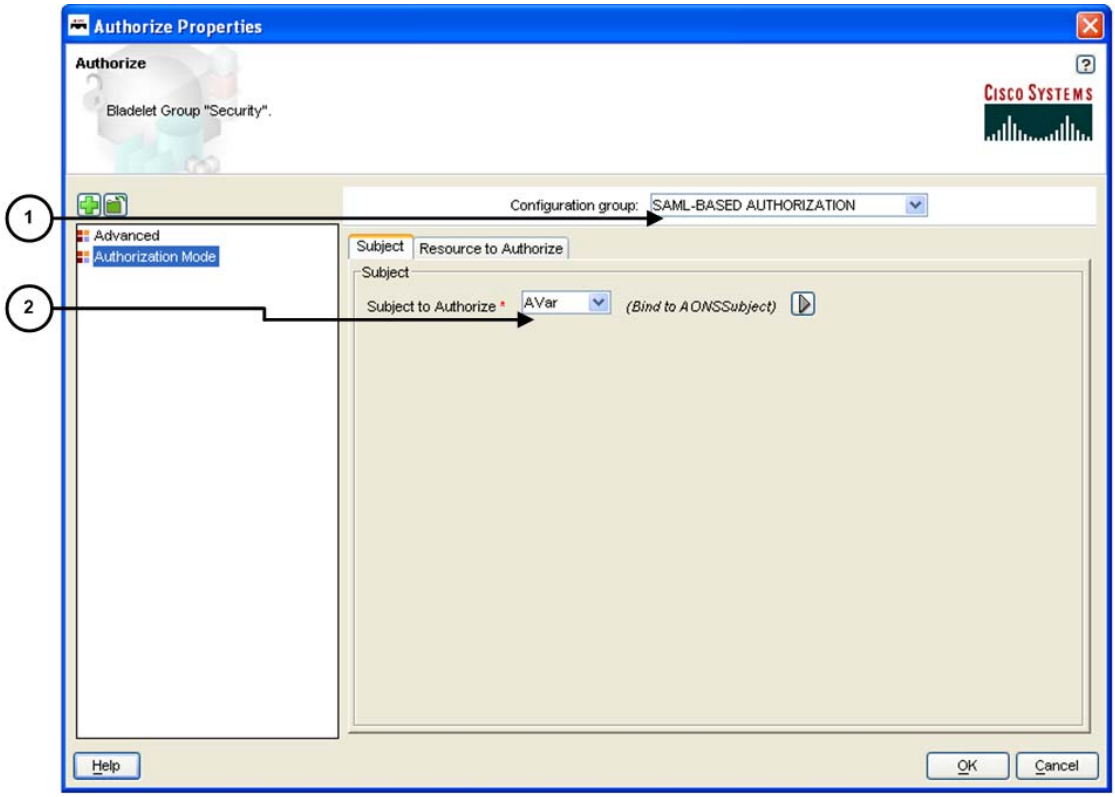
This section defines SAML Parameters that are used only in case of SAML-Based authorization.

Figure 2-71 Authorize Properties Window—Advanced



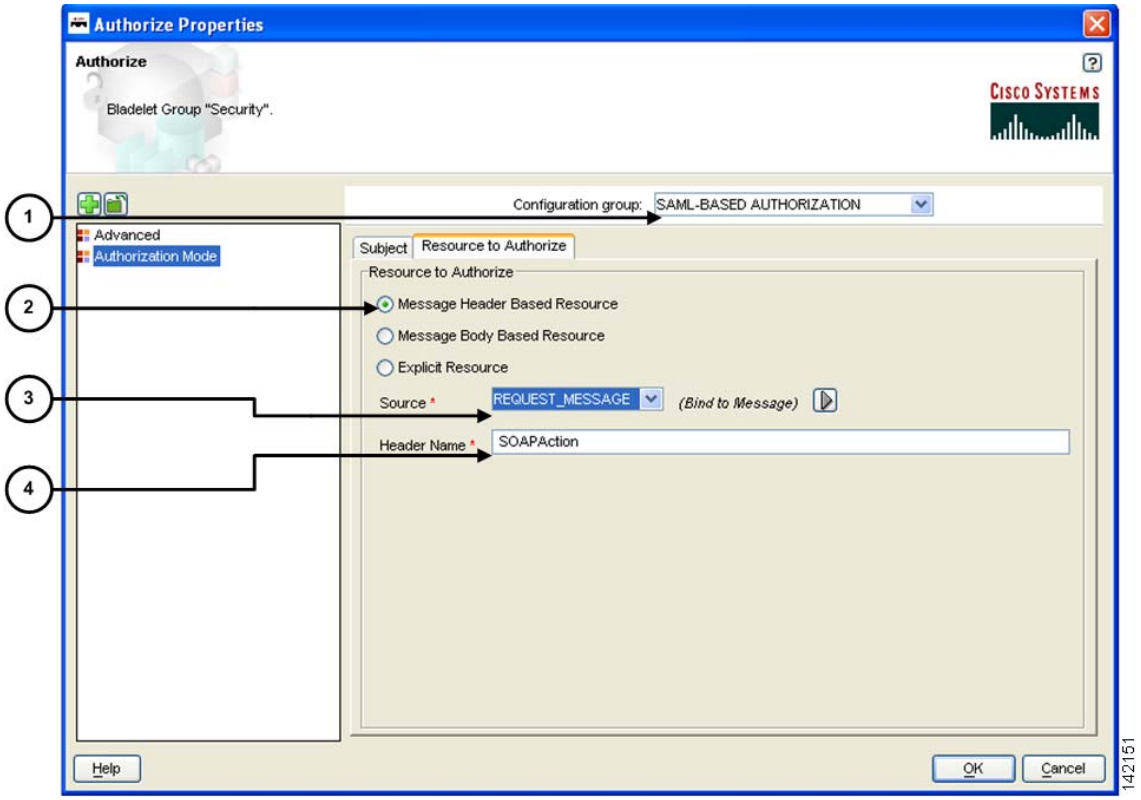
1	Accept Unsigned Assertion	Whether or not to accept an unsigned SAML assertion in the message: • Yes—Accepts an assertion even if it is not signed. • No—Does not process and verify an assertion if it is not signed.
2	Indeterminate Action	Action that must be taken if the assertion verification results in an Indeterminate Action. It treats an Indeterminate Action as Deny or Permit based on the value of this parameter.
3	No Token Action	If the Authorization Mode is set to SAML-Based Authorization, and if no SAML assertion is found in the AONSSubject, then it can result in Deny or Permit based on the value of this parameter.

Figure 2-72 Authorize Properties Window—Authorization Mode, SAML-based Authorization 1



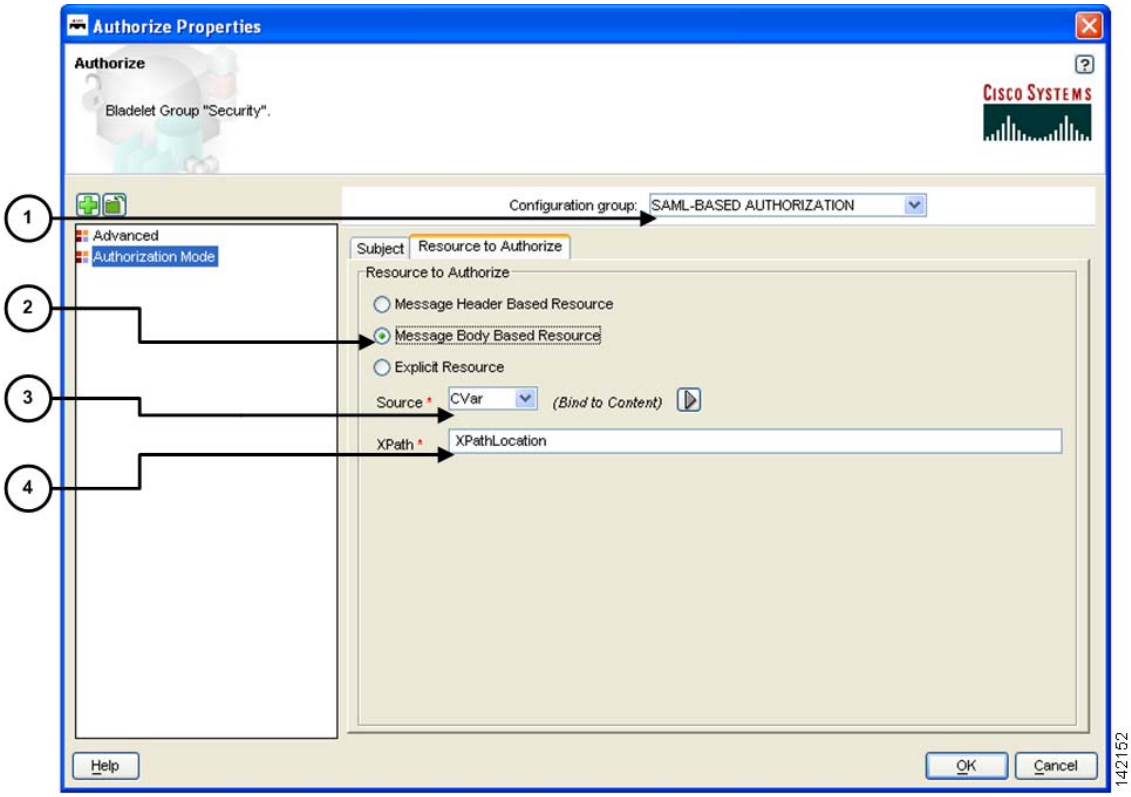
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Subject to Authorize	Subject to use for SAML authorization verification. Extract this subject before the authorization Bladelet is invoked in the PEP using Identify Bladelet.

Figure 2-73 Authorize Properties Window—Authorization Mode, SAML-based Authorization 2



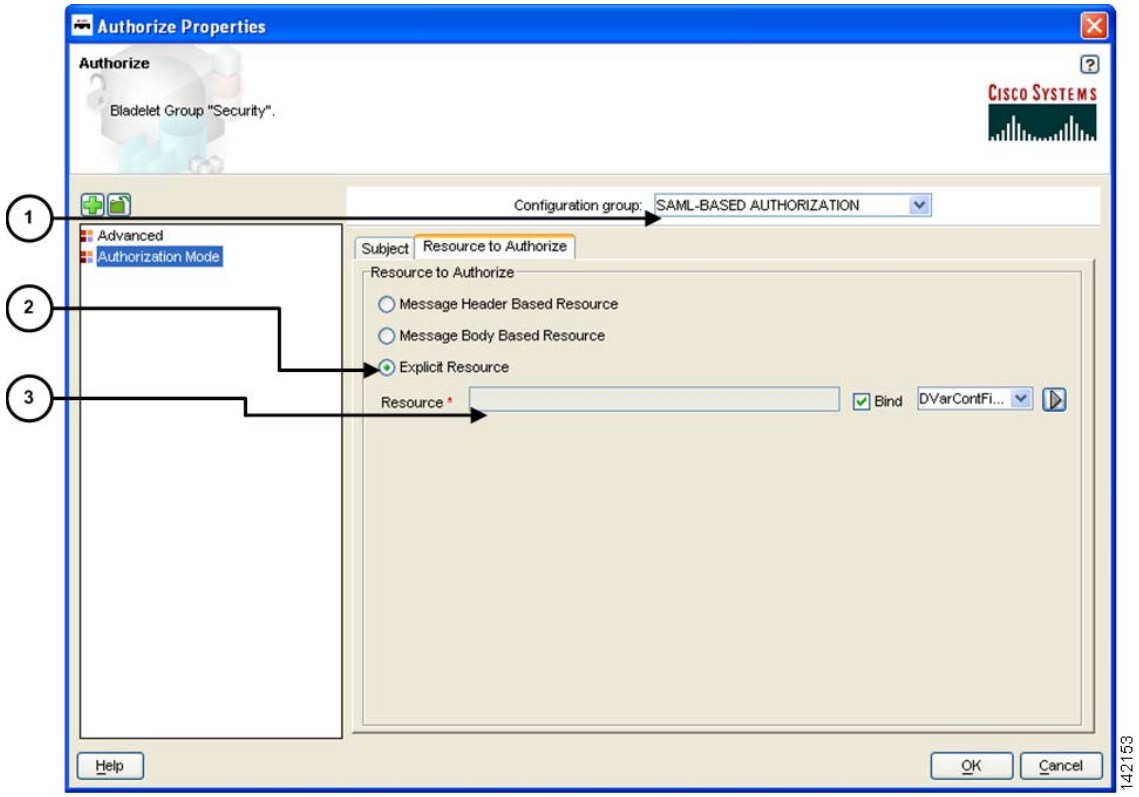
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Message Header Based Resource	Whether or not the resource to authorize is specified in the value of a message header field.
3	Source	Message that identifies the resource.
4	Header Name	Message header that contains the resource to authorize. By default, SOAPAction is specified as the header name.

Figure 2-74 Authorize Properties Window—Authorization Mode, SAML-based Authorization 3



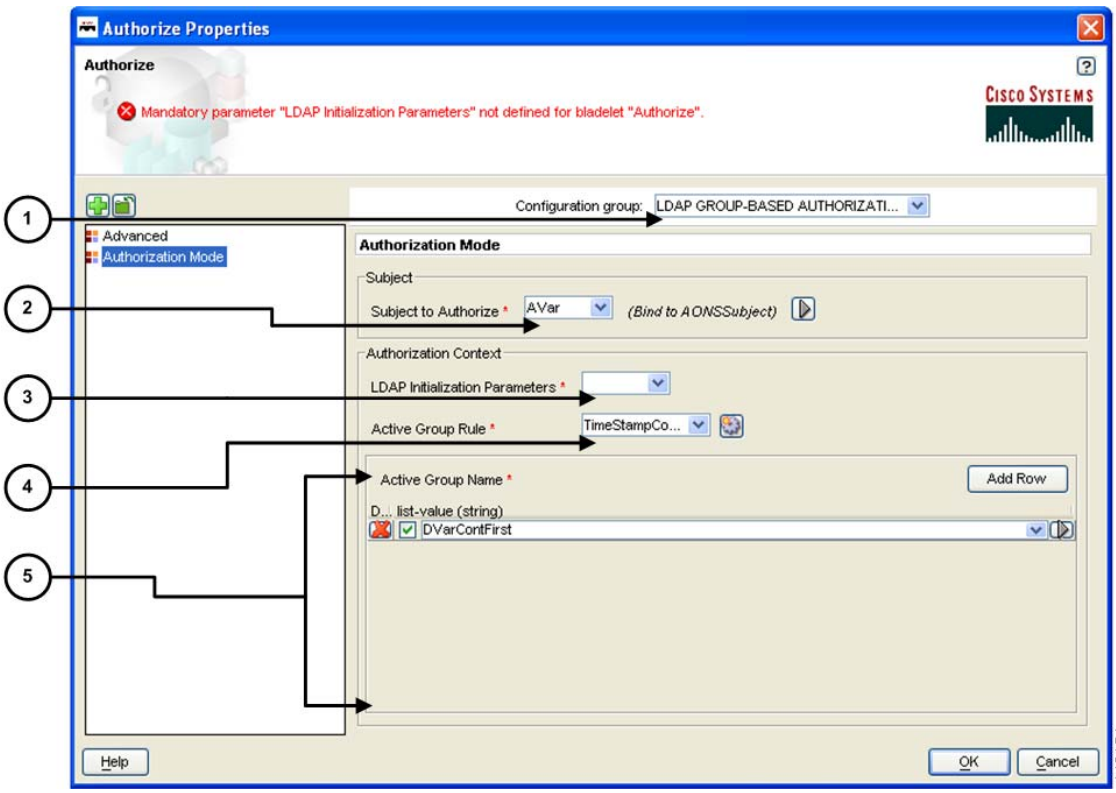
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Message Body Based Resource	Whether or not the resource to authorize is found in the message body. An XPath expression extracts the resource value from the message body.
3	Source	Message whose body contains the resource.
4	XPath	XPath expression that is applied on the message body to extract the resource value.

Figure 2-75 Authorize Properties Window—Authorization Mode, SAML-based Authorization 4



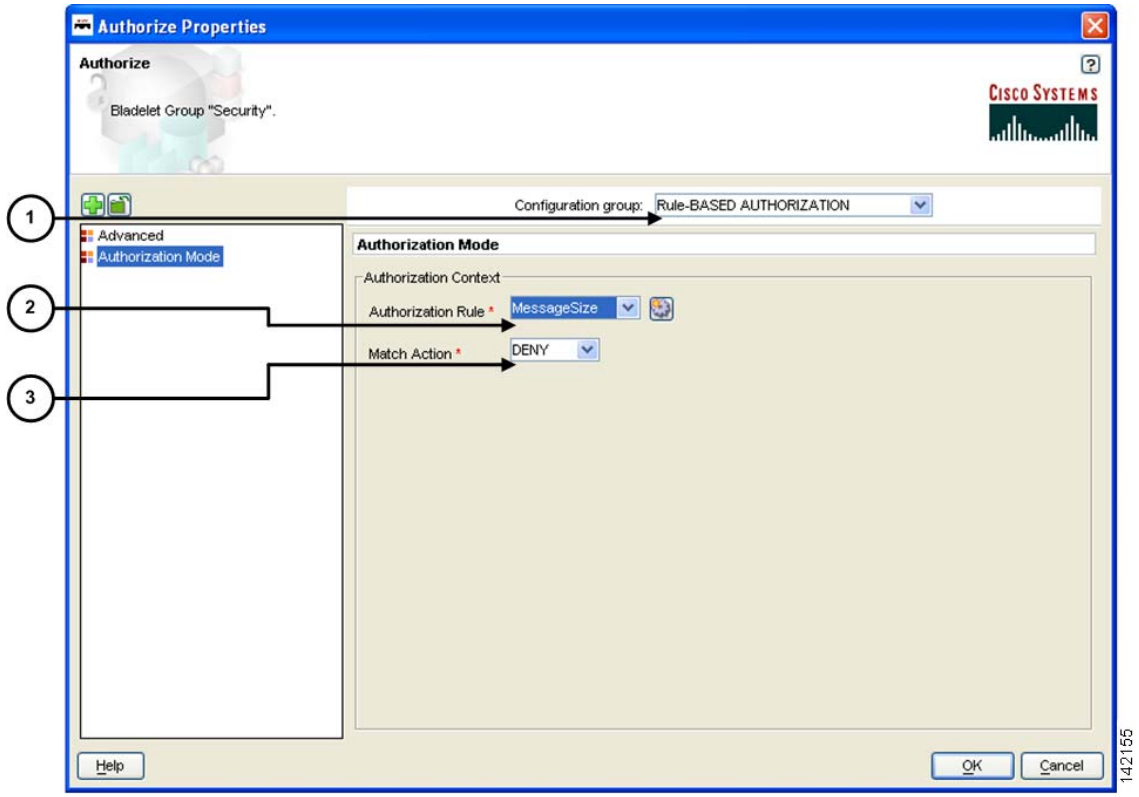
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Explicit Resource	Whether or not the resource to authorize is specified explicitly.
3	Resource	Resource value. Can be explicitly specified or it be bound to a PEP variable (String) that specifies the resource.

Figure 2-76 Authorize Properties Window—Authorization Mode, LDAP Group-Based Authorization



1	Configuration Group	Configuration group, set here to LDAP Group-Based Authorization.
2	Subject to Authorize	Subject to use for LDAP group-based authorization. This subject should be extracted before the authorization Bladelet is invoked in the PEP using Identify Bladelet. Further this subject should be a valid subject present in the LDAP repository specified by the LDAP Initialization Parameter below.
3	LDAP Initialization Parameters	Connection parameters to LDAP server. Also defines the configuration information used to access LDAP groups that you associated with AONSSubject (Subject to Authorize) occupies.
4	Active Group Rule	Policy rule that defines one or more conditions in a conjunctive expression that, if true, allow access to all the groups specified in the Active Group Name parameter. Select a displayed choice or add a rule by clicking the Rules Wizard icon.
5	Active Group Name	One or more user groups (list-value) in the LDAP repository that are allowed access if all conditions specified in Active Group Rule evaluate as true. Specify each group name by its distinguished name (DN).

Figure 2-77 Authorize Properties Window—Authorization Mode, Rule-Based Authorization



1	Configuration Group	Configuration group, set here to Rule-Based Authorization.
2	Authorization Rule	One or more conditions in a conjunctive expression. If all conditions evaluate as true, then action specified in Match Action parameter is taken.
3	Match Action	Action to be taken when all conditions specified in the authorization rule evaluate as true.

Outcome

- On success, a user is allowed or denied access to the resource.
  - If a user is allowed access, it sets the Success output path.
  - If a user is denied access, it sets the Fail output path.

Exceptions

None.

Encrypt

Summary

The Encrypt Bladelet encrypts all or parts of the input message to maintain data integrity. Encrypt parts of an XML or SOAP message by specifying the XPath locations of the elements to be encrypted in the message. AON can encrypt XML, SOAP and non-XML messages and their attachments.

Prerequisites and Dependencies

- If the Bladelet is configured to encrypt attachment content, ensure that an Extract Composite Content Bladelet exists in the PEP before this Encryp Bladelet. Configure the output of the Extract Composite Content Bladelet as input to the Encrypt Bladelet to encrypt the attachment content.
- Configure Encryption Policies and deploy them using the AMC server to send policies and keystores to AON.

Details

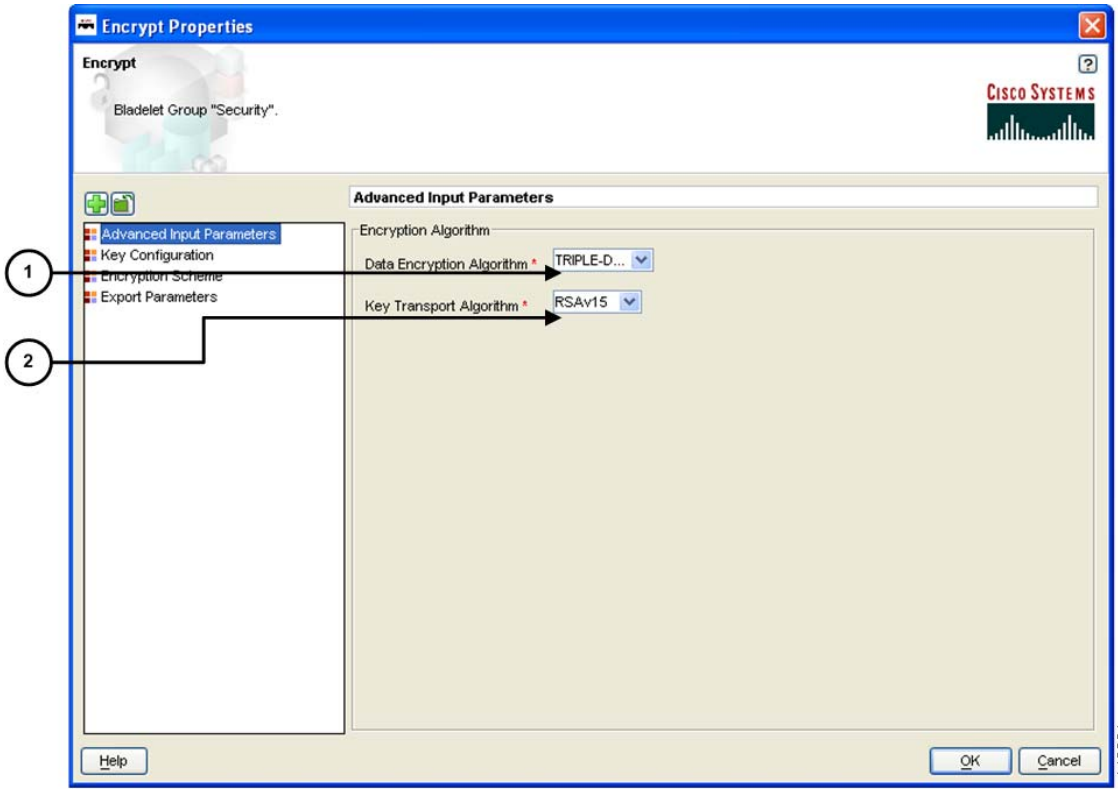
This Bladelet encrypts incoming SOAP, XML, and Non-XML messages using a dynamically generated symmetric key. The symmetric key is encrypted using the asymmetric public key of the message recipient. Given the public key of the recipient message as an input parameter, this Bladelet moves the CPU-intensive encryption operation to AON. Configure one or more elements in the message to be encrypted using XPath expressions.

Set Encrypt Bladelet's Output Content only if the output content is a MIME content. This happens for encrypting of SOAP with Attachments, XML with Attachments, non-XML and non-XML with Attachments.

For other cases (Xpath encrypting of SOAP and XML), the input content is modified in-place, so you need not create a new content variable. In such cases, use the Content that was passed as input to the Bladelet.

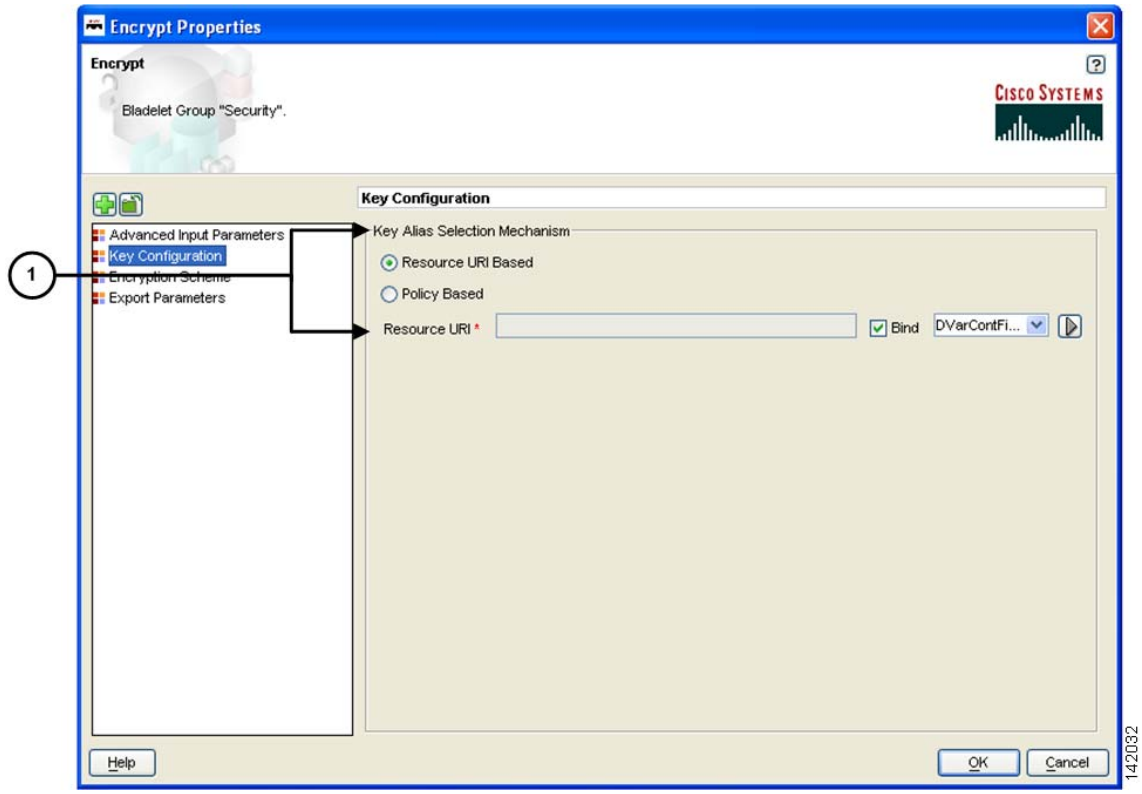
AON checks the destination URI of the message to determine the key alias for Encryption. For asymmetric key encryption, the encryption key alias is identical to the destination hostname. For example, if the destination URI is http://server1.domain.com/someservice, the encryption Bladelet expects an RSA key with the alias server1.domain.com in the keystore.

Figure 2-78 Encrypt Properties Window—Advanced Input Parameters



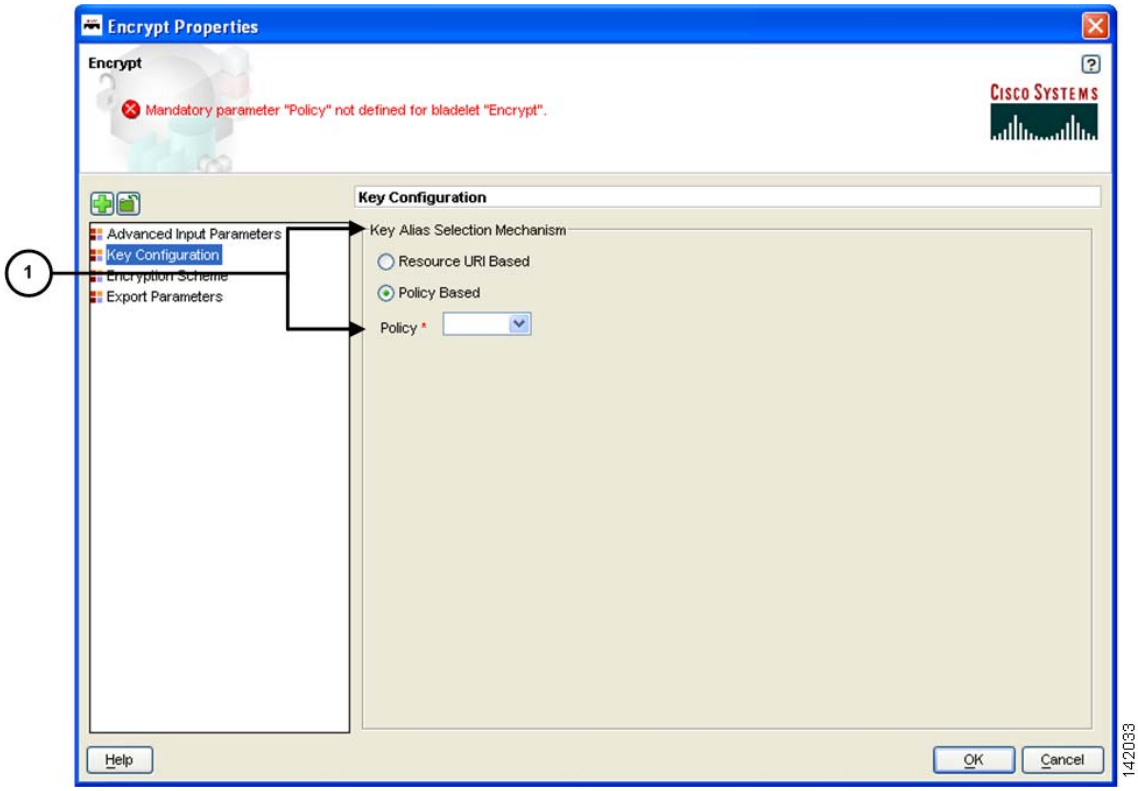
1	Data Encryption Algorithm	Algorithm used to encrypt the actual data. Choices: Triple-DES, AES128, AES192, and AES256.
2	Key Transport Algorithm	Encryption key. Currently only RSAv1.5 is supported.

Figure 2-79 Encrypt Properties Window—Key Configuration, Resource URI Based



1	Resource URI Based	URI of the intended recipient of this encrypted message. The key alias corresponding to this resource encrypts the symmetric key. Must already be configured on the AMC server.

Figure 2-80 Encrypt Properties Window—Key Configuration, Key Alias Policy Based

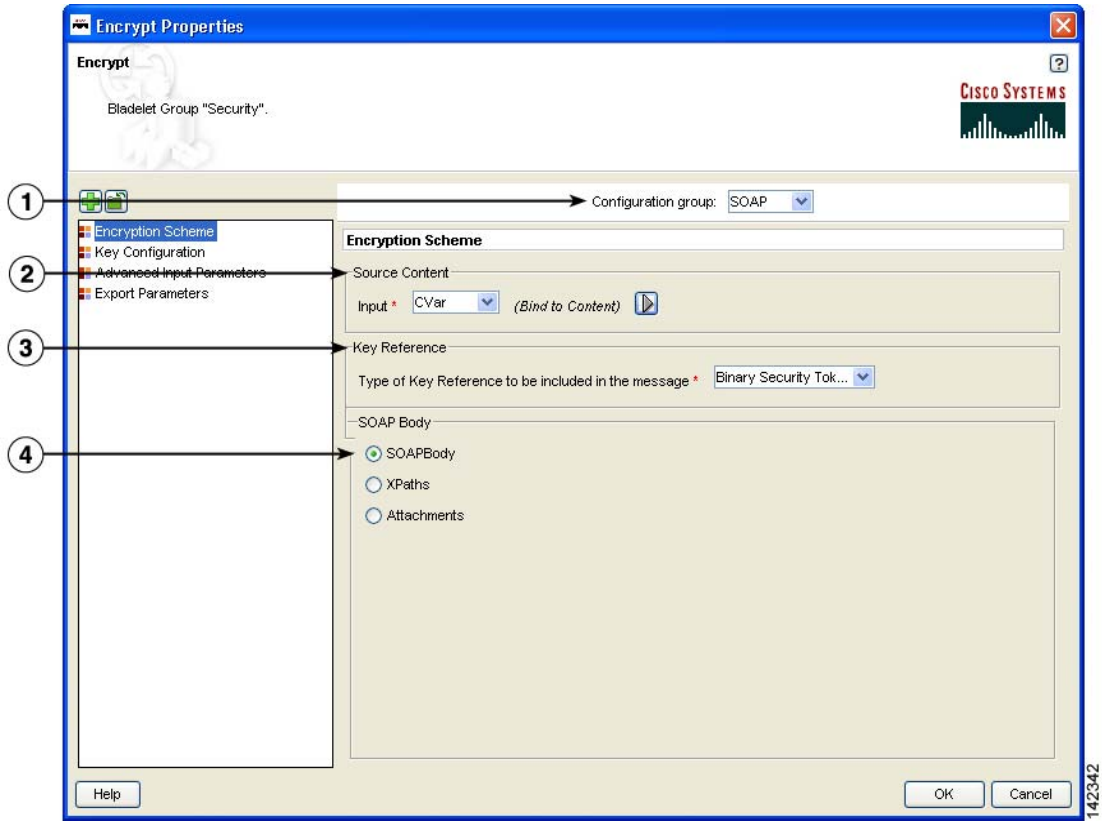


1	Policy Based	Reference of the encryption policy. The key alias in this policy encrypts the symmetric key, regardless of the resource URI that may be configured in this policy. Must already be configured on the AMC server.

Three types of Configuration groups in the Encryption Scheme section affect the way the settings are determined:

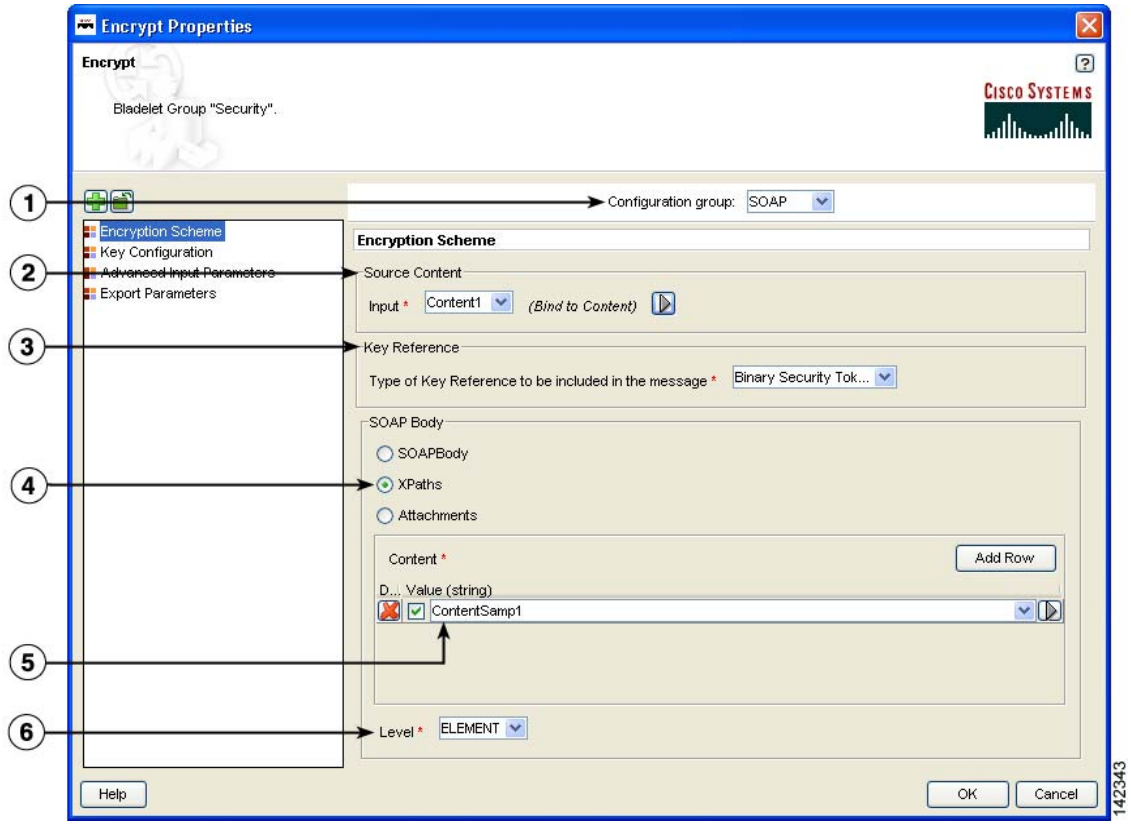
- SOAP (Figure 2-81 to Figure 2-83)
- XML (Figure 2-84 and Figure 2-85)
- Non-XML (Figure 2-86 and Figure 2-87)

Figure 2-81 Encrypt Properties Window—Encryption Scheme, SOAP, SOAPBody



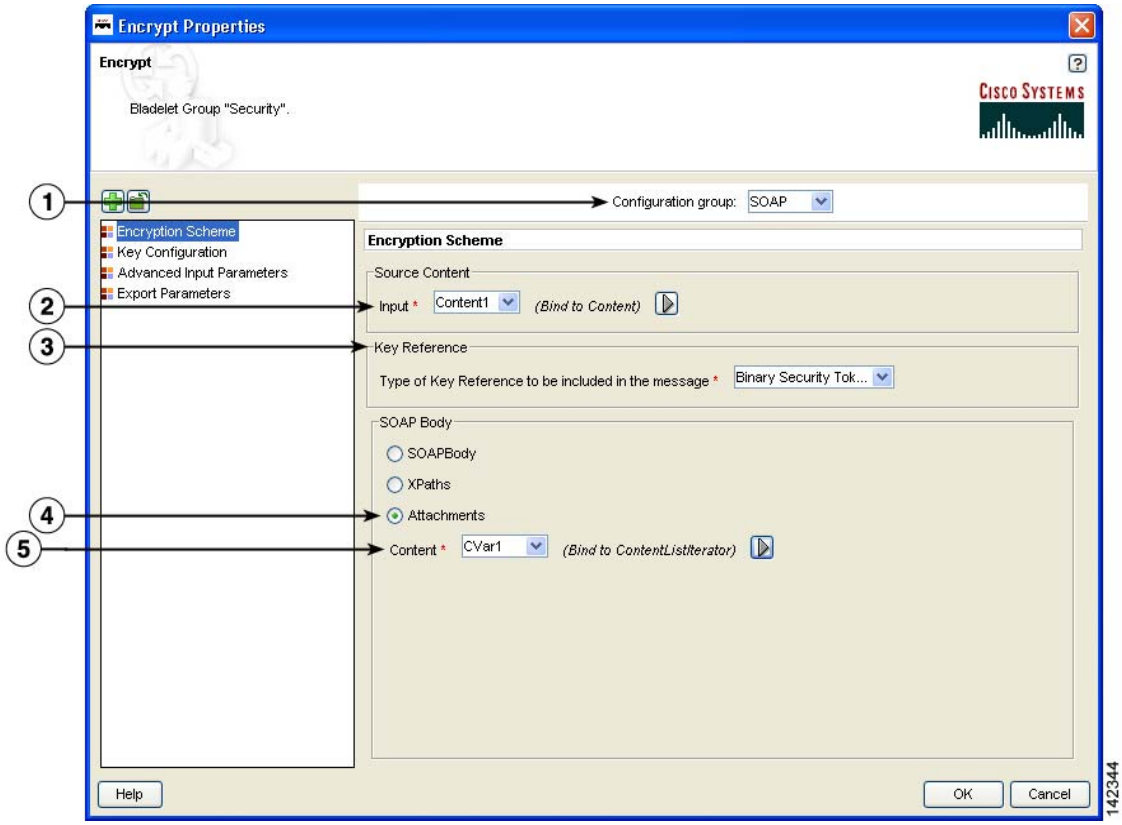
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	Content to be encrypted—XML or SOAP content containing the data that needs to be encrypted.
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	SOAPBody	SOAP Body: SOAP body encryption.

Figure 2-82 Encrypt Properties Window—Encryption Scheme, SOAP, XPaths



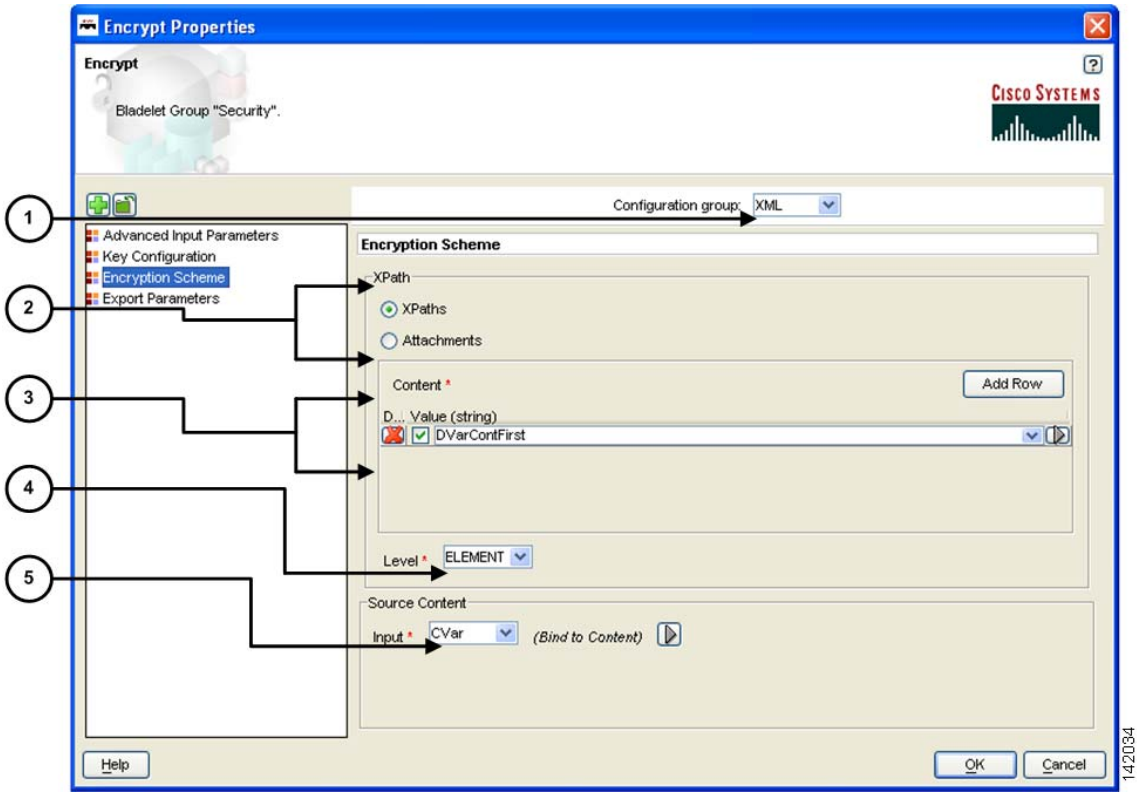
1	Configuration group	Configuration group, set here to SOAP.
2	Input	Content to be encrypted.
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	XPaths	XPath Locations of the elements to be encrypted in the SOAP message. You may add multiple XPath strings using the "Add Row" button.
5	Content	Content to be encrypted. May be an XML or SOAP content containing the data that needs to be encrypted.
6	Level	Level, set here is to Element. Level choices: • Element—Whole XML element needs to be encrypted, including the element name • Content—Only the contents of the XML element need to be encrypted; causes the name of the element to be shown

Figure 2-83 Encrypt Properties Window—Encryption Scheme, SOAP, Attachments



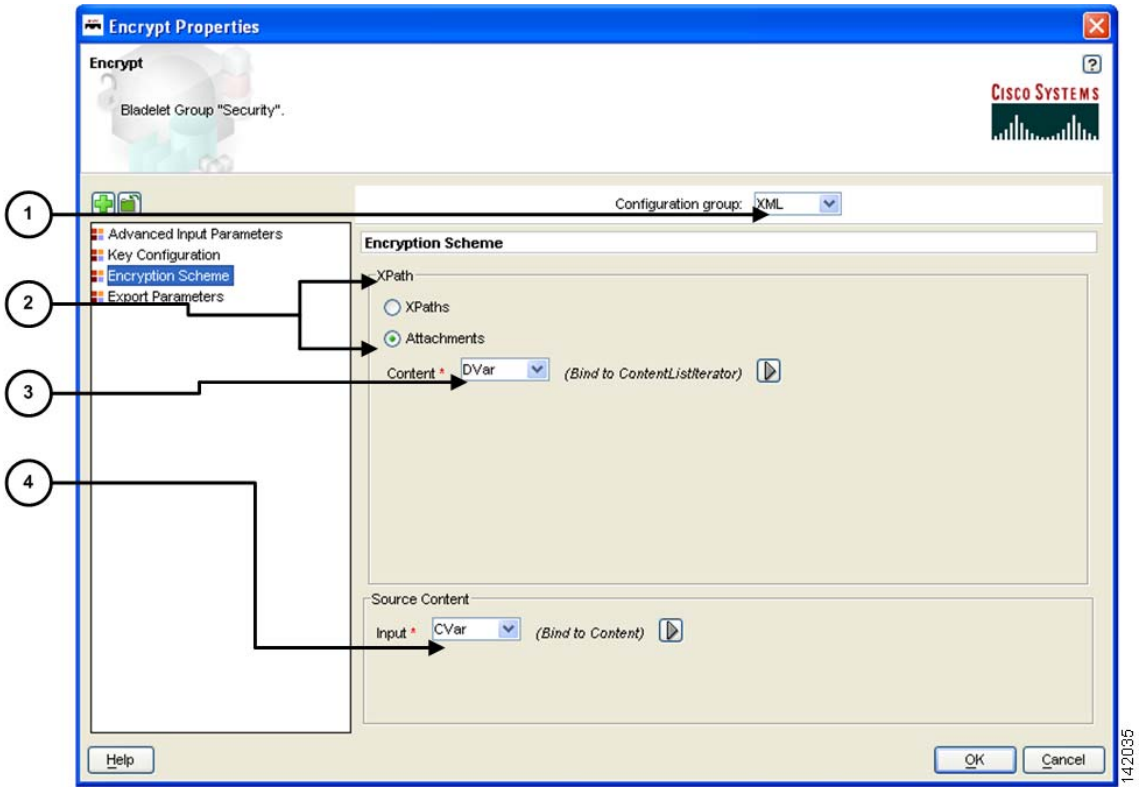
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	Content to be encrypted. Set if the encryption scheme is an attachment.
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	Attachments	List of attachments to be encrypted. This list is the output of a MIME-Extract Bladelet that should have preceded the Encryption Bladelet and extracted the attachments to be encrypted.
4	Content	Content to be encrypted. May be an XML or SOAP content containing the data that needs to be encrypted.

Figure 2-84 Encrypt Properties Window—Encryption Scheme, XML, XPaths



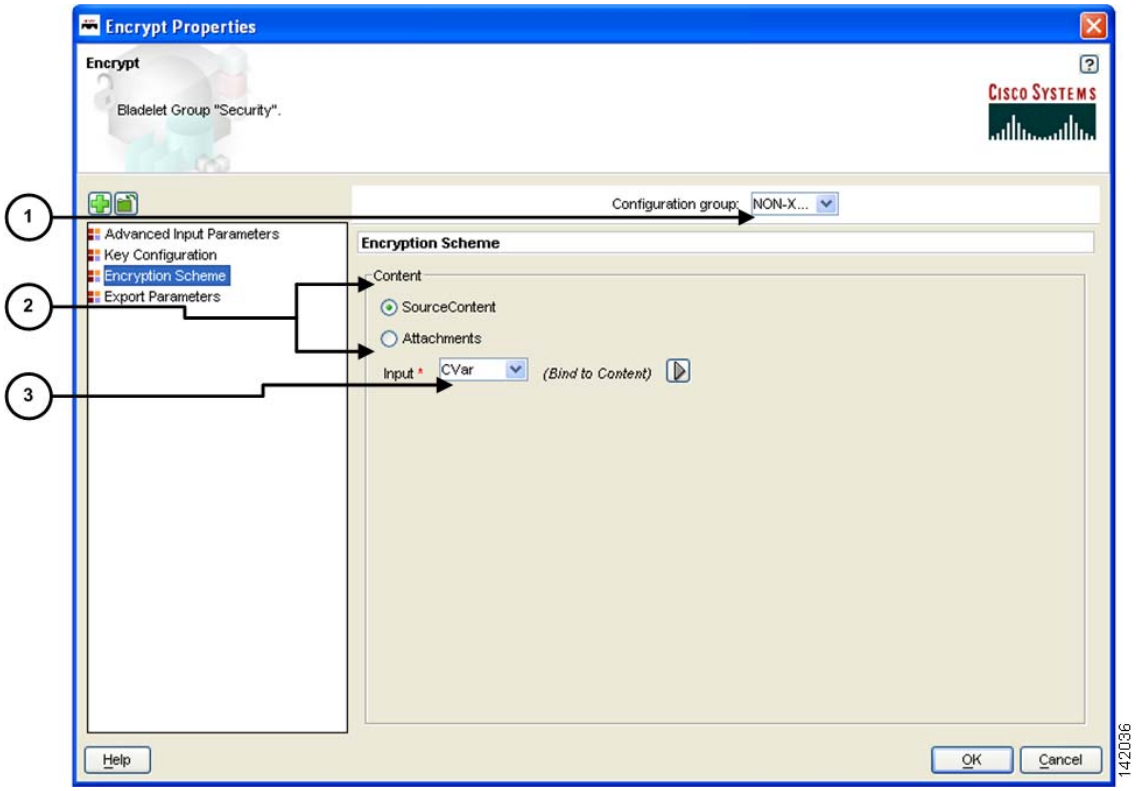
1	Configuration Group	Configuration group, set here to XML.
2	XPaths	XPath Locations of the elements to be encrypted in the XML message. You may add multiple XPath strings using the "Add Row" button.
3	Content	Content to be encrypted.
4	Level	Level: • Element—Whole XML element needs to be encrypted, including the element name. • Content—Only the contents of the XML element needs to be encrypted; causes the name of the element to be shown.
5	Input	Content to be encrypted. May be an XML or SOAP content containing the data that needs to be encrypted.

Figure 2-85 Encrypt Properties Window—Encryption Scheme, XML, Attachments



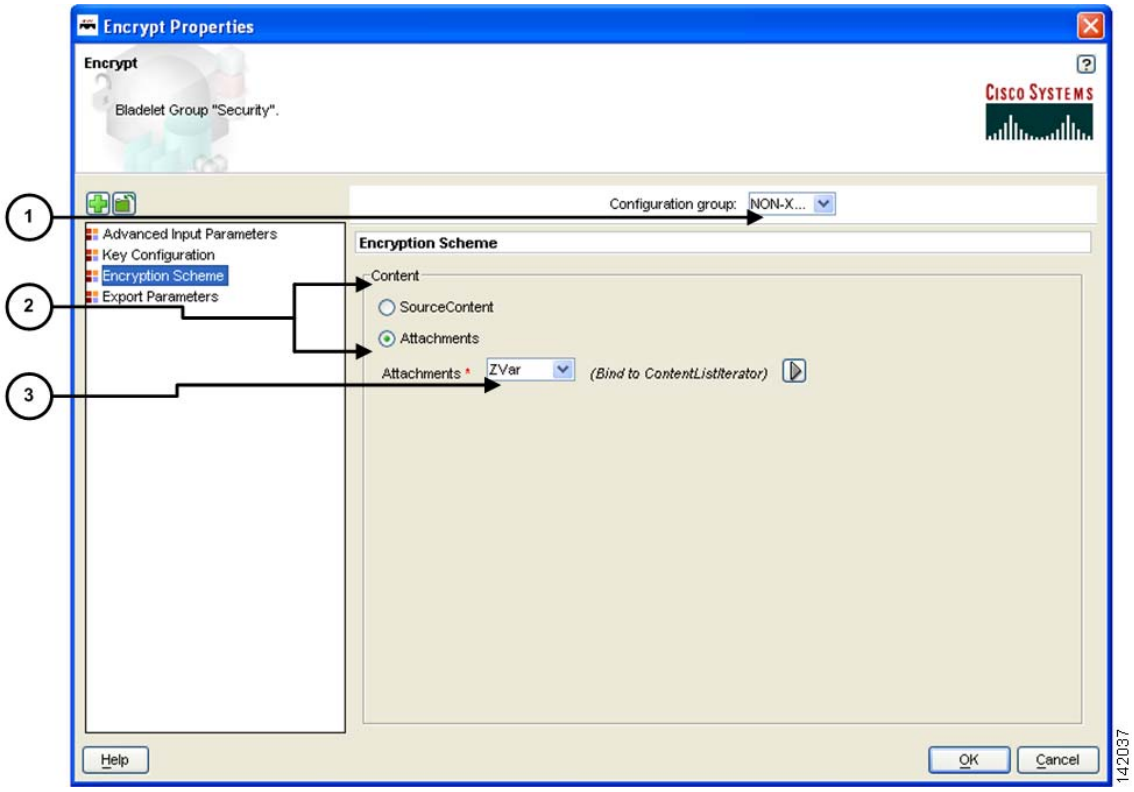
1	Configuration Group	Configuration group, set here to XML.
2	Attachments	List of attachments to be encrypted. This list is the output of a MIME-Extract Bladelet that should have preceded the Encryption Bladelet and extracted the attachments to be encrypted.
3	Content	Content to be encrypted.
4	Input	Content to be encrypted. May be an XML or SOAP content containing the data that needs to be encrypted.

Figure 2-86 Encrypt Properties Window—Encryption Scheme, Non-XML, Source Content



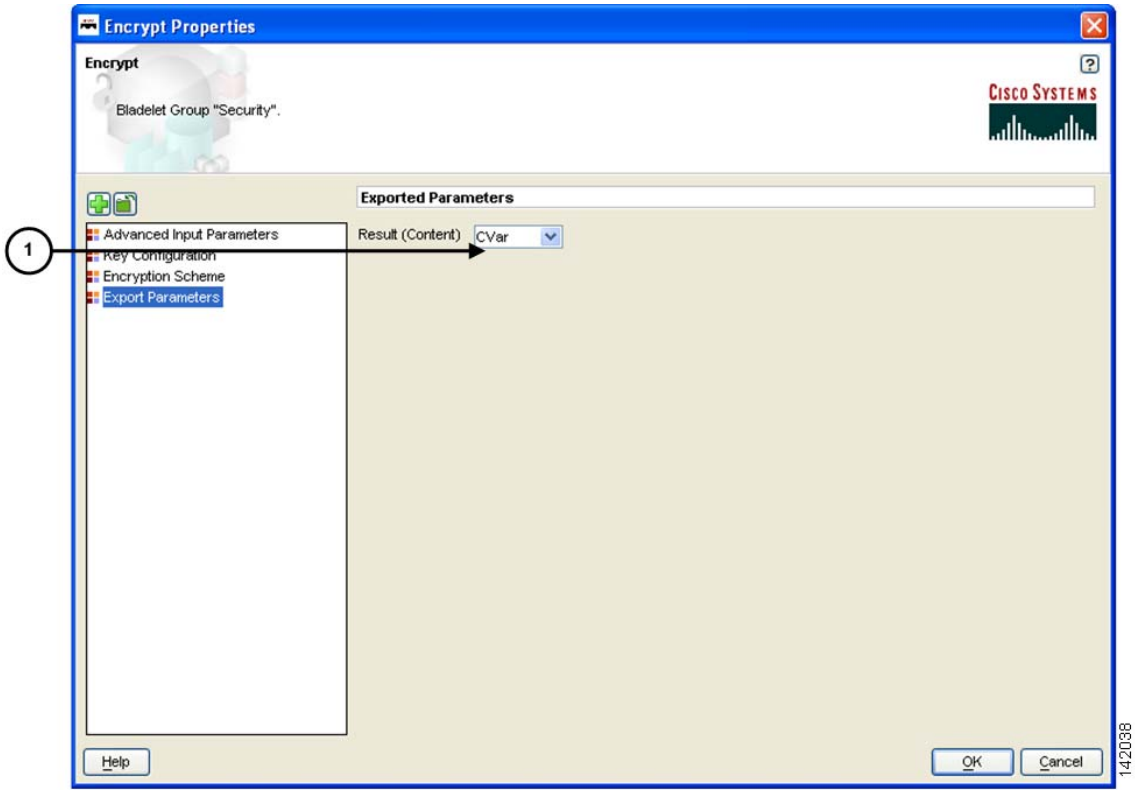
1	Configuration Group	Configuration group, set here to Non-XML.
2	Source Content	Non-XML content to be encrypted, if the content is of type non-XML.
3	Input	Content to be encrypted. This may be an XML or SOAP content containing the data that needs to be encrypted.

Figure 2-87 Encrypt Properties Window—Encryption Scheme, Non-XML, Attachments



1	Configuration Group	Configuration group, set here to Non-XML.
2	Attachments	List of attachments to be encrypted. This list is the output of a MIME-Extract Bladelet that should have preceded the Encryption Bladelet and extracted the attachments to be encrypted.
3	Attachments	Type of attachment.

Figure 2-88 Encrypt Properties Window—Export Parameters



1	Result	Output variable that contains the encrypted output of this Bladelet. Must be set if attachments (XML, SOAP, or non-XML) are being encrypted. In all other cases, the input content type is modified inline to replace the original data with the encrypted data.

Outcome

- Success: Path taken if the Bladelet successfully encrypts the incoming message
- Failure: Path taken if the Bladelet is unable to encrypt the message for any reason

Exceptions

- Public Key Not Found: Path taken if the Bladelet is unable to find a public key to encrypt the symmetric key with. This may happen if the configuration for selecting an asymmetric key is incorrect or if the Encryption policy and keystore have not been correctly provisioned.
- Data Not Found: Path taken if the Bladelet is unable to find the data that was configured to be encrypted, in the message. This happens when one or more XPaths configured in the Bladelet configuration do not resolve to any elements in the message.

Verify Signature

Summary

As the name suggests, verify signature verifies digital signature contained in XML/SOAP/non-XML message.In summary:

- The signature verification Bladelet usually verifies all the signatures contained in the original message, including multipart and nonmultipart messages.
- If the XKMS verification is enabled, then the AON node should be capable of reaching external VERISIGN website.

Prerequisites and Dependencies

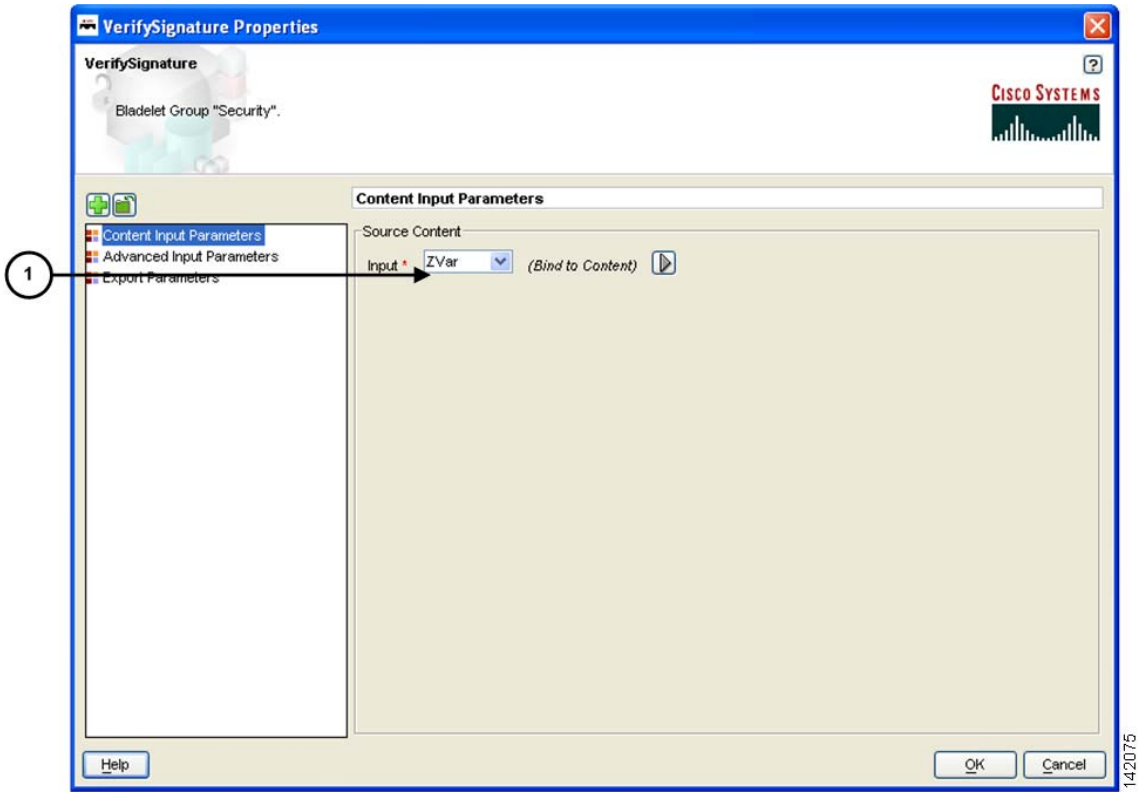
- Create and import necessary keystores.

Details

The Verify Signature Bladelet usually verifies all the signatures contained in the original message, including multipart and nonmultipart messages.

If the XKMS verification is enabled, then the AON node should be capable of reaching external VERISIGN website.

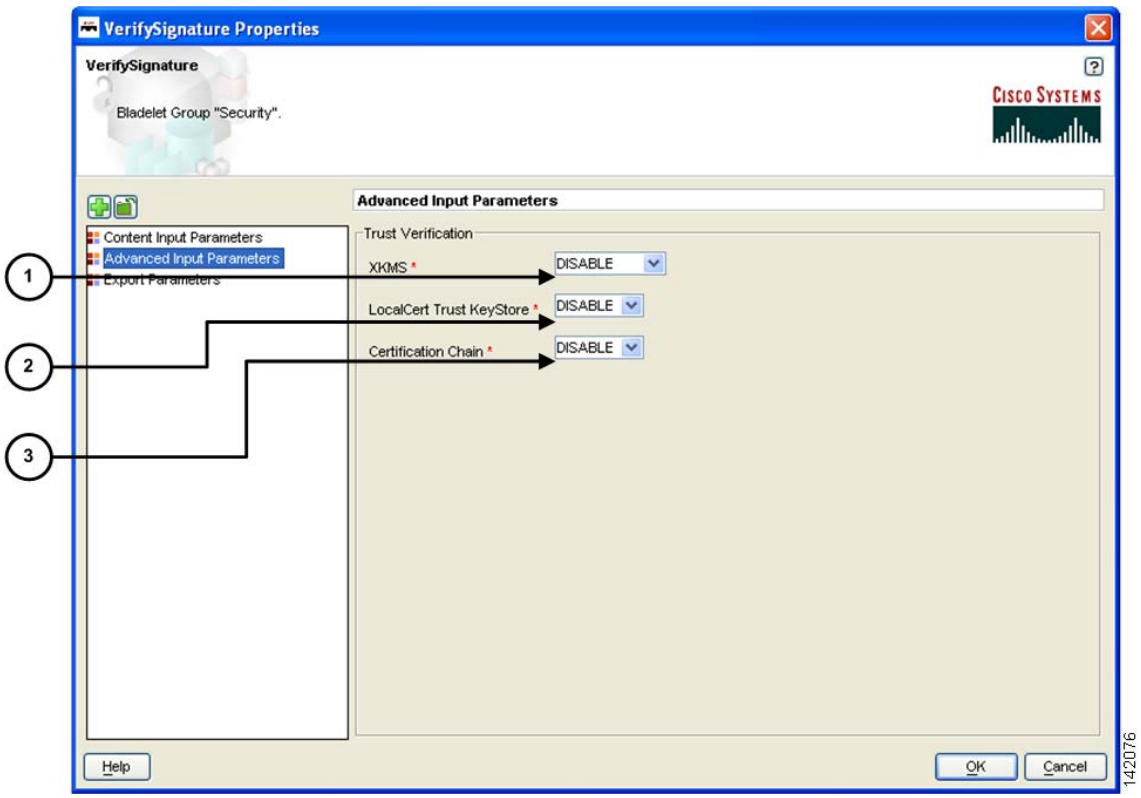
Figure 2-89 Verify Signature Properties Window—Content to Verify



1	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().

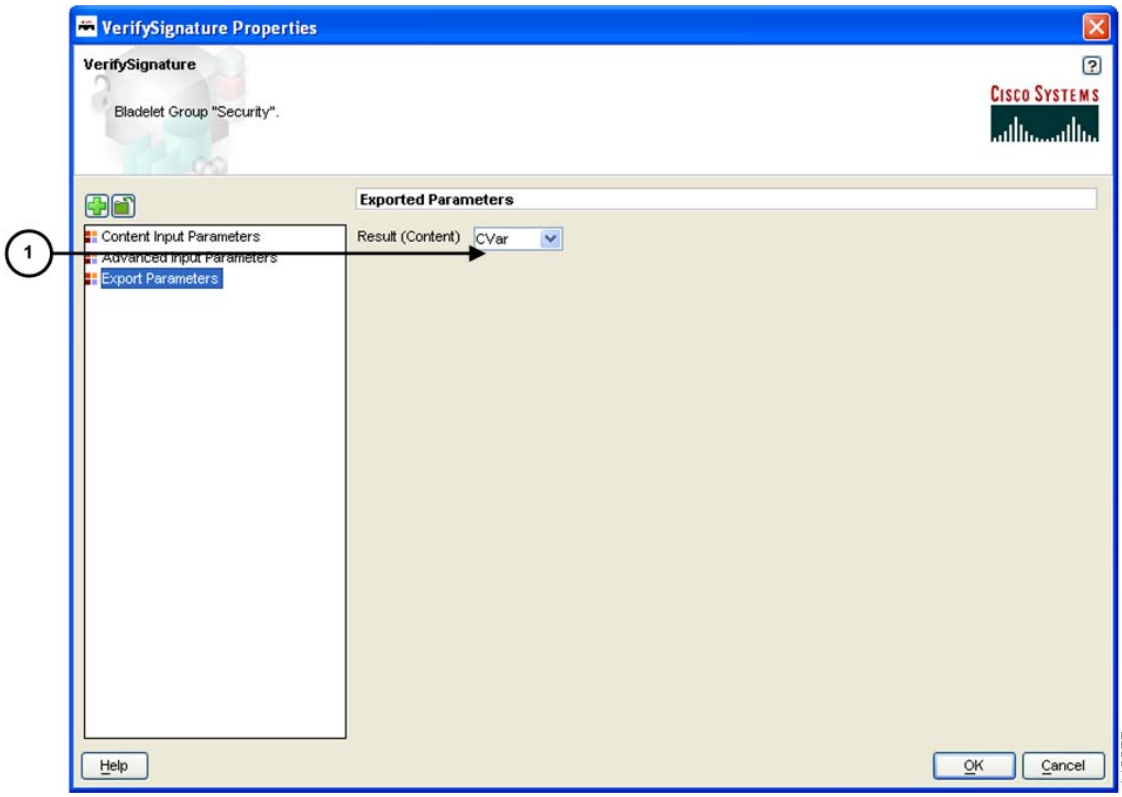
If Local Certificate Trust and/or Certificate Chain Verification is enabled, then configure the local Trust Store. The Certificate found in the Signature is expected to be found in the local Trust Store.

Figure 2-90 Verify Signature Properties Window—Advanced Input Parameters



1	XKMS	Whether or not XKMS-based trust verification is performed. Choices: Disable, Pilot, and Production.
2	Local Certificate Trust KeyStore	Whether or not local-trust-store verification is performed for the certificate used in the digital signature. Choices: Disable, Enable, and Both.
3	Certification Chain	Whether or not certificate-chain verification is performed. Choices: Disable, Enable, and Both.

Figure 2-91 Verify Signature Properties Window—Export Parameters



1	Result	New content that needs to be exported from the Bladelet.

Outcome

- On success, it verifies all signatures, then takes the success path.
- If even one signature verification fails, it takes the fail path.

Exceptions

- Signature Not Found: No signature information is available in the message.

Sign

Summary

Sign Bladelet basically creates digital signature on partial or entire SOAP/XML documents. This Bladelet is capable of signing non-XML and multipart messages. In summary:

- If the signing Bladelet signs relevant parts of MIME message, execute the ExtractCompositeContent Bladelet before the signing Bladelet so as to obtain contentListIterator variables that can be used in signing Bladelet.
- A new export variable should be created so as to contain the signed MIME message. This sign MIME message can integrated back into the original message by using the CreateMessage Bladelet.
- For non-MIME message, the original message is modified inline and hence no need to configure the export parameter.

Prerequisites and Dependencies

- Create and import necessary keystores and create a node based signing policy by configuring key alias to a particular key pair's key alias, existing in the keystore.
- If the original message is a MIME message, execute the Extract Composite Content Bladelet to extract the base content and interested attachment's contentListIterators.

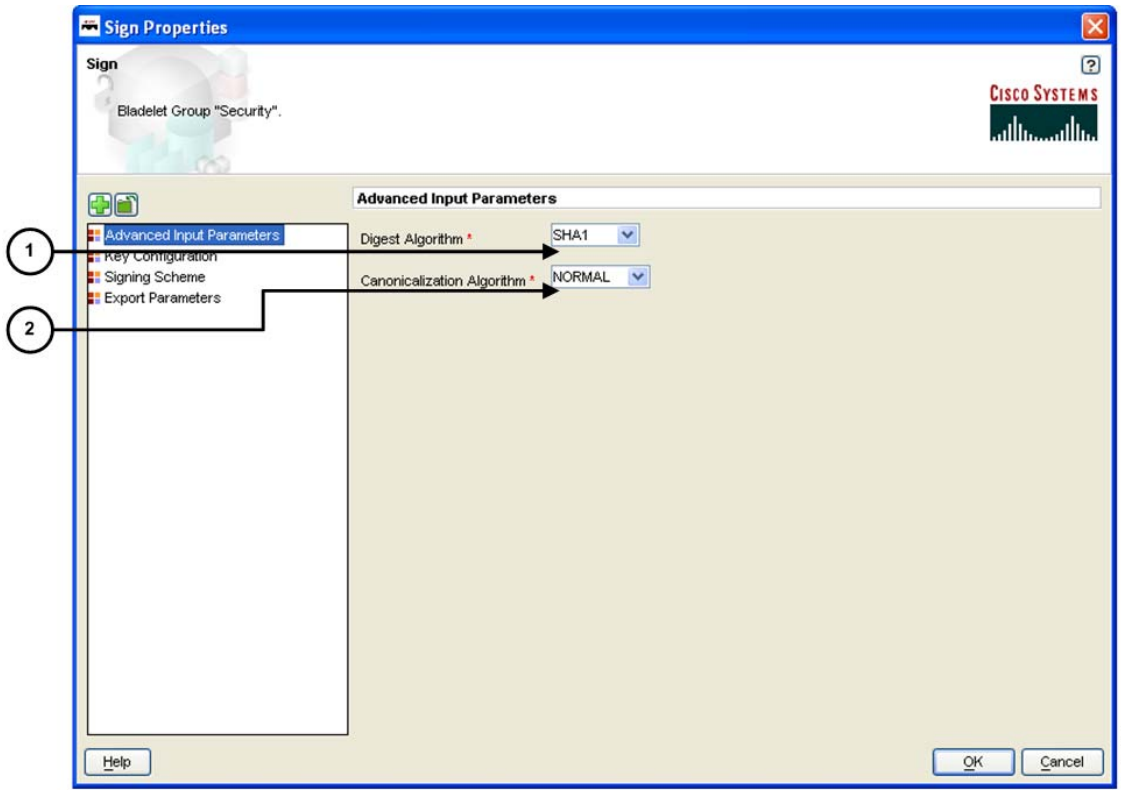
Details

If you use the Sign Bladelet to sign relevant parts of MIME message, execute the Extract Composite Content Bladelet before the signing Bladelet so as to obtain contentListIterator variables that can be used in the Sign Bladelet.

Create a new export variable to contain the signed MIME message. Integrate this signed MIME message back into the original message by using the Create Message Bladelet.

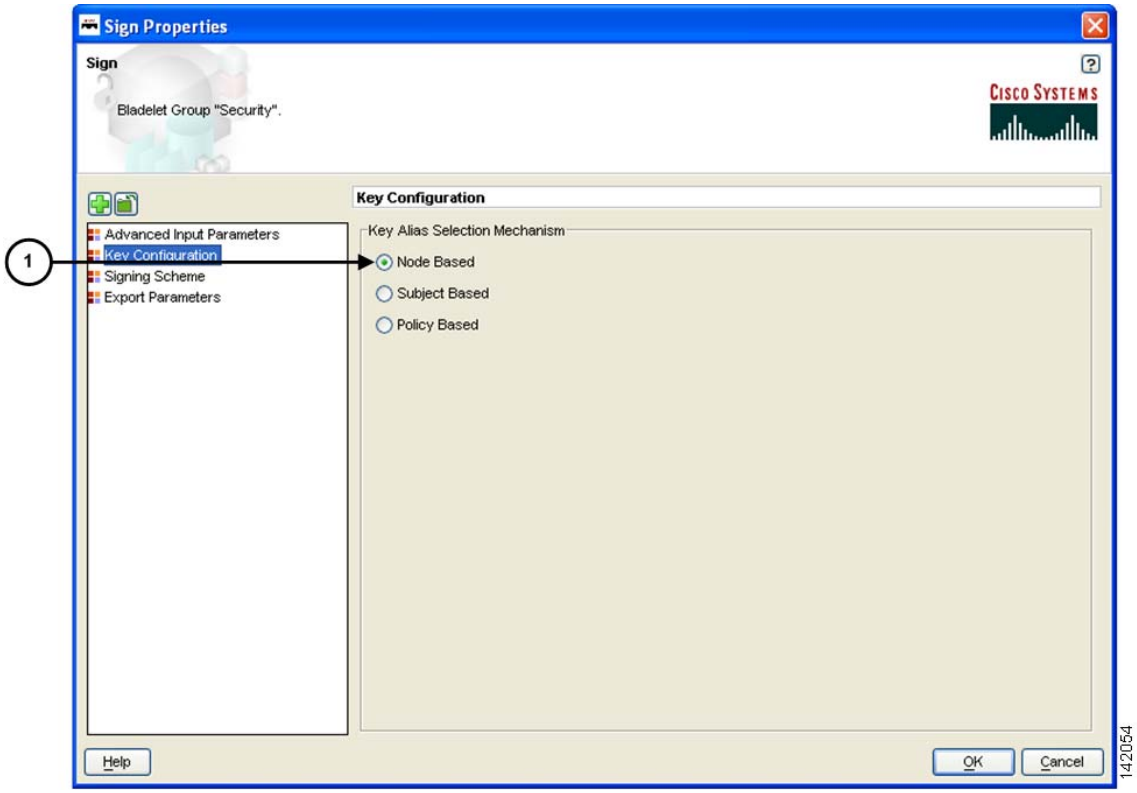
For a non-MIME message, the original message is modified inline and hence no need to configure the export parameter.

Figure 2-92 Sign Properties Window—Advanced Input Parameters



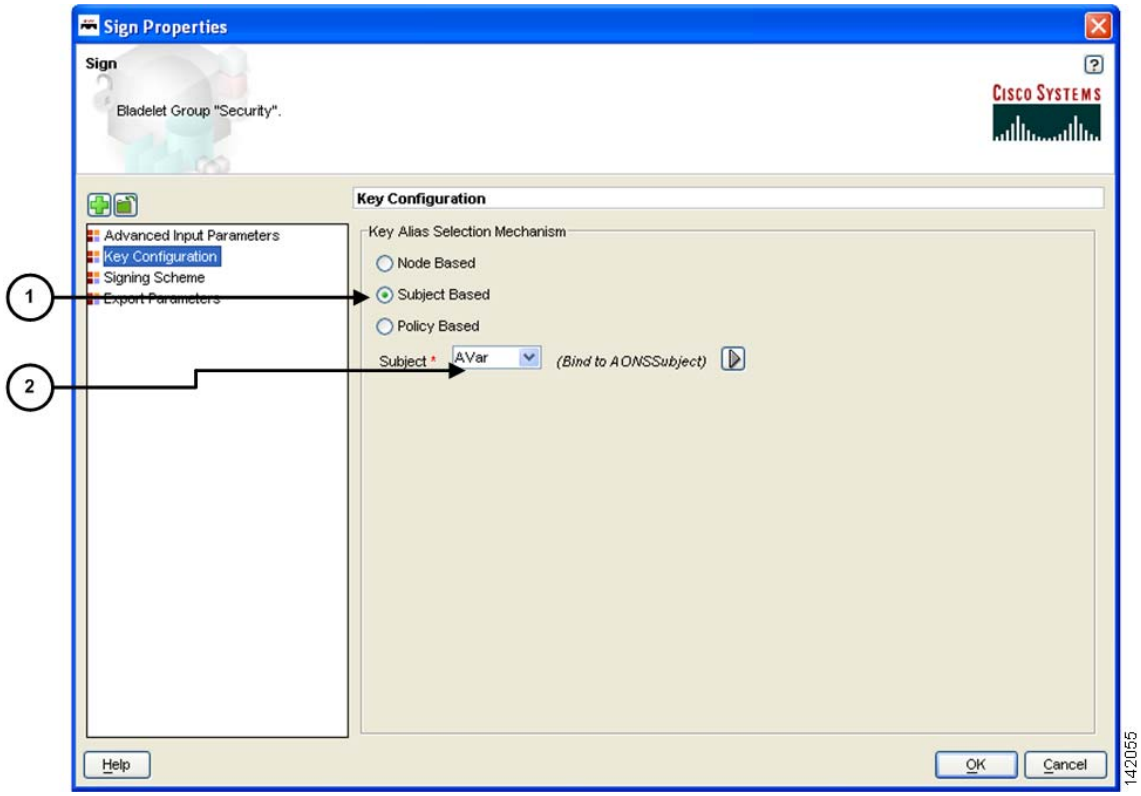
1	Digest Algorithm	Type of digest algorithm to be used to create digital signature.
2	Canonicalization Algorithm	Type of canonicalization algorithm to be used to create digital signature.

Figure 2-93 Sign Properties Window—Key Configuration, Node Based



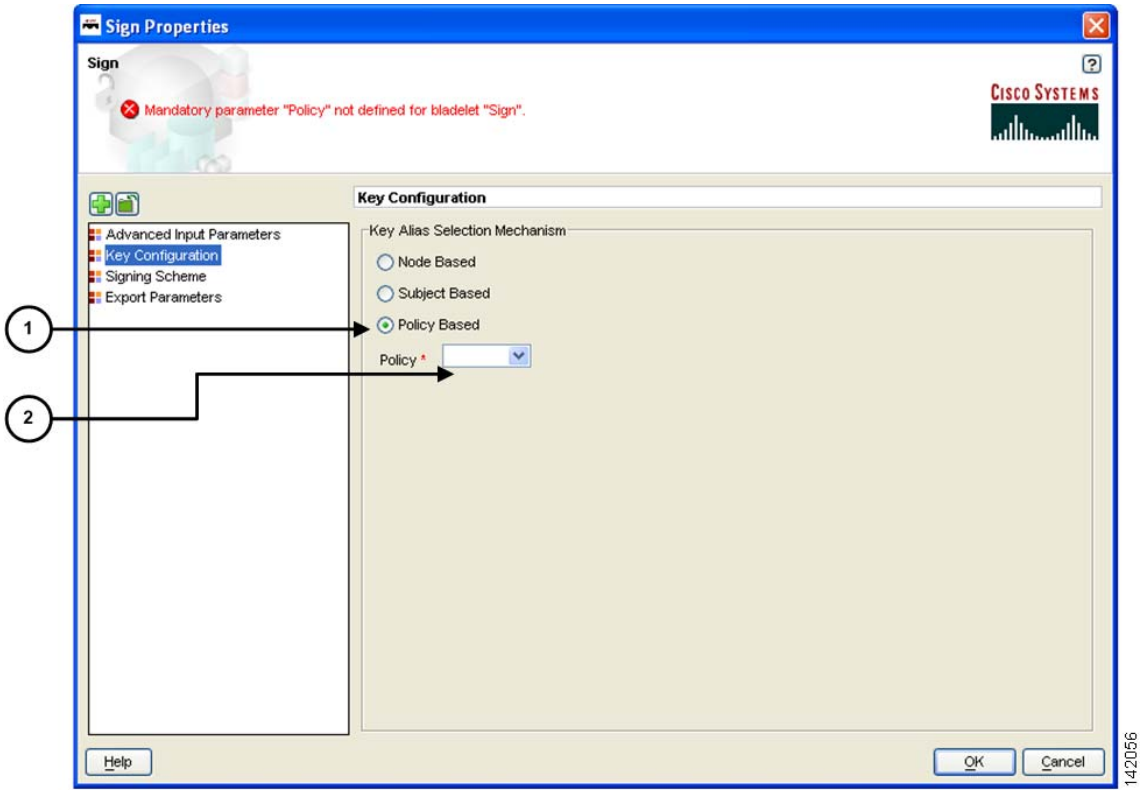
1	Node Based	Type of key configuration. Use node-based key alias instead of any signing policy. Uses the AON key for signing. Must already be configured in AMC-Keystore.

Figure 2-94 Sign Properties Window—Key Configuration, Subject Based



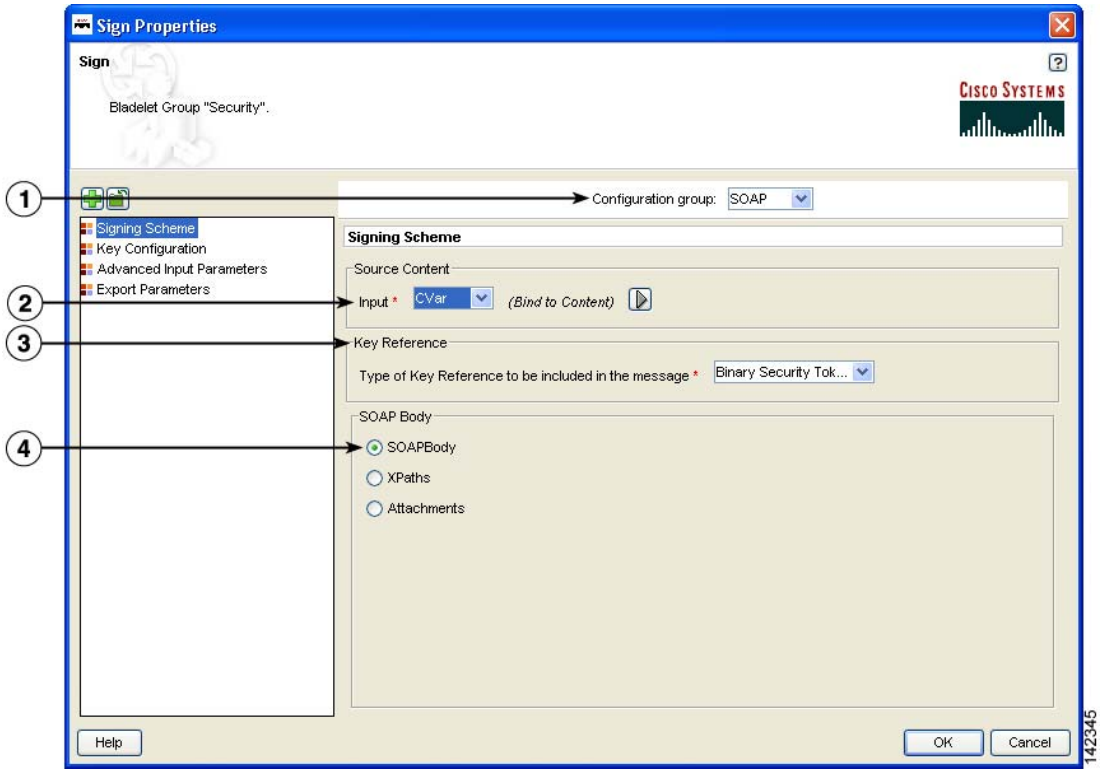
1	Subject Based	Type of key configuration. Key alias is extracted based on the value of the AONSSubject PEP variable.
2	Subject	Select subject such as the value AVar.

Figure 2-95 Sign Properties Window—Key Configuration, Policy Based



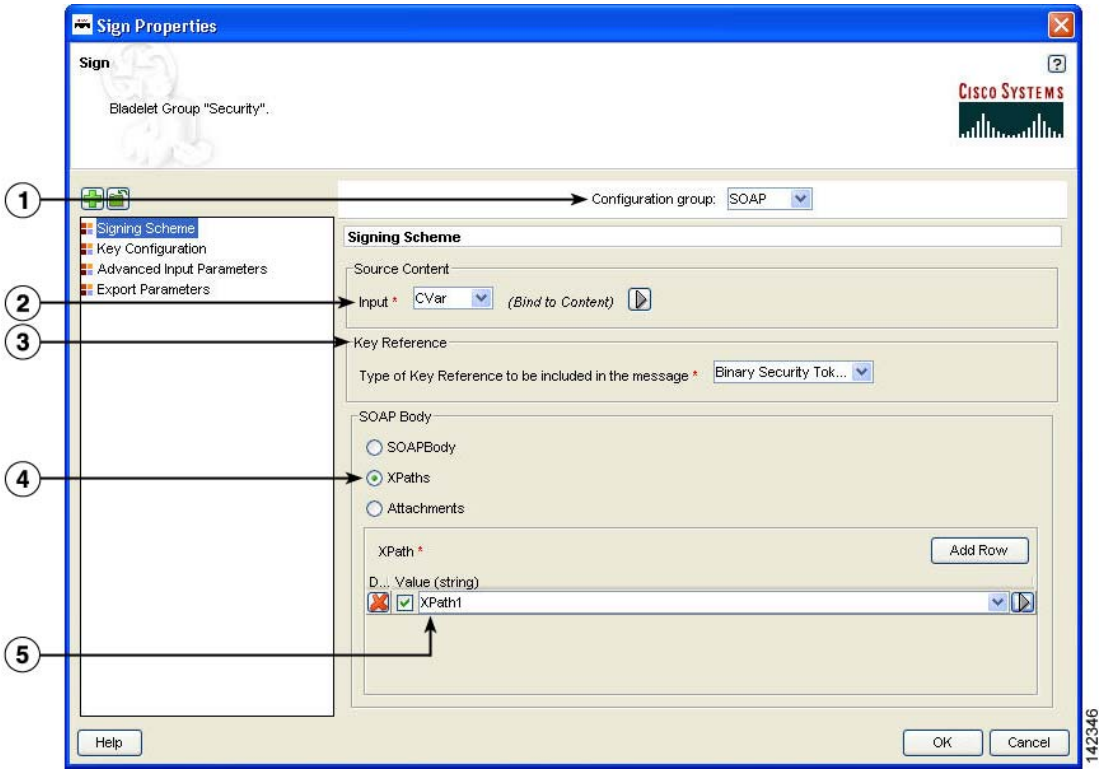
1	Policy Based	Type of key configuration. Signing policy containing configuration to the key alias.
2	Policy	Policy. Must already be configured on the AMC server.

Figure 2-96 Sign Properties Window—Signing Scheme, SOAP, SOAPBody



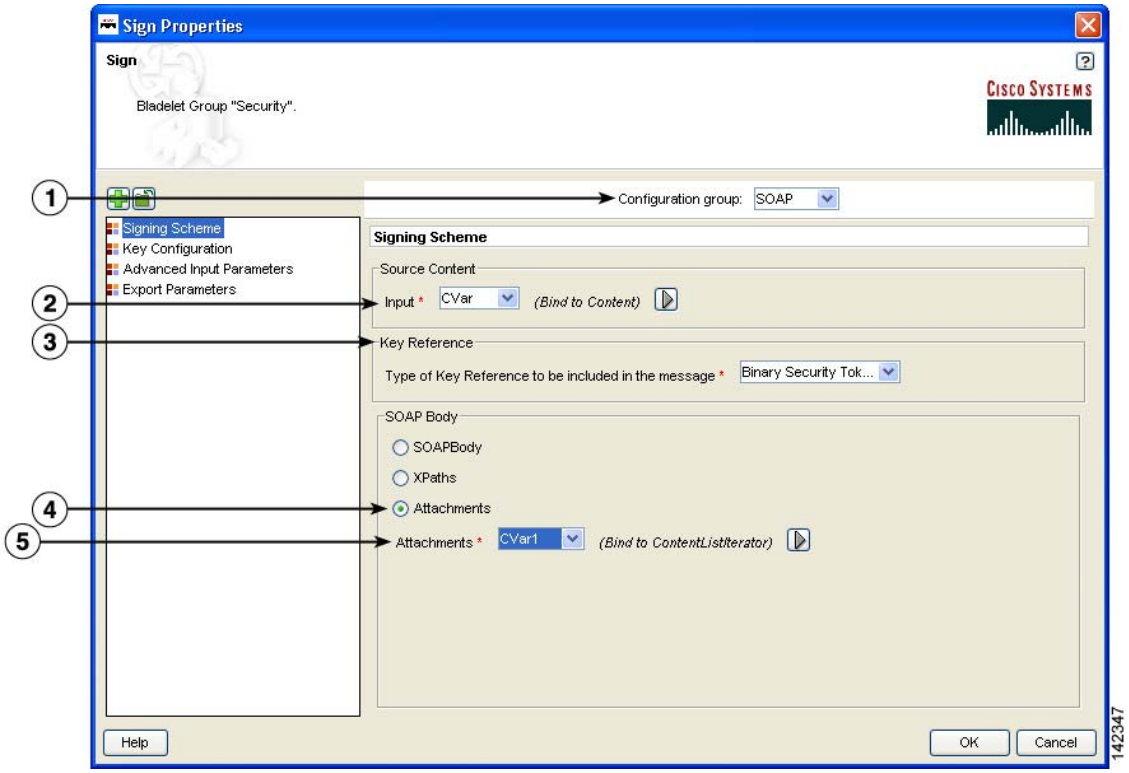
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	Source-content input. If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	SOAPBody	SOAPBody Signing Scheme. Whole soap body should be signed.

Figure 2-97 Sign Properties Window—Signing Scheme, SOAP, XPaths



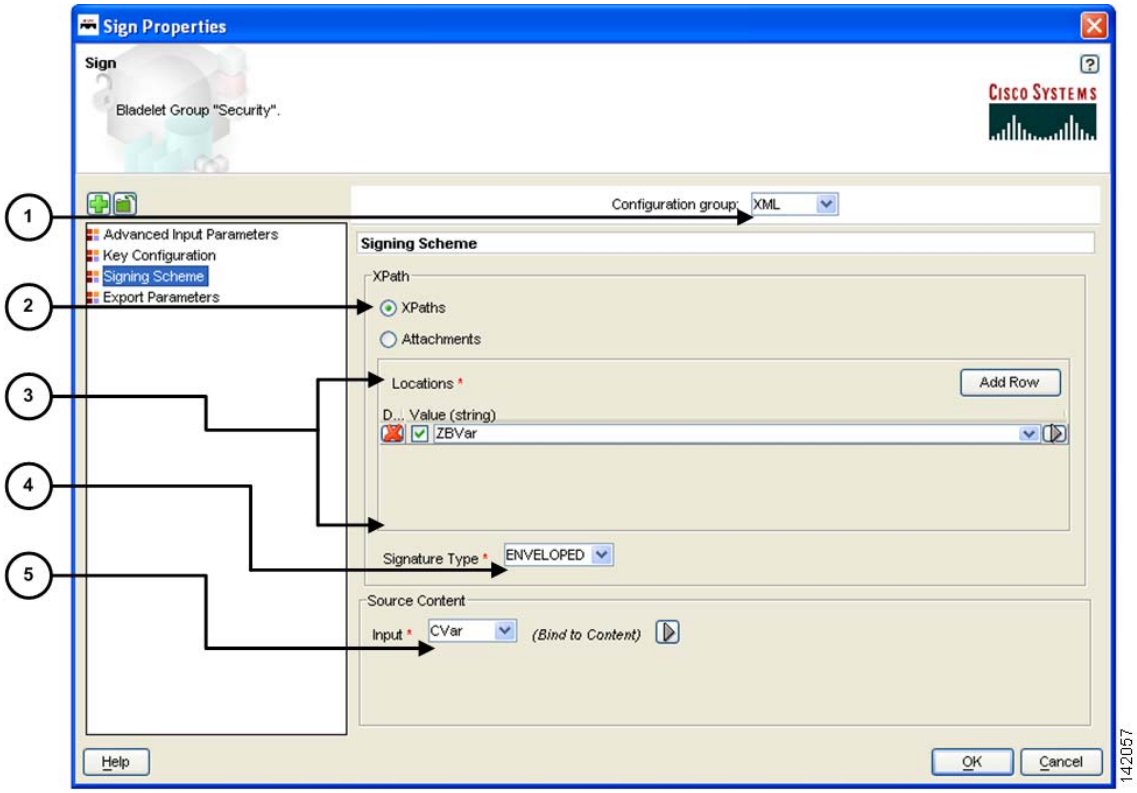
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	XPath Signing Scheme	List of Xpath expressions that are used to sign relevant portions on soap message.
5	XPath Value	XPath values (string form).

Figure 2-98 Sign Properties Window—Signing Scheme, SOAP, Attachments



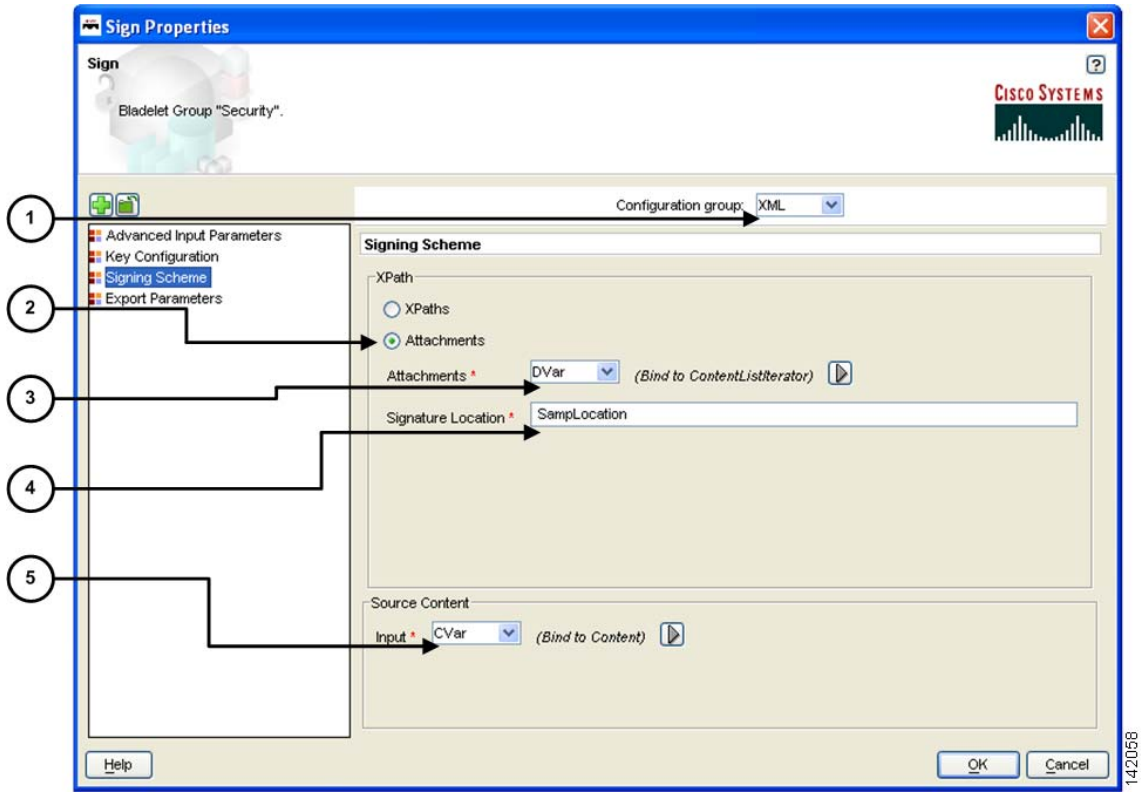
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	Attachments Signing Scheme	Attachments of multipart message, where root part is SOAP message, to be signed.
5	Attachments	Attachments of multipart message, where root part is SOAP message, to be signed.

Figure 2-99 Sign Properties Window—Signing Scheme, XML, XPaths



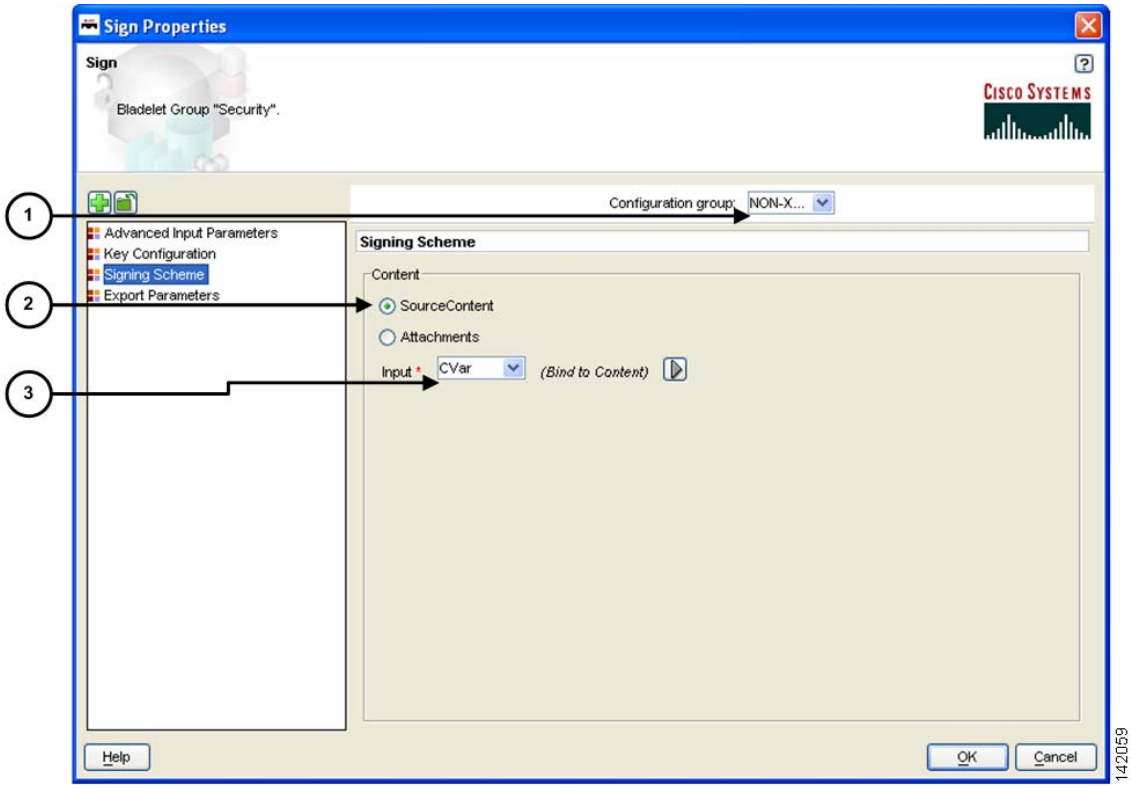
1	Configuration Group	Configuration group, set here to XML.
2	XPaths	List of Xpath expressions that are used to sign relevant portions on soap message
3	XPath Locations	One or more XPath locations (in string form).
4	Signature Type	Signature type: Enveloped or Enveloping.
5	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().

Figure 2-100 Sign Properties Window—Signing Scheme, XML, Attachments



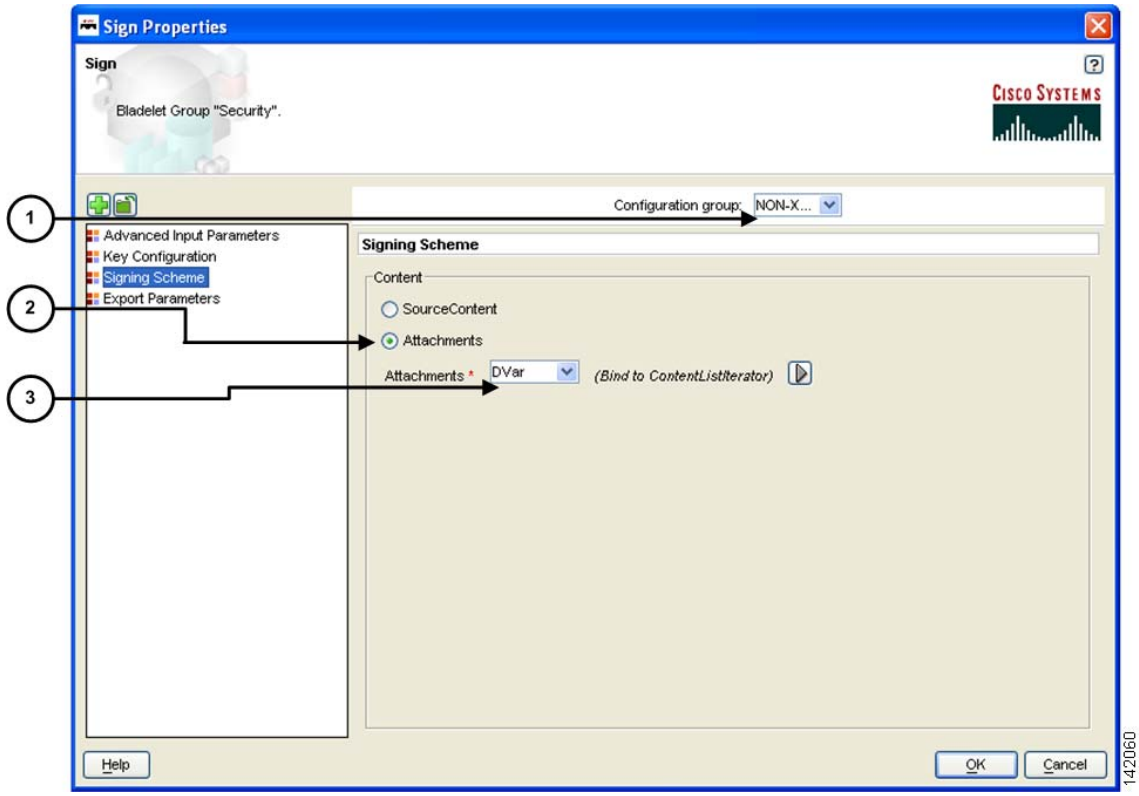
1	Configuration Group	Configuration group, set here to XML.
2	Attachments	Attachments of a multipart message, where root part is SOAP message, to be signed.
3	Attachments List	Selected attachments.
4	Signature Location	Location of the signature.
5	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().

Figure 2-101 Sign Properties Window—Signing Scheme, Non-XML, Source Content



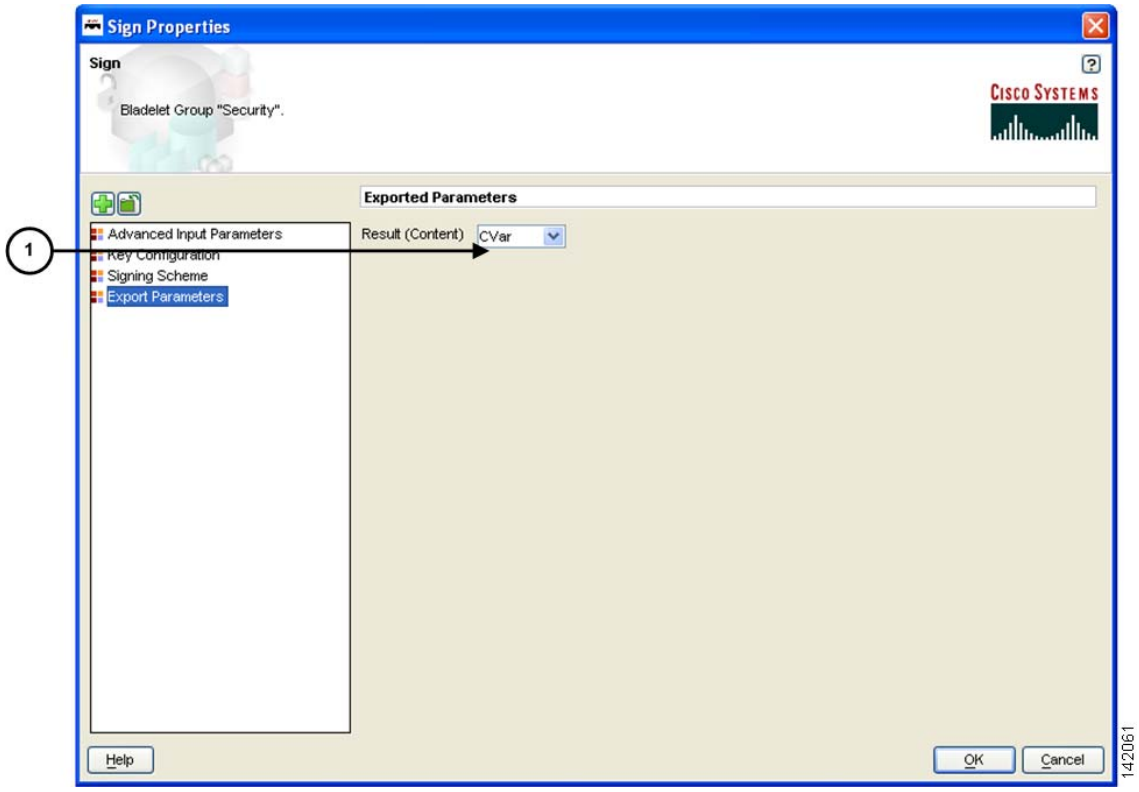
1	Configuration Group	Configuration group, set here to Non-XML.
2	Source Content	Base content of the original MIME message.
3	Input	Input content.

Figure 2-102 Sign Properties Window—Signing Scheme, Non-XML, Attachments



1	Configuration Group	Configuration group, set here to XML.
2	Attachments	Data structure that stores list of interested contents, which need to be digitally signed.
3	Attachments List	One or more attachments.

Figure 2-103 Sign Properties Window—Export Parameters



1	Result	Signed content. Usually use when the original message is a MIME message or non-XML message.

Outcome

- On successfully signing, requested messages that are not multipart messages contain digital signature information. For non-XML and multipart messages, export signed content of the Bladelet.

Exceptions

- Private Key Not Found: If the private key cannot be extracted from configured key alias.
- Data Not Found: No data is found to create the digital signature.

Decrypt

Summary

The Decrypt Bladelet decrypts encrypted XML, SOAP or non-XML messages as well as attachments.

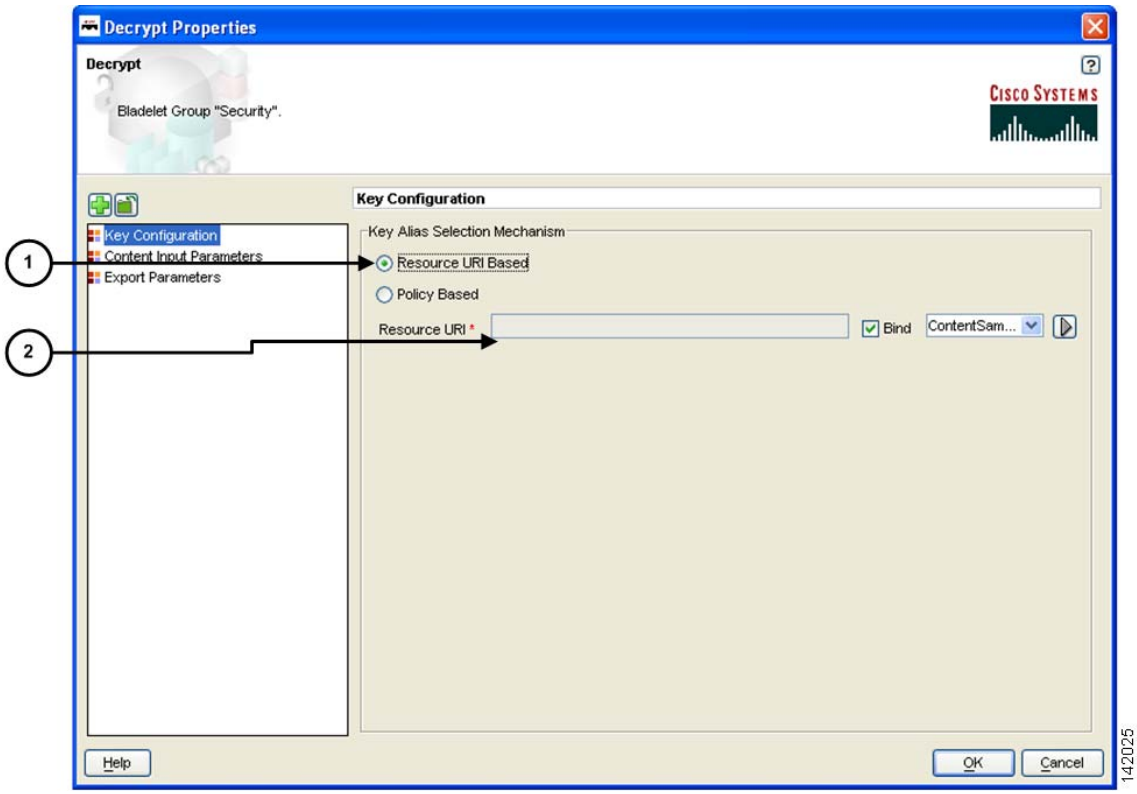
Prerequisites and Dependencies

- Configure decrypt policies and deploy them using the AMC server o send policies and keystores to AON.

Details

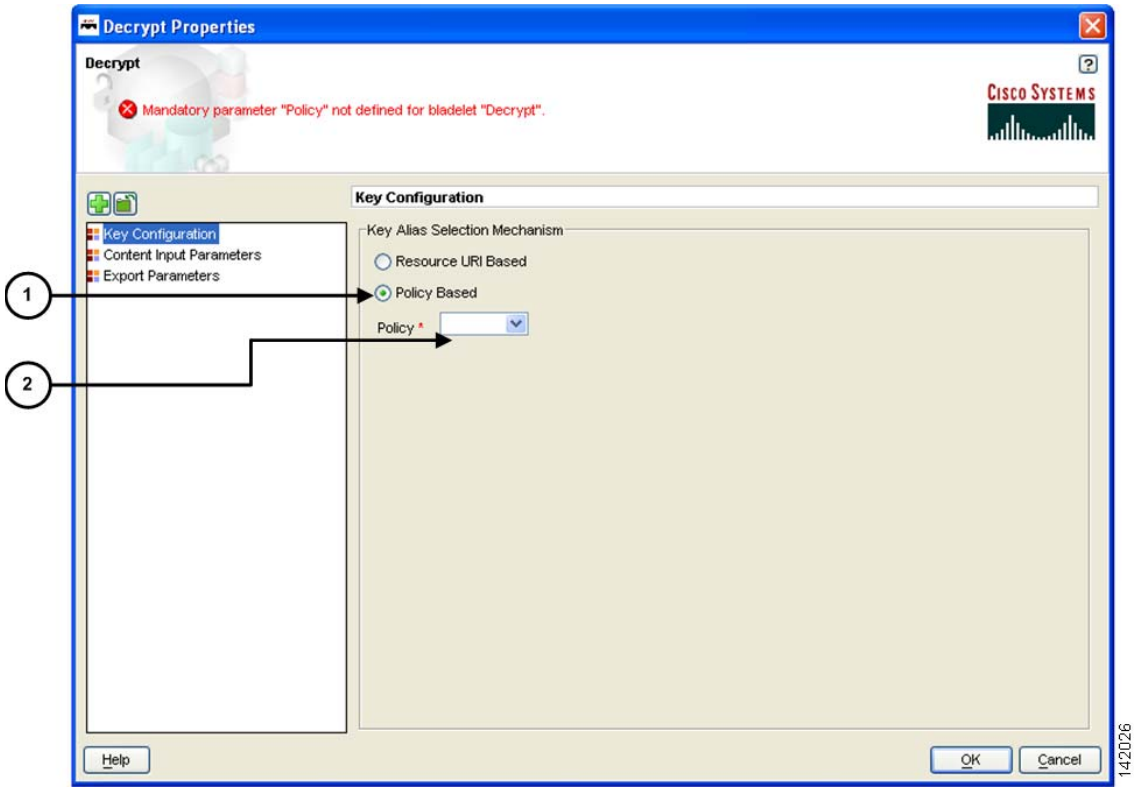
The Decrypt Bladelet decrypts SOAP messages containing data that has been encrypted with a symmetric key that has also been encrypted using an asymmetric public key. Given the private key of the recipient message as an input parameter, this Bladelet moves the CPU-intensive decryption operation to AON. Decrypt any or all of the encrypted data in a SOAP document by specifying the corresponding elements using XPath expressions. AON checks the destination URI of the message to determine the key alias for Decryption. For asymmetric key decryption, the Decryption key alias is identical to the destination hostname.

Figure 2-104 Decrypt Properties Window—Key Configuration, Resource URI Based



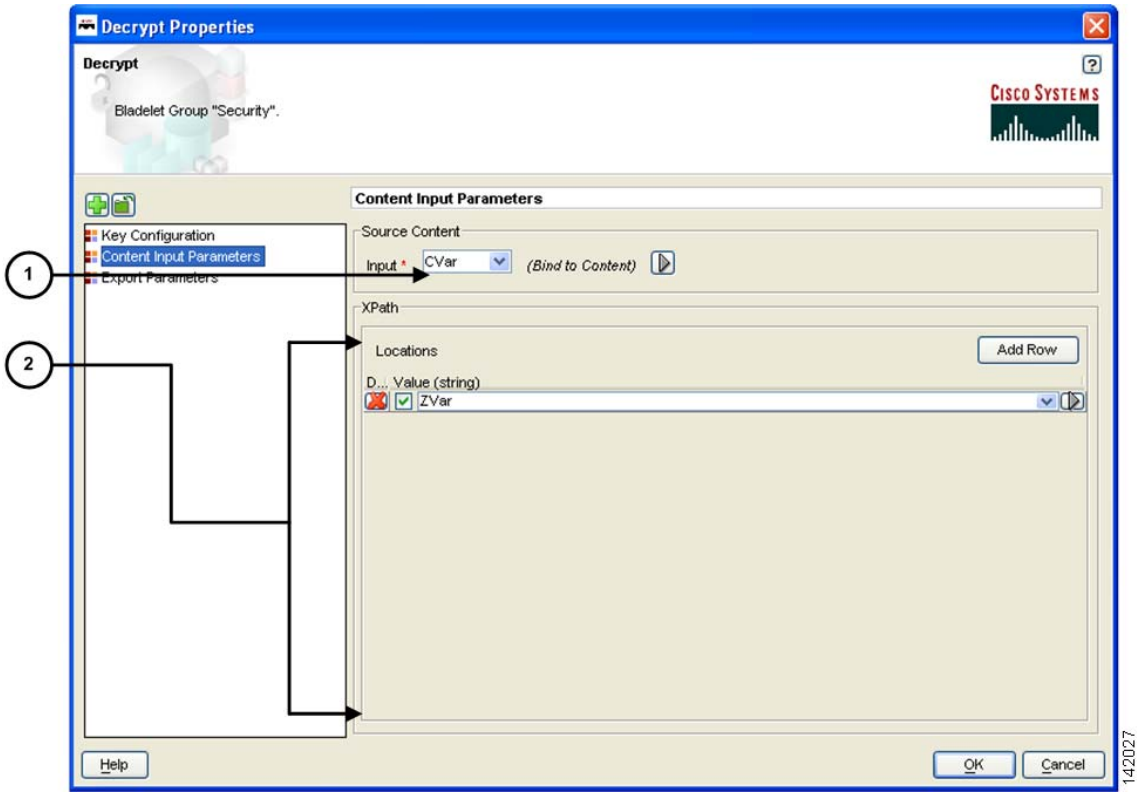
1	Resource URI Based	Resource URI Based is set as the key alias selection method.
2	Resource URI	URI of the intended recipient of this encrypted message. The key alias corresponding to this resource decrypts the symmetric key. Must already be configured on the AMC server.

Figure 2-105 Decrypt Properties Window—Key Configuration, Policy Based



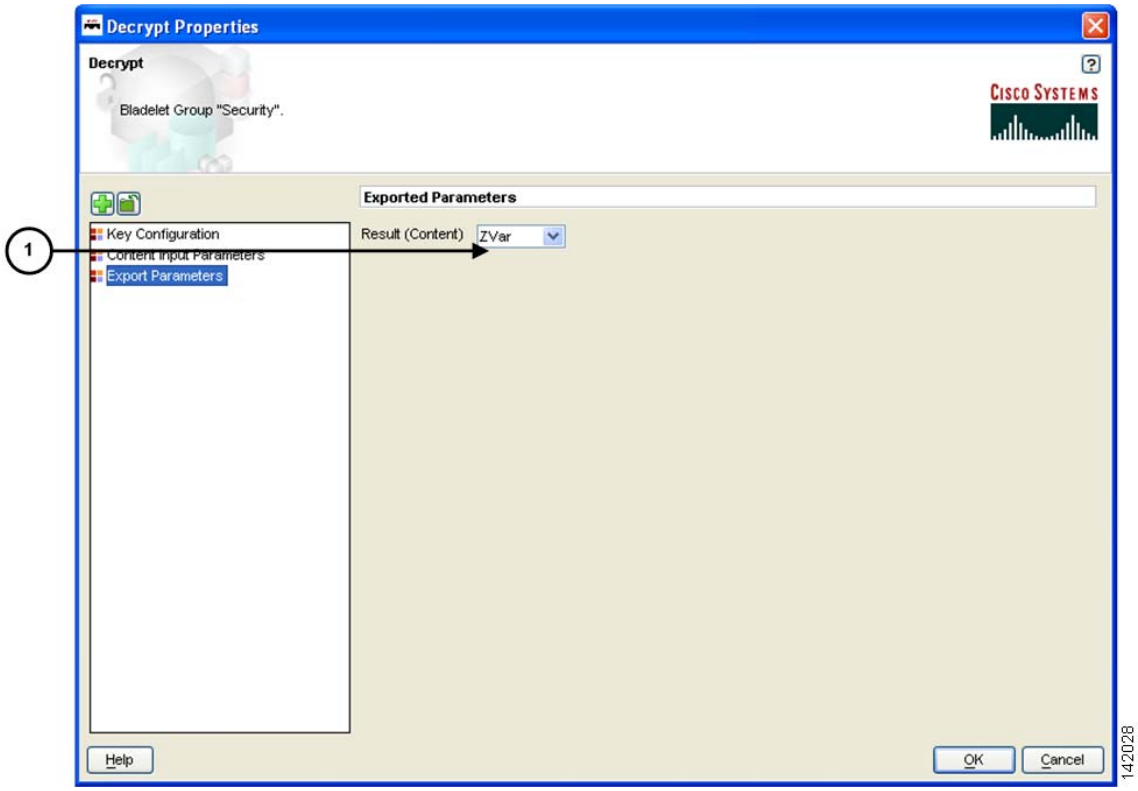
1	Policy Based	Policy Based is set as the key alias selection method.
2	Policy	Reference of the decryption policy. The key alias in this policy decrypts the symmetric key, irrespective of the resource URI that may be configured in this policy. Must already be configured on the AMC server.

Figure 2-106 Decrypt Properties Window—Content to Decrypt



1	Input	Input content that contains the encrypted data.
2	Xpath Locations	One or more XPaths for elements to be decrypted in the message. If blank, decrypts all encrypted data in the message.

Figure 2-107 Decrypt Properties Window—Export Parameters



1	Result	Output variable that contains the decrypted output of this Bladelet. Need not be set if the message being encrypted is of plain XML, SOAP or non-XML types (without attachments).

Outcome

- Success: Path taken if the Bladelet successfully decrypts the incoming message.
- Failure: Path taken if the Bladelet is unable to decrypt the message for any reason.

Exceptions

- Private Key Not Found: Path taken if the Bladelet is unable to retrieve the private key needed to decrypt the encrypted symmetric key from the message.
- Encrypted Data Not Found: Path taken if the Bladelet does not find any encrypted data in the message. Also if one or more XPaths are specified to decrypt, then this Exception is thrown if no encrypted elements are found at those XPath locations.

Identify

Summary

AON messages can use several types of claims or proof of identity. These items are generically referred to as “subjects.” This Bladelet can extract all subjects of specified types from the message being processed by the PEP.

Extract multiple types of identities at either the transport or message level, but not both.

Different types of identities are put into different sublists in SecurityContext and can be retrieved with different get functions.

As long as there is one identity extracted, the output path is “Success.” When no identity is extracted, the output path is “Failure.”

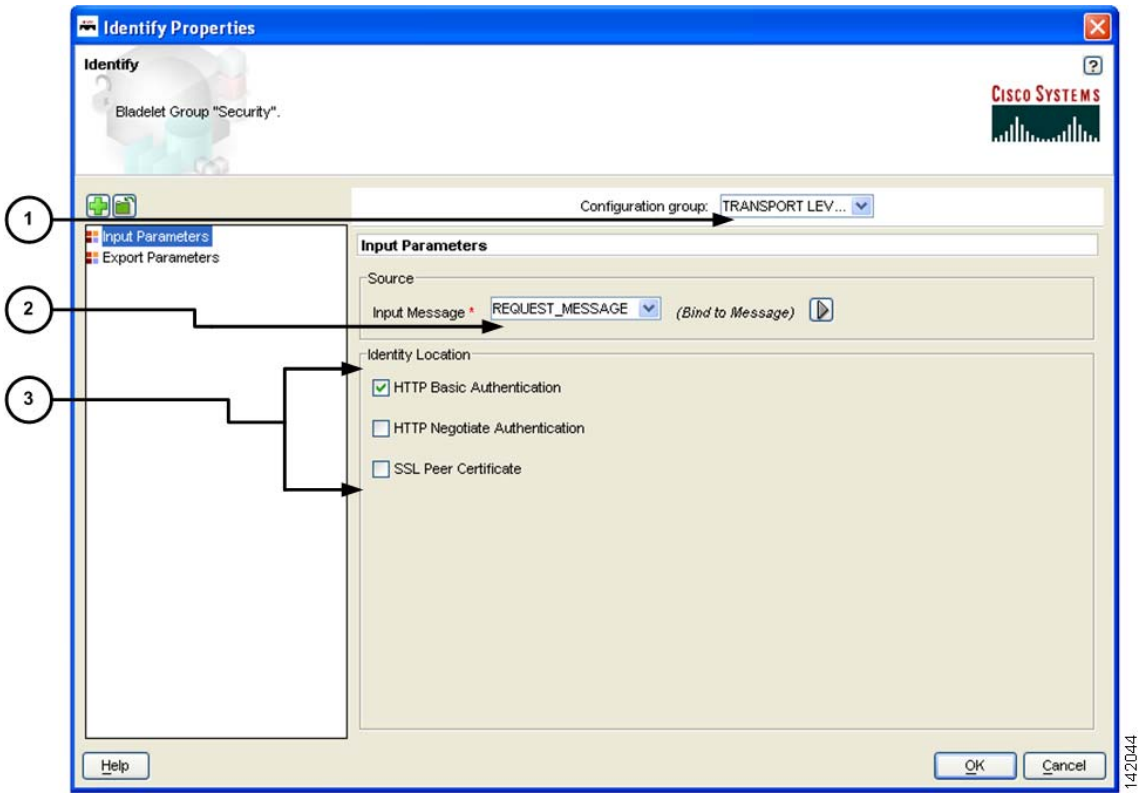
When there is no identity extracted, no HTTP-level challenge or soapfault is generated. Those message can be generated only by subsequent Bladelets that try to use the identity information for different purposes, such as authenticate and identity verify.

Prerequisites and Dependencies

None.

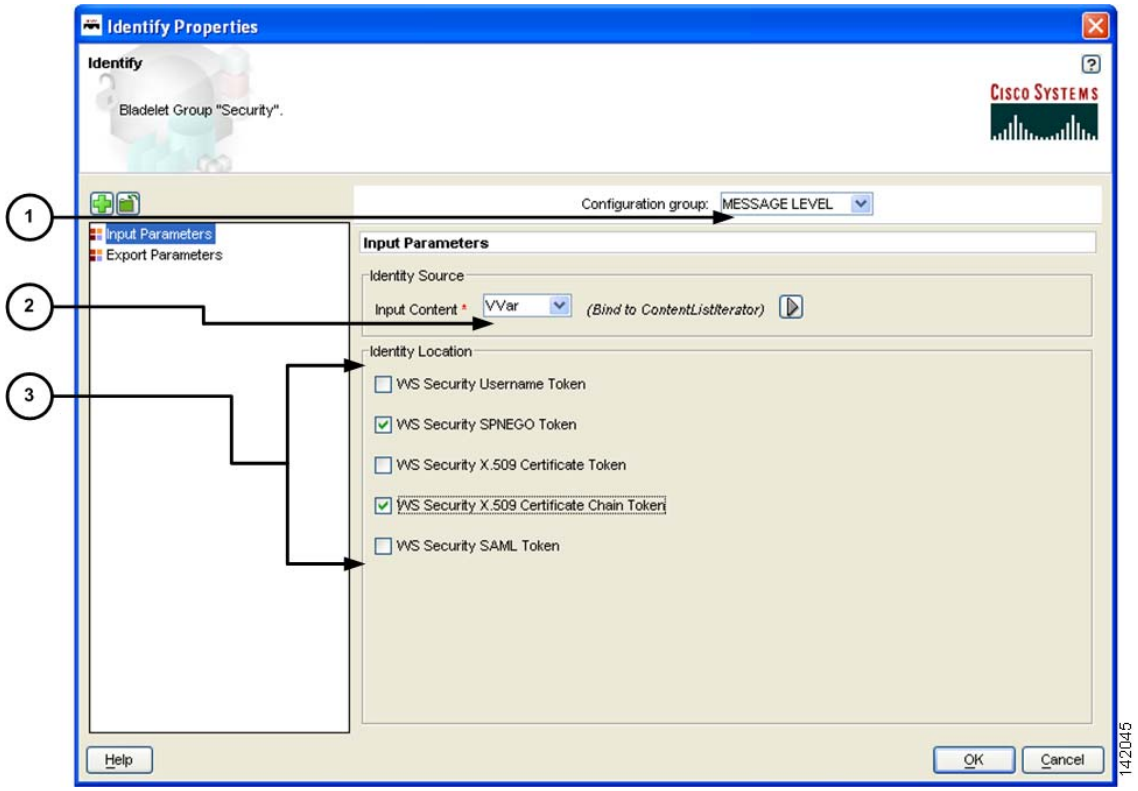
Details

Figure 2-108 Identify Properties Window—Input Parameters, Transport Level Identity



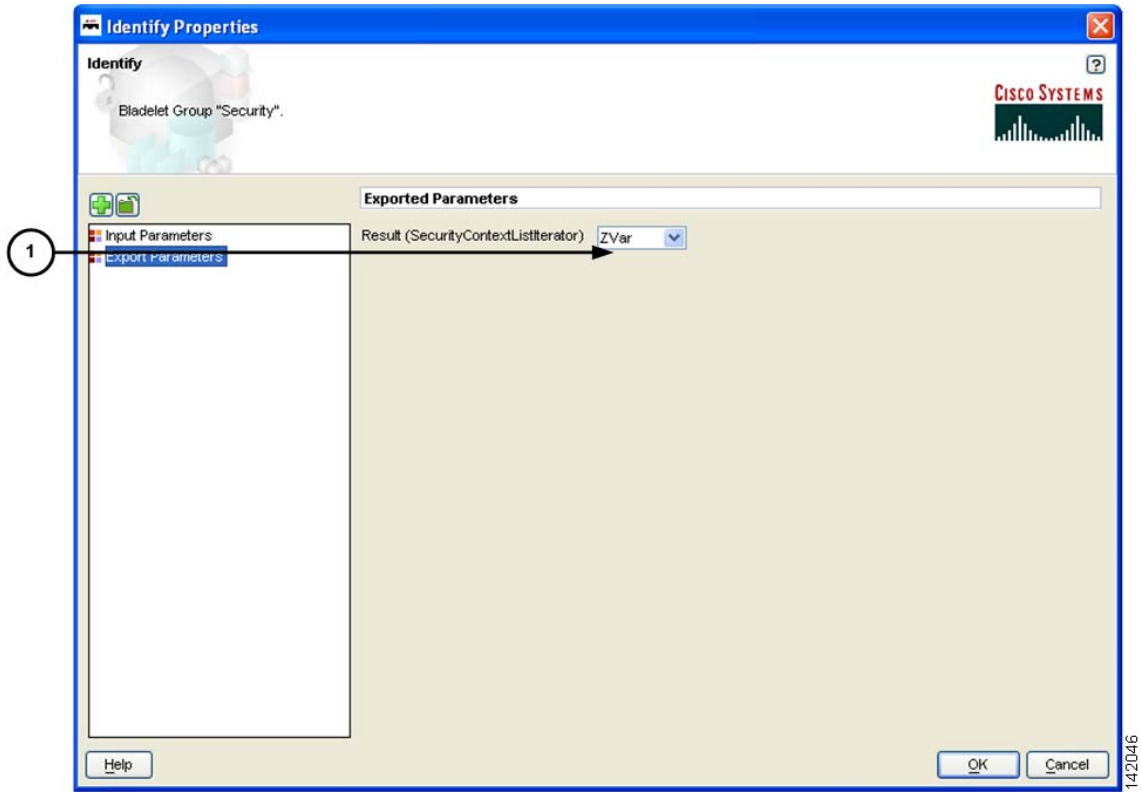
1	Configuration Group	Configuration group, set here to Transport Level.
2	Input Message	Incoming message to extract identity information from.
3	Identity Location	Location from which to extract HTTP information: • HTTP Basic Authentication—Extracts HTTP basic authentication information from incoming message. • HTTP Negotiate Authentication—Extracts HTTP negotiate authentication information from incoming message. • SSL Peer Certificate—Extracts SSL peer certificate from incoming message.

Figure 2-109 Identify Properties Window—Transport Layer Identity, Message Level Identity



1	Configuration Group	Configuration group, set here to Message Level.
2	Input Content	List of content to extract identity information from.
3	Identity Location	Location from which to extract security-token information: • WS Security Username Token—Extract WS-Security Username Token information from incoming contents. • WS Security SPNEGO Token—Extract WS-Security SPNEGO Token information from incoming contents. • WS Security X.509 Certificate Token—Extract WS-Security X.509 Certificate Token information from incoming contents. • WS Security X.509 Certificate Chain Token—Extract WS-Security X.509 Certificate Chain Token information from incoming contents. • WS Security SAML Token—Extract WS-Security SAML Token information from incoming contents.

Figure 2-110 Identify Properties Window—Export Parameters



1	Result	Data structure that stores the identity information extracted from the incoming message or contents. Allows the subsequent Bladelet to make use of the identity extraction results from the Identify Bladelet.

Outcome

- On success, a SecurityContextListIterator is populated with all the identity information extracted from incoming message.
- On failure, an empty SecurityContextListIterator is exported.

Exceptions

None.

Authenticate

Summary

The Authenticate Bladelet authenticates various credentials from the Identify Bladelet. An HTTP header or SOAP message are among the variety of sources that the Authenticate Bladelet can obtain identities from. You can set various property types for the Authenticate Bladelet.

Prerequisites and Dependencies

- The AONSSubjects to be authenticated are generated by Identify Bladelet. Ensure that Identify Bladelet precedes Authenticate Bladelet in a valid PEP and that the export parameter of Identify Bladelet retrieves the AONSSubjects.

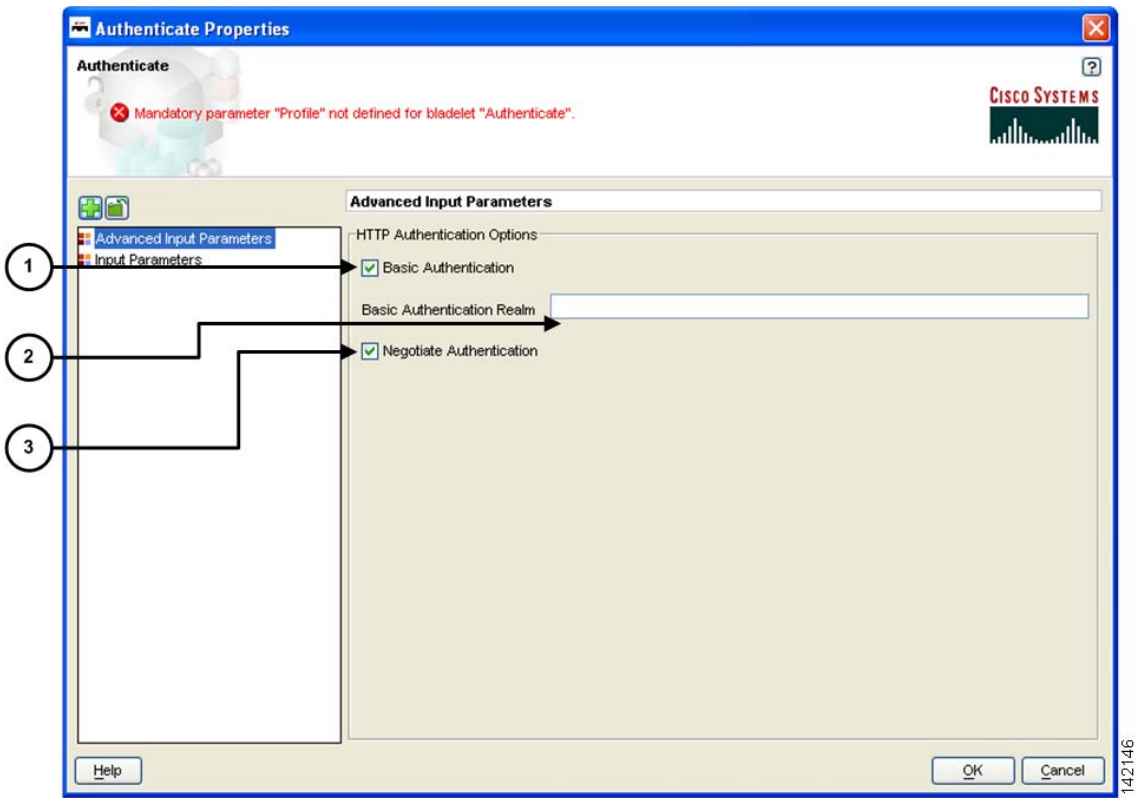
Details

An Authenticate Bladelet authenticates only one type of identity. To authenticate multiple types of identity, you must use multiple instances of Authenticate Bladelet in the PEP.

To perform HTTP-based authentication, put an Authenticate Bladelet on the “Failure” path of the Identify Bladelet used to extract the credential to generate proper HTTP authentication challenge.

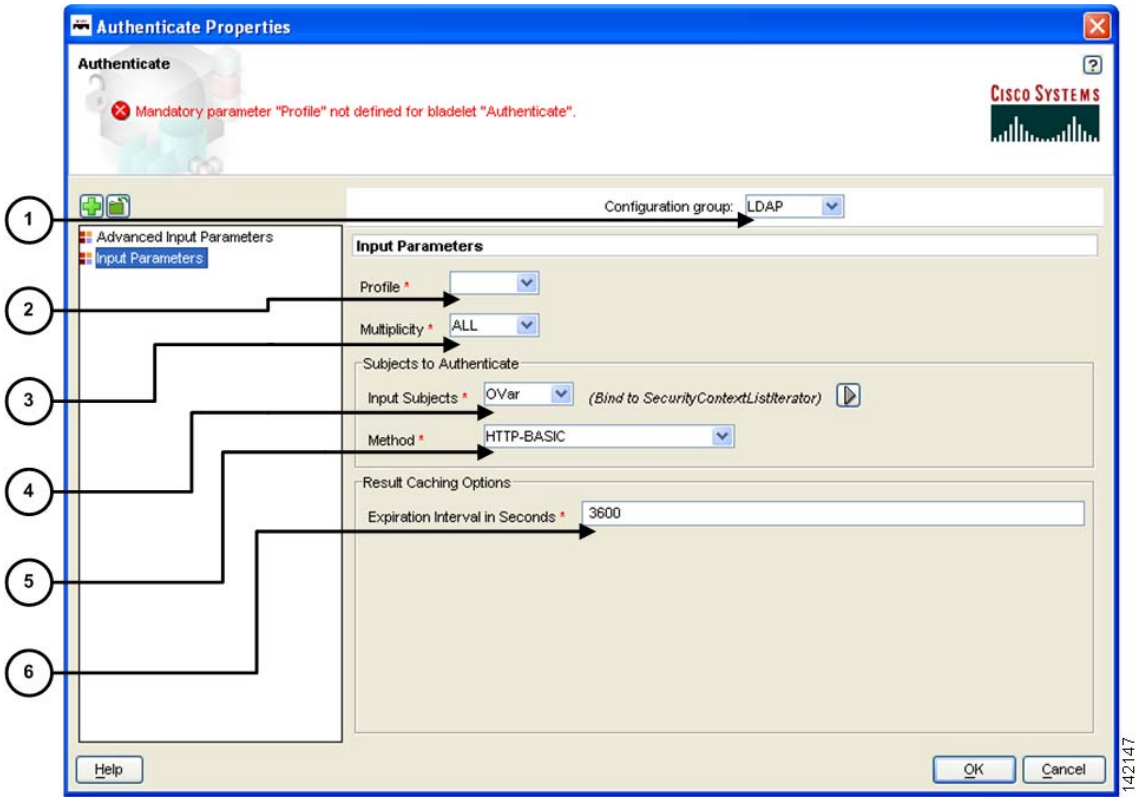
In order to generate HTTP response or proper soapfault message on exception cases, no Bladelet should be put on the exception path of the Authenticate Bladelet.

Figure 2-111 Authenticate Properties Window—Advanced Input Parameters



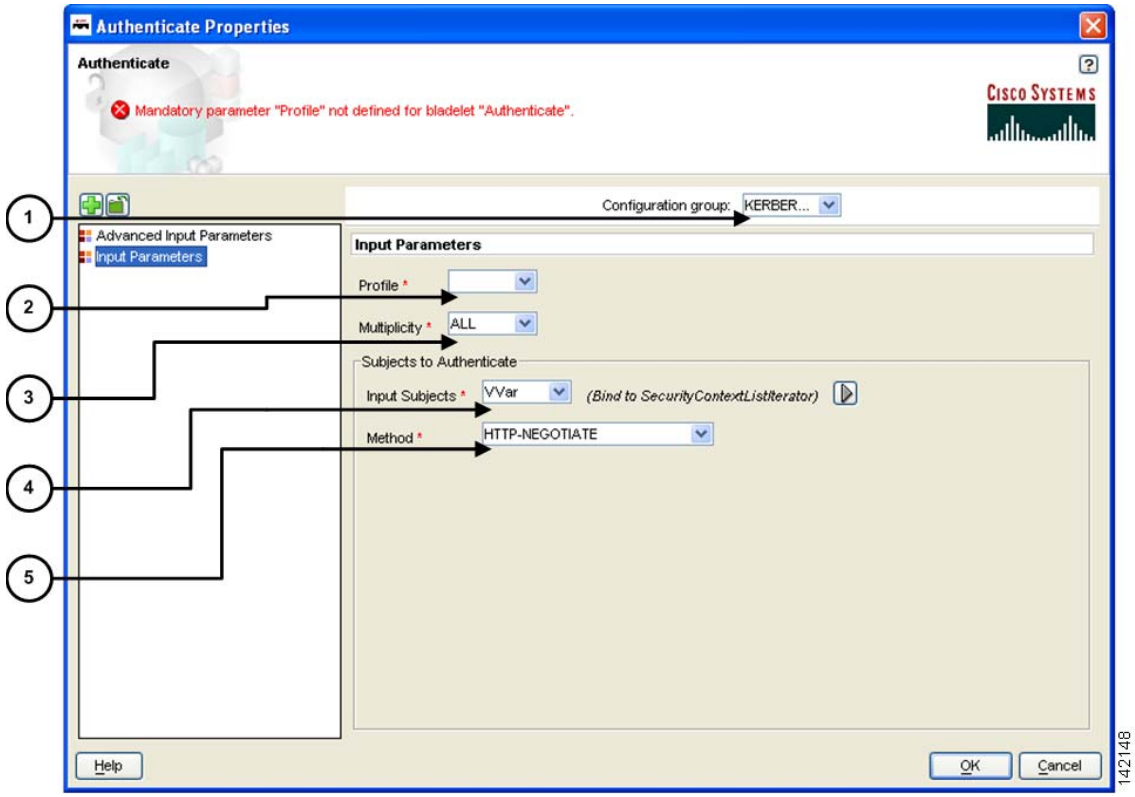
1	Basic Authentication	Whether or not this Bladelet supports HTTP basic authentication.
2	Basic Authentication Realm	Customized basic authentication realm. If nothing is defined, AON node hostname is used as default realm name.
3	Negotiate Authentication	Whether or not this Bladelet supports HTTP negotiate authentication.

Figure 2-112 Authenticate Properties Window—Authentication Scheme Configuration, LDAP



1	Configuration Group	Configuration group, set here to LDAP.
2	Profile	LDAP policy with configuration information for LDAP servers used to authenticate the subjects.
3	Multiplicity	Whether or not all or any subject in the list needs to be valid for the final success of the Bladelet.
4	Input Subjects	Data structure that stores the identity information to be authenticated. It should be exported by an Identity Bladelet.
5	Method	Type of the identity to be authenticated in this Bladelet.
6	Expiration Interval in Seconds	Time-to-live value for locally cached credentials.

Figure 2-113 Authenticate Properties Window—Input Parameters, Kerberos



1	Configuration Group	Configuration group, set here to Kerberos.
2	Profile	Kerberos policy with configuration information for KDC and Kerberos services used to authenticate the subjects.
3	Multiplicity	Whether or not all or any subject in the list needs to be valid for the final success of the Bladelet.
4	Input Subjects	Data structure storing the identity information to be authenticated. Should be exported by an Identity Bladelet.
5	Method	Type of the identity to be authenticated in this Bladelet.

Outcome

- On success, valid AONSSubject is marked as authenticated and can be retrieved through the following attributes of SecurityContext:
  - wssUsernameTokensAuthenticated
  - httpBasicAuthsAuthenticated
  - wssSPNEGOTokensAuthenticated
  - httpNegAuthsAuthenticated

Exceptions

- Credential Unavailable: No credential is available for the specified type in the source SecurityContextListIterator object.

- Communication Failure: Failed to communicate with the configured LDAP server or KDC.
- Credential Invalid: Authentication failed due to invalid credential.

Verify Identity

Summary

This Bladelet verifies whether the following types of identities are trusted by the AON node. The trust can be verified by CA root trust only or you can enforce that the certificate itself has to be present in the node's trust store.

Prerequisites and Dependencies

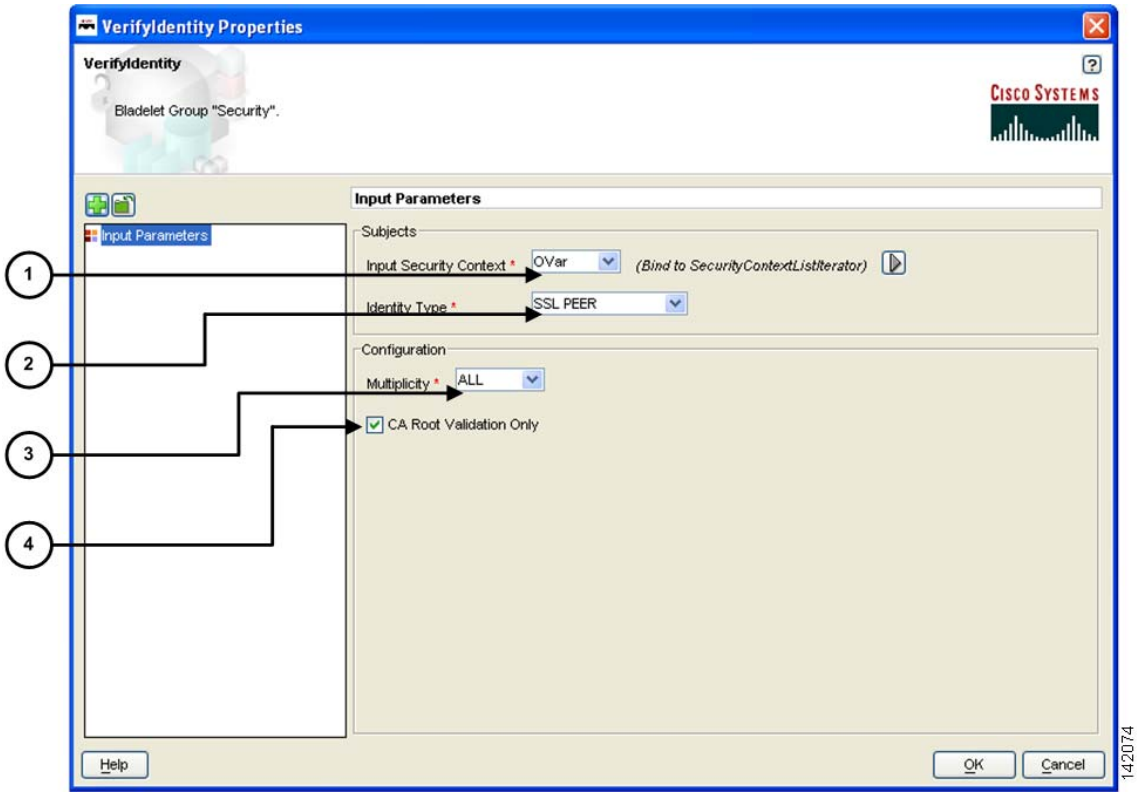
- The AONSSubjects to be verified are generated by Identify Bladelet. Ensure that Identify Bladelet precedes Verify Identity Bladelet in a valid PEP. Use the export parameter of Identify Bladelet to retrieve the AONSSubjects.
- Populate “trustedCACerts:” with trusted CA certificates. If the certificate itself has to be in the trust store to be considered trusted, populate the "trustedCerts" properly as well.

Details

A Verify Identity Bladelet can verify only one type of identity. To verify multiple types of credentials, multiple instances of Verify Identity Bladelets need to be used in the PEP.

In order to generate proper soapfault messages for exception cases, no Bladelet should be put on the exception path of the Verify Identity Bladelet.

Figure 2-114 Verify Identity Properties Window—Input Parameters



1	Input Security Context	Data structure that stores the identity information to be authenticated. Should be exported by an Identity Bladelet.
2	Identity Type	Type of the identity to be verified in this Bladelet.
3	Multiplicity	Whether all or any of the subject in the list needs to be valid for the final success of the Bladelet.
4	CA Root Validation Only	Whether the certificate needs to be trusted by one of the CAs in the CA trust store or to be present in the trust store of the node.

Outcome

- On success, valid AONSSubject is marked as verified and can be retrieved through the following attributes of SecurityContext:
  - wssX509CertTokensVerfied
  - wssX509CertPathTokensVerified
  - SAMLAssertionsVerified
  - SSLPeerCertsVerified

Exceptions

- Token Unavailable: No identity information is available for the specified type in the source SecurityContextListIterator object.
- Token Invalid: The identity is not trusted by the node.

Transformation Category

The Transformation Category has one Bladelet:

- Transform

Transform

Summary

This Bladelet performs transformation on AON Message Content. It can transform an XML message content to an XML or Non-XML content using XSLT Based Transformation mechanism. Further, Non-XML message content can also be transformed to XML or Non-XML message content by providing a content parser extension.

If the message is not a multipart message, then its contents can be transformed and result of the transformation can be placed in the specified message and additionally can be exported as a PEP variable.

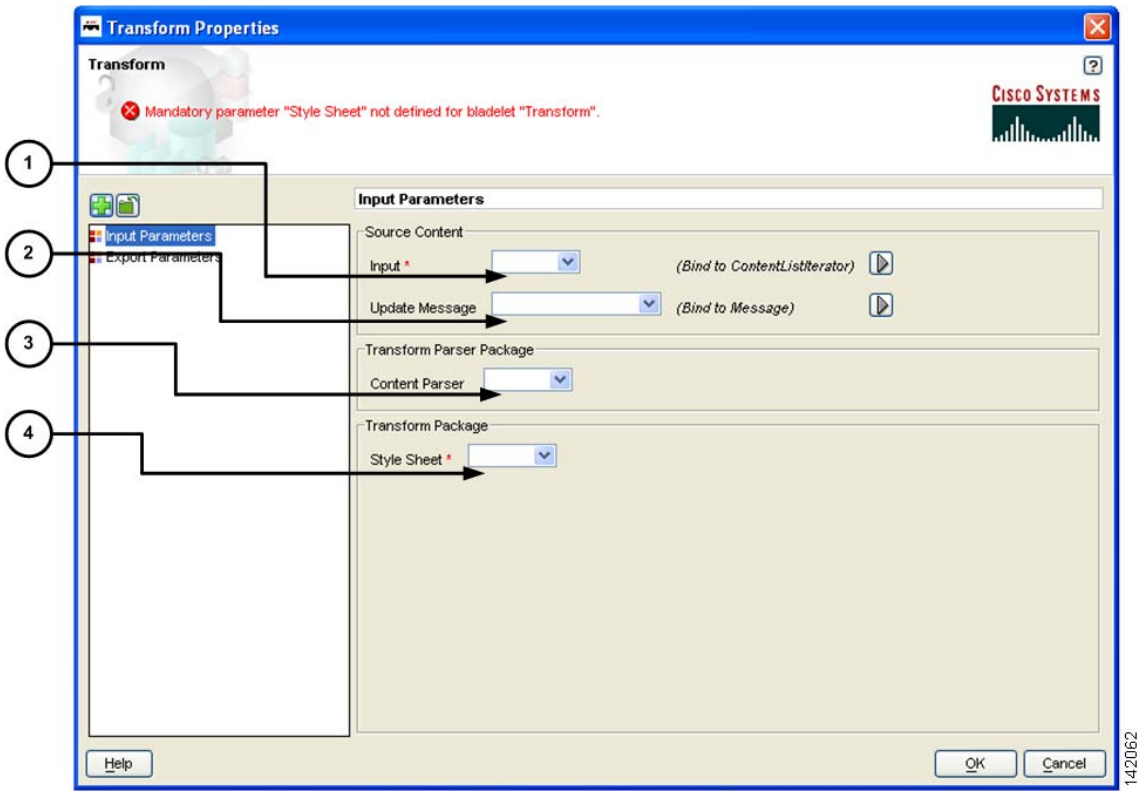
If the message is a multipart message and the list of contents are transformed in to a list of result contents, you must use BuildCompositeMessage to build a result multipart message.

Prerequisites and Dependencies

- Define a Transform Property Set value from the AMC server.
- Transform property set specifies a Style Sheet to use in transformation. In the Transform Property, specify the name of the style sheet and the package in which it is provided. Transform packages are created using ADS and loaded and registered in AMC. Deploy the transform package on a node before using the style sheet in transformation in PEP on the node.
- For using Content Parser property set in the Bladelet, define Content Parser property set from the AMC server.
- If the Content Parser property set so defined uses Parser Plug-in and Transformer Plug-in classes, design these classes and provide them in a Content Parser package in ADS. Load the package and register it with the AMC server.

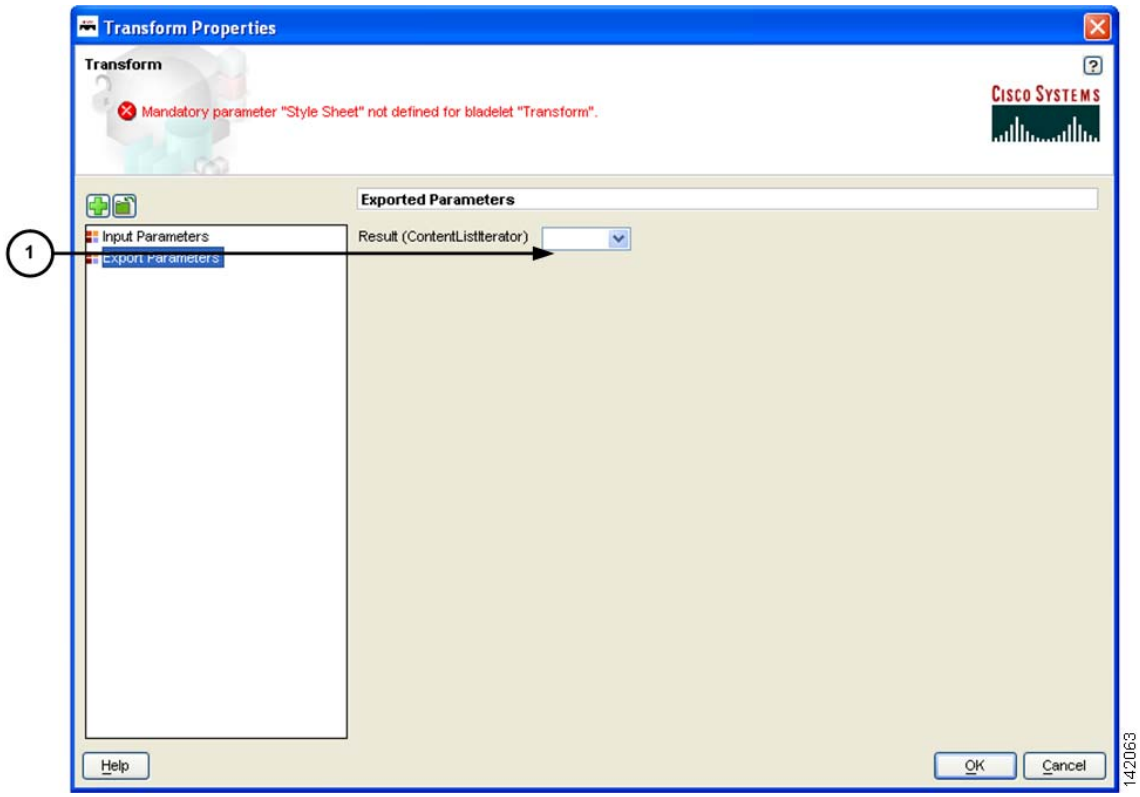
Details

Figure 2-115 Transform Properties Window—Input Parameters



1	Input	List of contents to transform. Content can come from either of the following: • It can be extracted and provided in a list by calling method content->iterator() on the Message PEP variable. • It can come from the results of a ExtractCompositeMessage Bladelet.
2	Update Message	Message in which the transformation result is placed. If input contains multiple contents, you must export the result of transformation in a Result. Use BuildCompositeMessage Bladelet to build a multipart message. Additionally or alternately, you can export the result in a PEP variable selected under the Export Parameter section in the Result field.
3	Content Parser	Content Parser property set. Defines parser plug-in and transformer plug-in classes to use if specified. Must already be created in the AMC server.
4	Style Sheet	Transform property set. Defines name of the style sheet to use for transformation. Must already be created in the AMC server.

Figure 2-116 Transform Properties Window—Export Parameters



1	Result	List that contains transformed contents. After the results of the transformation are placed in a PEP variable, they can be used in subsequent transformation or can be used to build a mulitpart message using BuildCompositeMessage Bladelet.

Outcome

- Transform Bladelet performs the transformation of message content based on the Style Sheet property and Content Parser policy. If transformation is successful, transformed content can be updated in the message selected in Updated Message field. If transformation is operating on a list of contents, the result of transformation must be exported in Result parameter.

  If transformation is successful, Success output path is set. In case of failure, Fail output path is set.

Exceptions

None.

Miscellaneous Category

The Miscellaneous category contains no Bladelets.

CHAPTER 3

ADS PEP Attributes Reference

This chapter presents detailed reference information that you need to use Cisco AON Development Studio (ADS) to assign Policy Execution Plan (PEP) attributes.

Note For more information on implementing an AON network, see the following:

- Other chapters in this guide:
  - Chapter 1, "Getting Started with Cisco ADS"
  - Chapter 2, "ADS Bladelets Reference"
  - Chapter 4, "ADS Message Types Reference"
- Other guides in the AON library:
  - *AON Installation and Administration Guide* (for information on the AMC server and nodes)
  - *AON Programming Guide* (for information on custom Bladelets, custom adapters, and application program interfaces)

Contents

Information About PEP Attributes

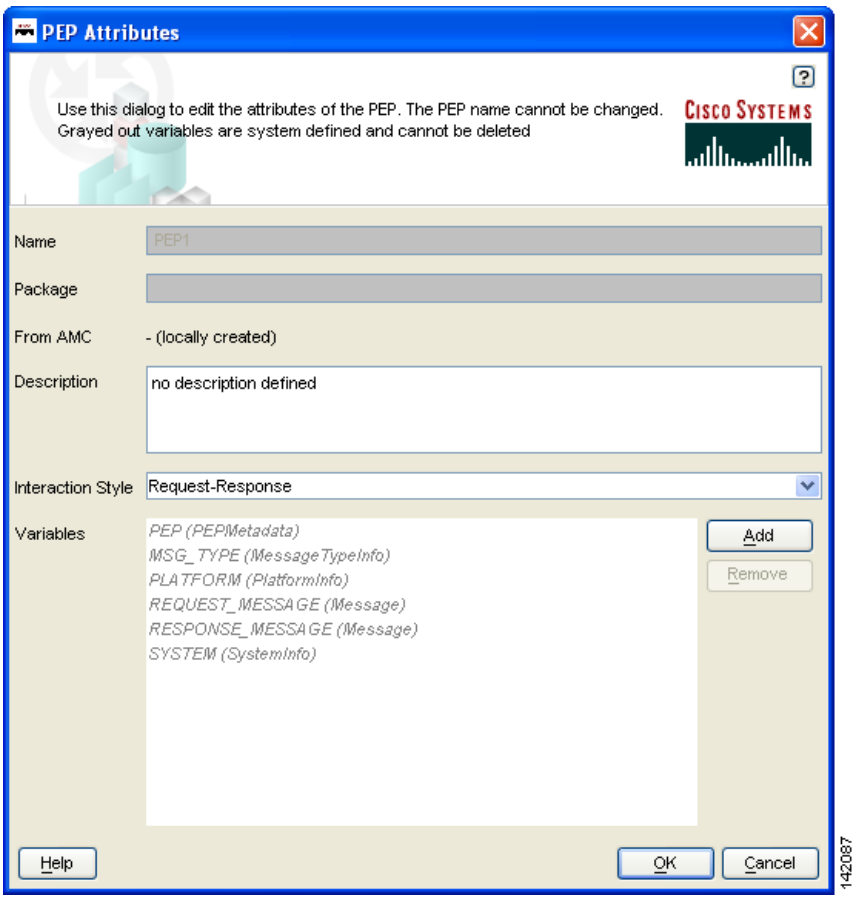
PEP attributes are items such as name, package, description, interaction style, and operating variables that apply to the entire collection of Bladelets and paths that constitute a PEP.

PEP Attribute Window and Dialog Boxes

Common tasks involving creating PEPs are discussed in Chapter 1, "Getting Started with Cisco ADS." This section describes how to assign PEP attributes, when you start to create a PEP or at any later time.

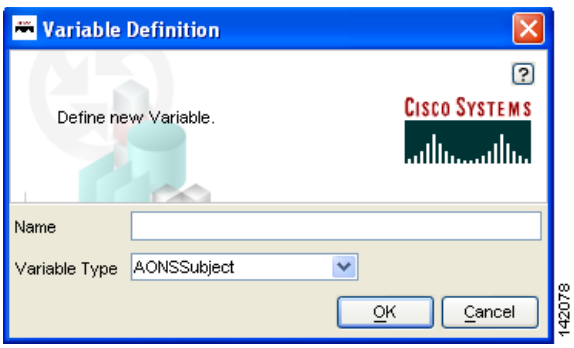
You assign PEP attributes by means of the PEP Attributes window (Figure 3-1) and subsequent dialog boxes. (To open this window, follow the procedure in the "Creating PEPs" section on page 1-9.)

Figure 3-1 PEP Attributes Window



Clicking **Add** opens the Variable Definition dialog box (Figure 3-2).

Figure 3-2 Variable Definition Dialog Box



Provide a name for the variable. From the Variable Type drop-down list, select a variable type.

Note For a list of variable types, see Table 3-1 on page 3-3.

PEP Attribute Variable-Type Choices

PEP attributes are listed in Table 3-1.

Table 3-1 Variable Definition: Available Types

Type	Description
AONSSubject	Subject of the AON message.
AONSSubjectListIterator	Iterator for a list of AONSubject objects.
Content	Content of the AON. An object of this type is created by the CreateContent Bladelet and is consumed by the CreateMessage Bladelet. You do not create this directly; rather, it is created by the CreateContent Bladelet. The message type has a content attribute that returns the content of the message (example: REQUEST_MESSAGE.content() gives the content of the incoming message). Content has the following attributes: • numParts—If the underlying content is a MIME content, 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(). • document—If the underlying content is of the XML type, returns the Document object corresponding to the actual content. The document representation of the content in the incoming message is given by REQUEST_MESSAGE.content().document().
ContentListIterator	List of content values (see Content for details) that can be accessed one at a time.
Document	DOM document. An object of this type can be extracted from the Content object (see Content for details) if the content is XML. You do not create this directly.
FindContentListIterator	Iterator for content list search results (see Content for details).
FindResult	Collection of search results for one Xpath. You do not create this directly; rather, it is generated by the Find Bladelet. It has the following attributes: • String value—For a single node, returns the string value of the node; for a list of nodes, returns the string value of the first node. • String node Value(*i*)—Returns the string value of the *i*th node • int size—Returns the size of the result set.

Table 3-1 Variable Definition: Available Types (continued)

Type	Description
FindResultMapIterator	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 that Xpath. This is generated by the Find Bladelet. It has the following attributes: • FindResult first • FindResult last • Findresult elementAt
FindResultMapListIterator	Iterator for a map list of Xpath/Regex search results (see above).
Message	AON message. The PEP variable REQUEST_MESSAGE of this type is available in the request-action and represents the incoming message. The PEP variable REQUEST_MESSAGE of this type is available in the response-action and represents the outgoing message. The CreateMessage Bladelet can create an object of this type in the PEP. You do not create this directly. It has the following attributes: • messageId—Returns the id of the message. The id of an incoming message is given by: REQUEST_MESSAGE.messageId() • timeStamp—Time at which the message was created. The timestamp of the incoming message is given by REQUEST_MESSAGE.timeStamp() • srcIP—IP address of the message source. The source IP of the incoming message is given by REQUEST_MESSAGE.srcIP() • srcPort—Port number of the message source. The source port of the incoming message is given by REQUEST_MESSAGE.srcPort() • destIP—IP address of the message destination. The destination IP of the incoming message is given by REQUEST_MESSAGE.destIP() • destPort—Port number of the message destination. The destination port of the incoming message is given by REQUEST_MESSAGE.destPort() • destProtocol—String representation of the message protocol. The protocol name of the incoming message is given by REQUEST_MESSAGE.destProtocol() • header—Value of the header in the message. The User-Agent header of the incoming message is given by REQUEST_MESSAGE.header(User-Agent) • content—AON Content of the message. The content of the incoming message is given by REQUEST_MESSAGE.content() • URI—Destination uniform resource identifier (URI) of the message. The URI of the incoming message is given by REQUEST_MESSAGE.URI()

Table 3-1 Variable Definition: Available Types (continued)

Type	Description
SearchResult	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.
SearchResultListIterator	Iterator over a list of SearchResult objects.
SecurityContext	Store of subject and credential information for certain message or content.
SecurityContextListIterator	Iterator for a list of SecurityContext objects.
Boolean	Value of either true or false.
byte	Value that you can input directly. Range: –128 to 127.
double	Value that you can input directly. Range: 4.9E–324 to 1.7976931348623157E308.
float	Value that you can input directly. Range: 1.401298464324817E–45 to 3.4028234663852886E38.
int	Value that you can input directly. AON uses this in its Bladelets. Range: –2147483648 to 2147483647.
iterator	Keeps the state of a position in a list. It can be used for accessing items from the data structure and can be viewed as a list, array, or stream. It is able to do so one at a time.
list	List or collection of objects. The element of a list can be any object (such as a string). It has the following attributes: • Size—Returns the size of the list. For example, if a PEP variable "I" of type list is defined in the PEP, I.size() gives the number of elements in the list. To retrieve the *n*th element in a list, specify list[*n*]. For example, to get the first element in a list, specify list[0]
long	Value that you can input directly. Range: –9223372036854775808 to 9223372036854775807.
map	Collection of name-value entries. AON supports maps of string-typed keys to string-typed values. It has the following values: • Size—Returns the size of the map. For example, if a PEP variable "m" of type map is defined in the PEP, m.size() gives the number of entries in the map. To retrieve the value in a map corresponding to key "K", use map{K}. In case of a list of maps, to access the value corresponding to key "K" in the first map in the list, use list [0]{K}.
object	Any object. You do not provide this directly. If input to a Bladelet is of this type, any PEP variable can be bound to the input.
short	Value that you can input directly. Range: –32768 to 32767.
string	String of characters. Type the string in the ADS text box or text area.

CHAPTER 4

ADS Message Types Reference

This chapter presents detailed reference information that you need to use Cisco AON Development Studio (ADS) to assign message types.

Note For more information on implementing an AON network, see the following:

- Other chapters in this guide:
  - Chapter 1, "Getting Started with Cisco ADS"
  - Chapter 2, "ADS Bladelets Reference"
  - Chapter 3, "ADS PEP Attributes Reference"
- Other guides in the AON library
  - *AON Installation and Administration Guide* (for information on the AMC server and nodes)
  - *AON Programming Guide* (for information on custom Bladelets, custom adapters, and application program interfaces)

Contents

Information About Message Types

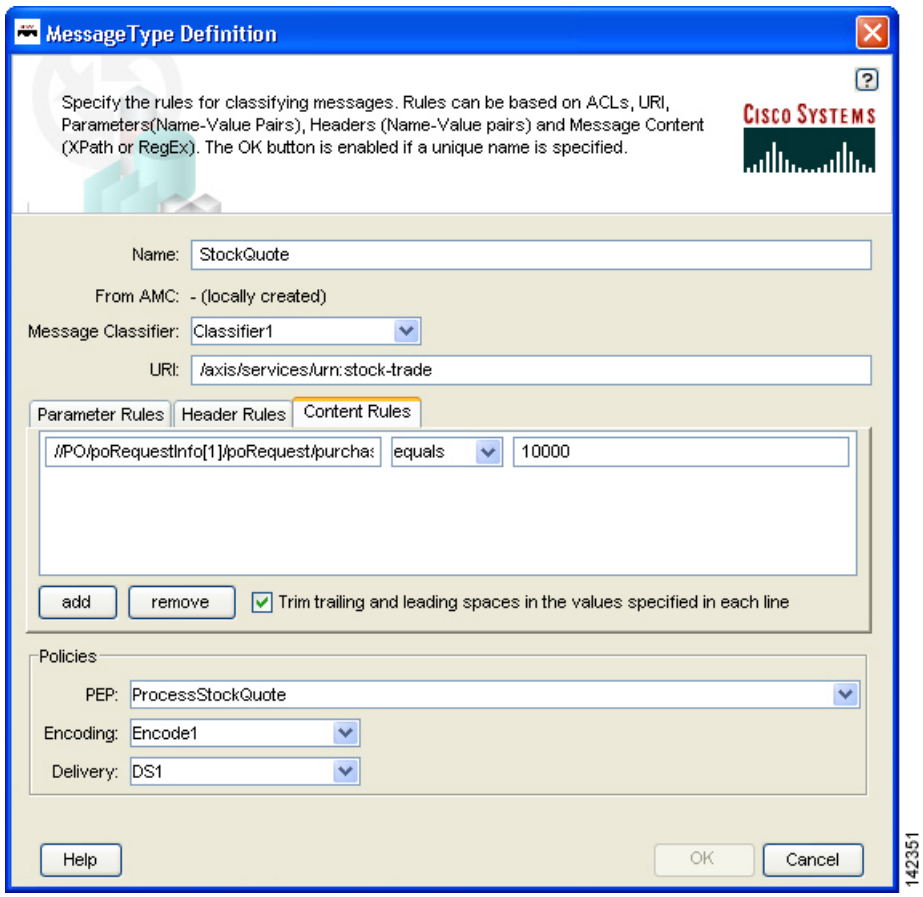
A message type is a filter that determines what type of message a PEP is to process. An AON node identifies a message of interest based on the details you specify in a message type. These details, which can include items such as message content, IP addresses, or message headers, are used to trigger the associated PEP. They can also be used to trigger encoding and message delivery properties.

Message Type Window and Dialog Boxes

Common tasks involving creating PEPs are discussed in Chapter 1, "Getting Started with Cisco ADS." This section describes how to assign message types.

You assign message types by means of the Message Type Definition window (Figure 4-1) and subsequent dialog boxes. (To open this window, follow the procedure in the "Creating PEPs" section on page 1-9.) Window fields and choices are described in the "Message Type Choices" section on page 4-3.

Figure 4-1 Message Type Definition Window



Message Type Choices

Note

- Only the name is a required field. Other fields are optional, although you will want to configure as many fields as possible to ensure that the node properly identifies messages of interest.
- Many of the following windows allow you to specify values in one or more of the following ways:
  - By typing them in directly
  - By selecting them from a drop-down list
  - By binding the parameter to a specific value

Table 4-1 Message Type Definition Window

Field	Description
Name	Name for the message type.
Message Classifier	Message classifier. IP address and port of the originator and destination of a message. Must already be configured on the AMC server. The full path in AMC is **AMC > Network Nodes > Configure > ACL/Classifier**.
URI	Uniform resource identifier (URI) for the message.
Rule Type	Rule type: • Parameter Rules—Identify messages based on data contained in their URLs. • Header Rules—Identify messages based on data contained in their headers. • Content Rules—Identify messages based on content in the message body.
Rules	Rule, typically with the following components: • Item that the node should identify • Equals or Not Equals • Value associated with the item The following example shows an XPath-expression content rule: `//PO/poRequestInfo[1]/poRequest/purchaseInvoiceNumber` **`equals`** `100000` When the node receives an XML message in which the invoice number equals 100000, the message is further processed by the associated PEP and other policies. **Note** Due to limitations within the Windows user interface, it is possible to unknowingly include spaces before or after an expression, especially if you are pasting a lengthy string from another application. These extra spaces can cause processing errors when actual messages arrive without the spaces. To avoid this (that is, to ignore extra spaces), check the “Trim trailing and leading spaces...” box.

Table 4-1 Message Type Definition Window (continued)

Field	Description
PEP	Existing PEP to be used to process messages identified by the message type.
Encoding	Encoding property that defines how and when a message is to be compressed. Must already be configured on the AMC server. The full path in AMC is **Properties > Application > Node > Edit Properties > Encoding**.
Delivery	Delivery property that affects the ordered and reliable delivery of messages to destinations. Depending on the configuration, multiple messages can be delivered in a predefined order, and the node continues attempting to deliver a message until successful. Must already be configured on the AMC server. The full path in AMC is **Properties > Application > Node > Edit Properties > Delivery Connection**.