CHAPTER 3

ADS Bladelets Reference

Revised: March, 2007, OL-11798-01

A Bladelet is an operation that is performed on a message. It is a user defined software component that implements certain interfaces and provides a useful unit of functions. For example, Authentication bladelet provides authentication against various authentication schemes such as, LDAP, Kerberos, and Netegrity; it will not do anything else.

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:
  - Getting Started with Cisco ADS
  - Setting Bladelet Properties, Variables, and Rules
  - ADS PEP Attributes Reference
  - ADS Message Types Reference
  - E-Mail to Cisco ADS Support
- 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:

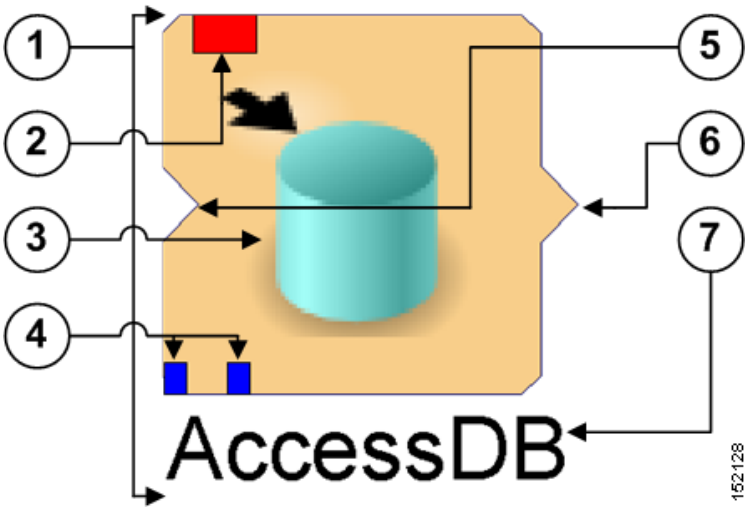
<configuration-group>

<configuration-subgroup>

<parameter-group> and <parameter>

Figure 3-1 shows the various components of a Bladelet. (The example shown below is an Access DB Bladelet.)

Figure 3-1 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 Choices

This section describes the predefined Bladelets that ADS displays in its Task Pane, page 1-9, in Getting Started with Cisco ADS. 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 3-3
- External Access Category, page 3-4
- General Category, page 3-15
- Logic Category, page 3-25
- Message Handling Category, page 3-38
- Routing Category, page 3-70
- Security Category, page 3-93
- Transformation Category, page 3-146
- Miscellaneous Category, page 3-148

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
- Break 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.

Break Marker

Use the Break marker only inside loops. It is only allowed in a Loop and cannot be used elsewhere. You cannot place any other bladelets after the Break marker. The Break marker is used to exit out of the loops. For details, see the Loop Bladelet.

There are no properties to set for this marker.

External Access Category

In the External Access Category, there are two Bladelets:

- Access HTTP, page 3-4
- Access DB, page 3-11

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, Dependencies, and Restrictions

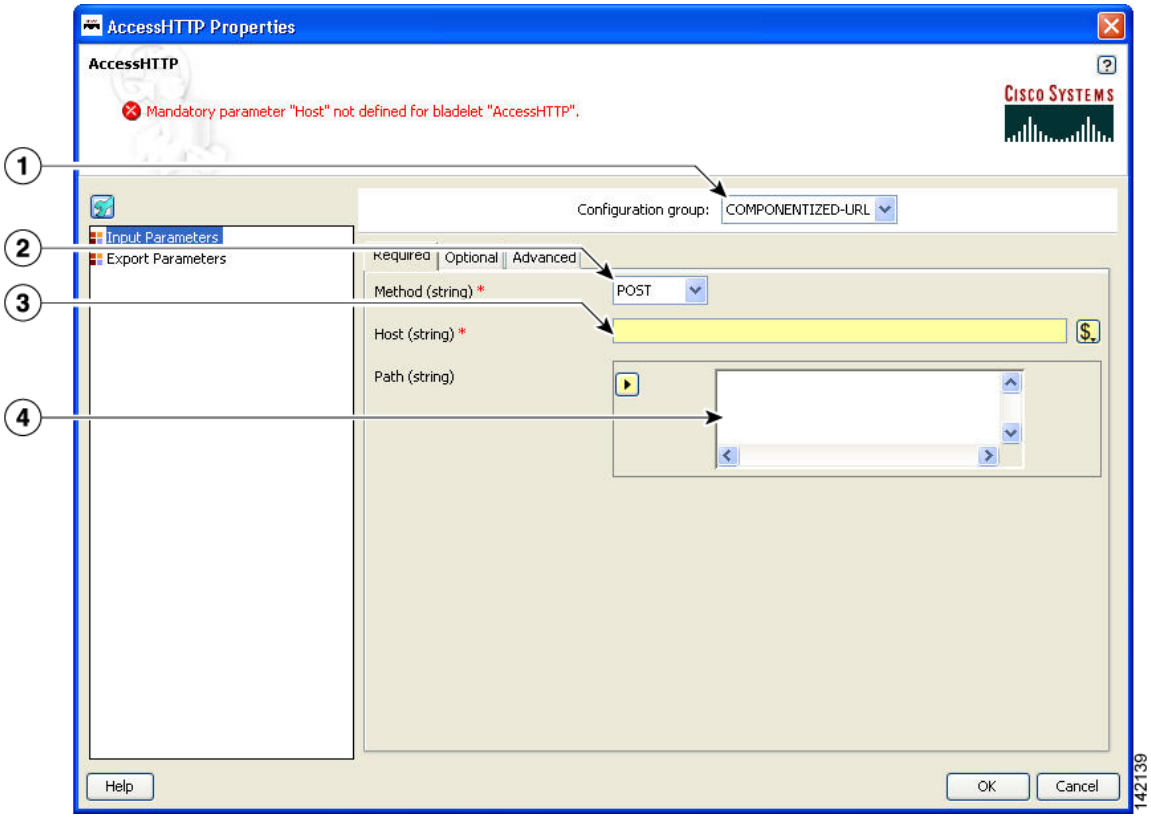
AccessHTTP is synchronous, and thus cannot be used to execute a PEP running on the same node.

Details

Figure 3-2 to Figure 3-4 show required, optional, and advanced settings for the Componentized URL Configuration group.

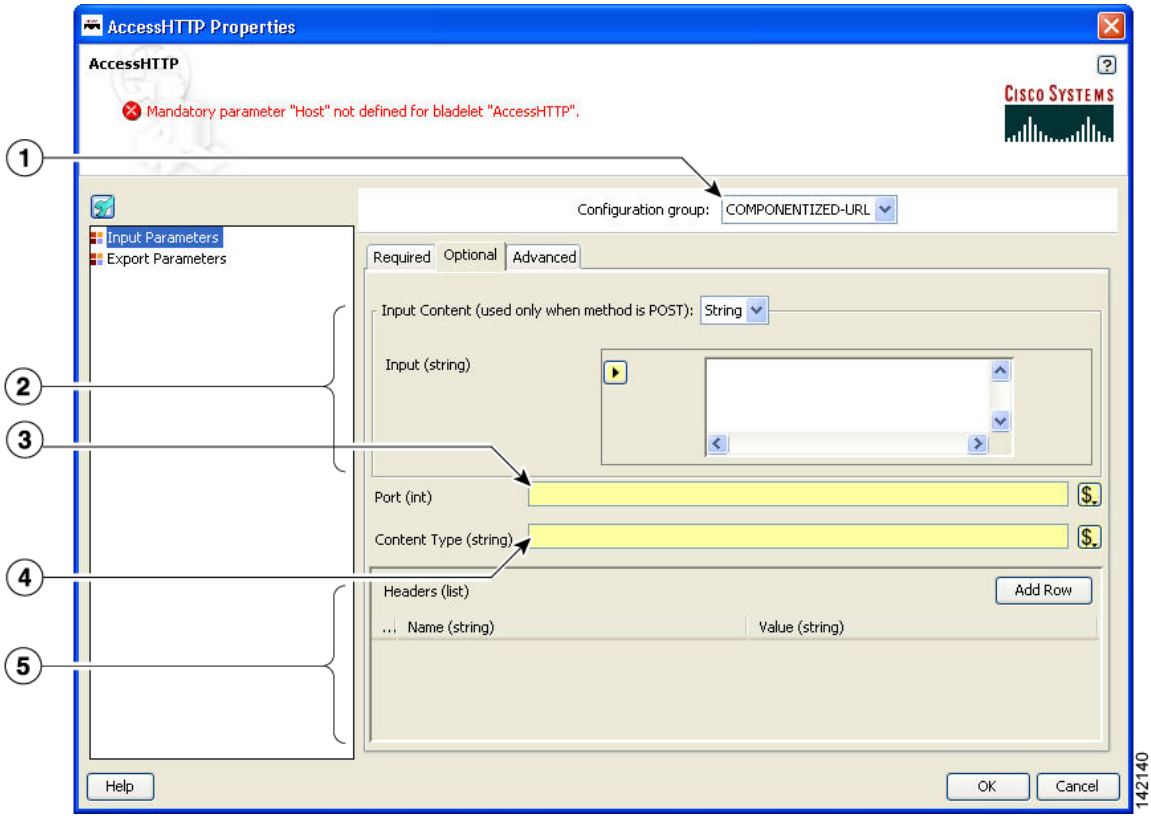
Figure 3-5 to Figure 3-7 show required, optional, and advanced settings for the Normal URL Configuration group.

Figure 3-2 Access HTTP Properties Window—Input Parameters, Componentized URL, Required Tab



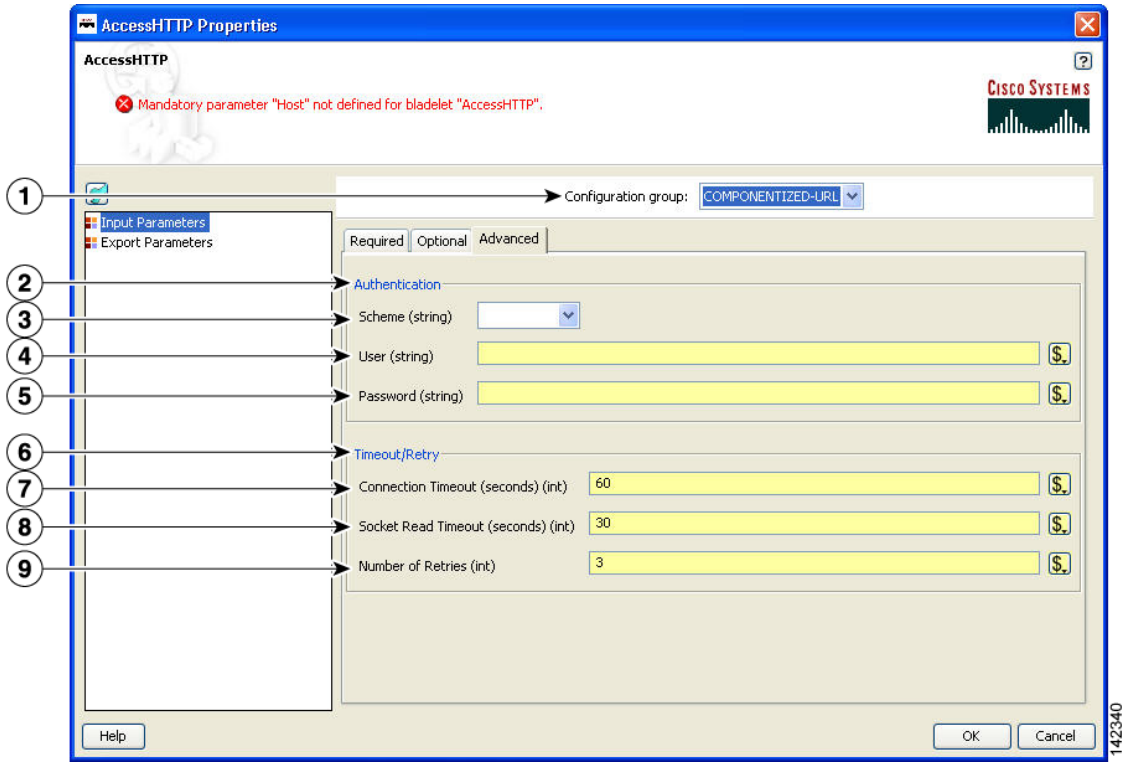
1	Configuration group	Configuration group, set here to Componentized URL.
2	Method	Method. Choices: Post and Get.
3	Host	Hostname or IP address of the HTTP server.
4	Path	Path portion of the URL (/index.jsp).

Figure 3-3 Access HTTP Properties Window—Input Parameters, Componentized URL, Optional Tab



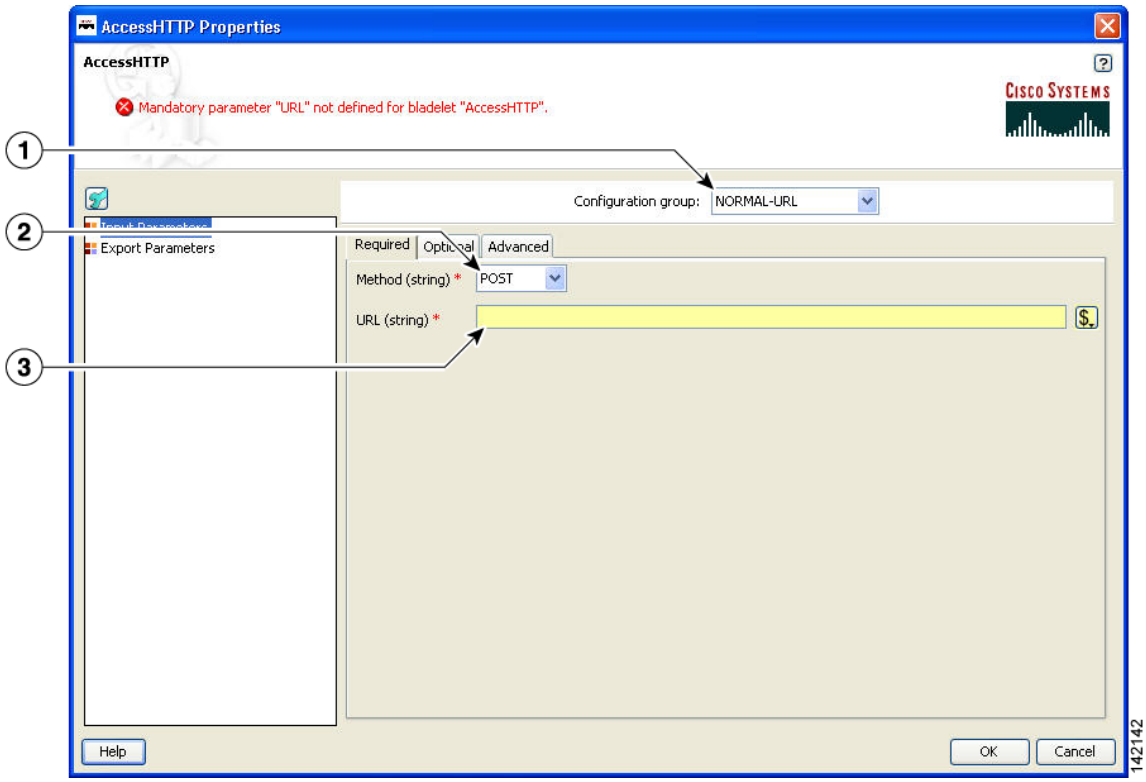
1	Configuration Group	Configuration group, set here to Componentized URL.
2	Port	Port number to be used. Defaults to 80.
3	Content Type	MIME type of the content.
4	Headers	Header name and corresponding value (string types).
5	Input Content	Payload of the HTTP call. Required only in case of POST.

Figure 3-4 Access HTTP Properties Window—Input Parameters, Componentized URL, Advanced Tab



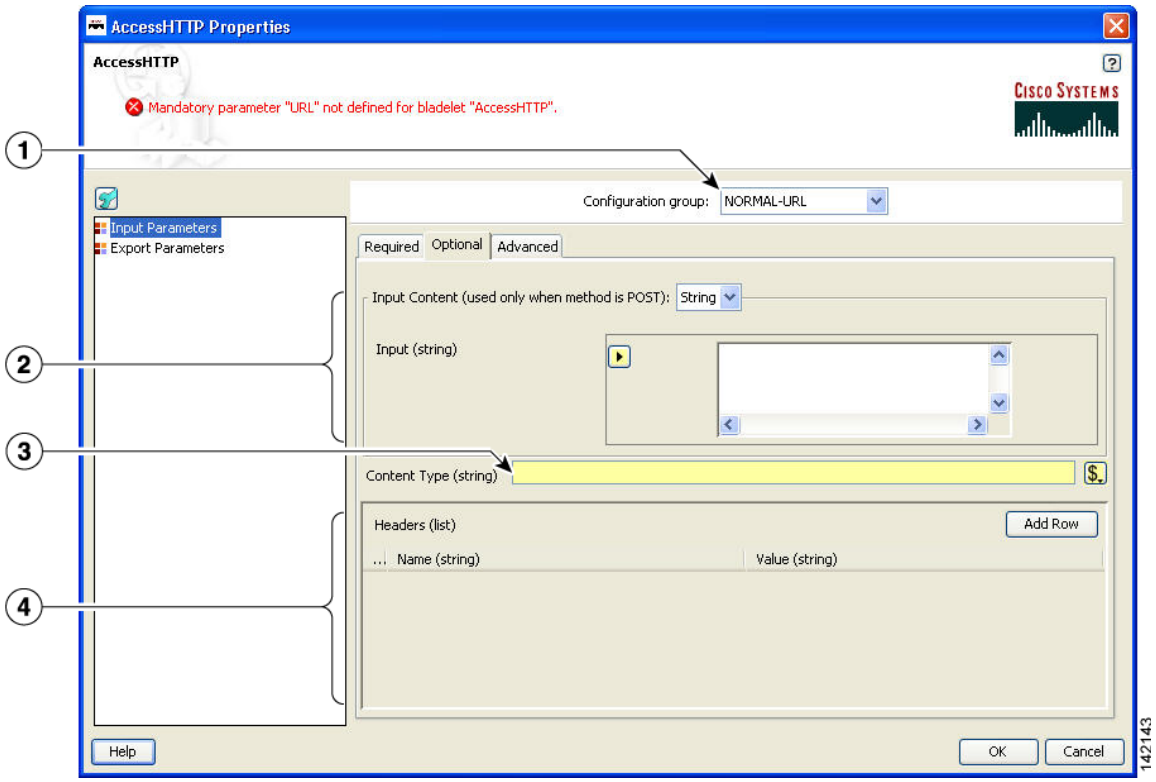
1	Configuration Group	Configuration group, set here to Componentized URL
2	Authentication	Basic HTTP is the only authentication scheme supported today.
3	Scheme	Basic HTTP.
4	User	User ID.
5	Password	Password.
6	Timeout/Retires	Timeout and retries to establish a connection.
7	Connection Timeout (seconds)	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 3-5 Access HTTP Properties Window—Input Parameters, Normal URL, Required Tab



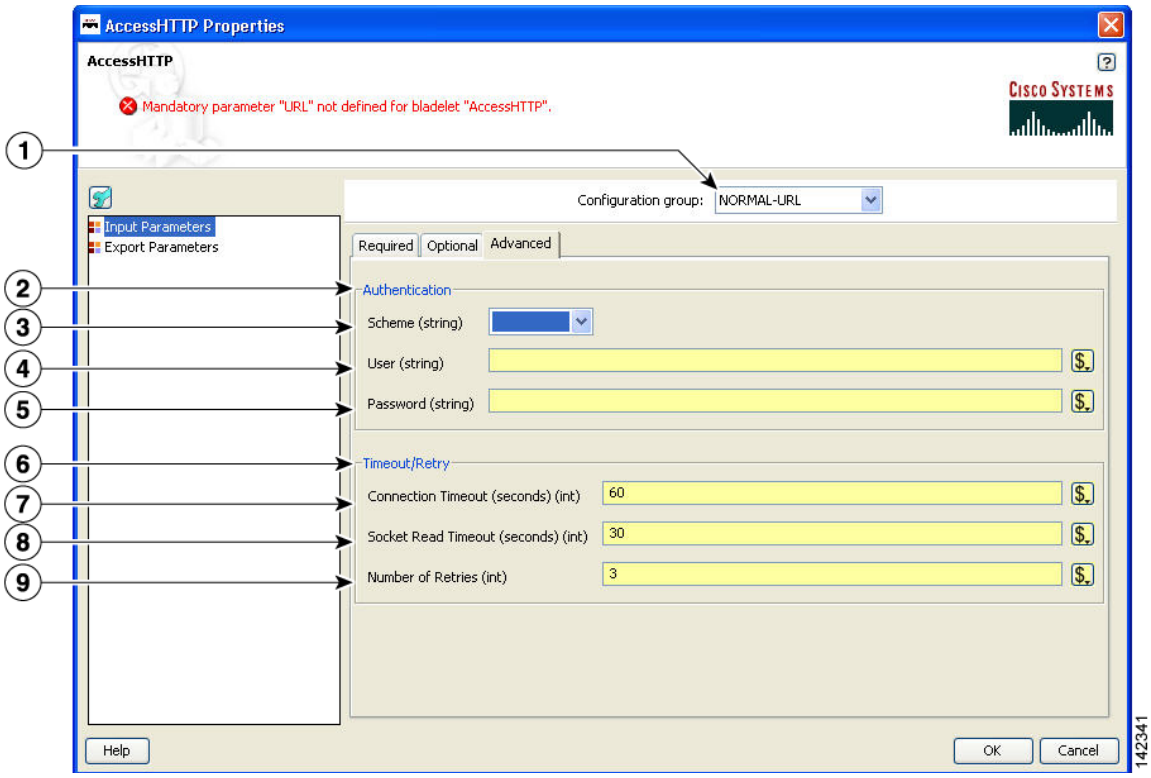
1	Configuration Group	Configuration group, set here to Normal URL.
2	Method	Method. Choices: POST or Get.
3	URL	Complete URL.

Figure 3-6 Access HTTP Properties Window—Input Parameters, Normal URL, Optional Tab



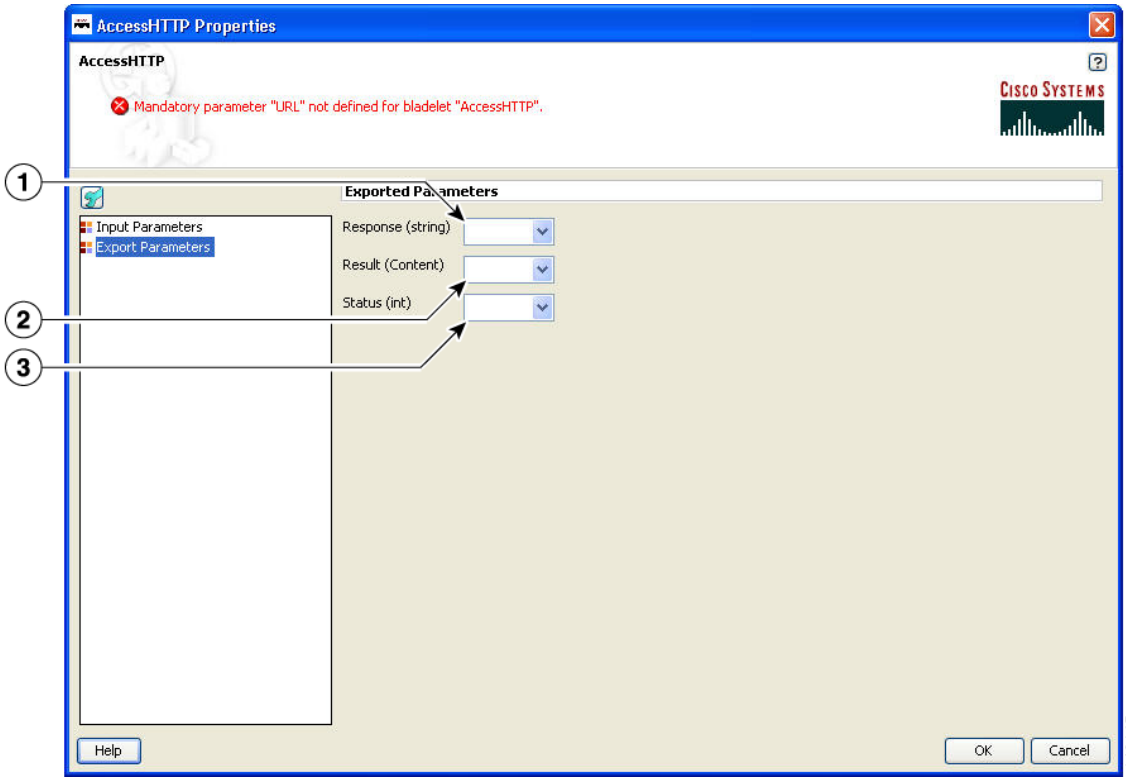
1	Configuration group	Configuration group, set here to Normal URL.
2	Content Type	MIME type of the content.
3	Headers	Header name and corresponding value.
4	Input Content	Payload of the HTTP call. Required only in case of Post.

Figure 3-7 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 3-8 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

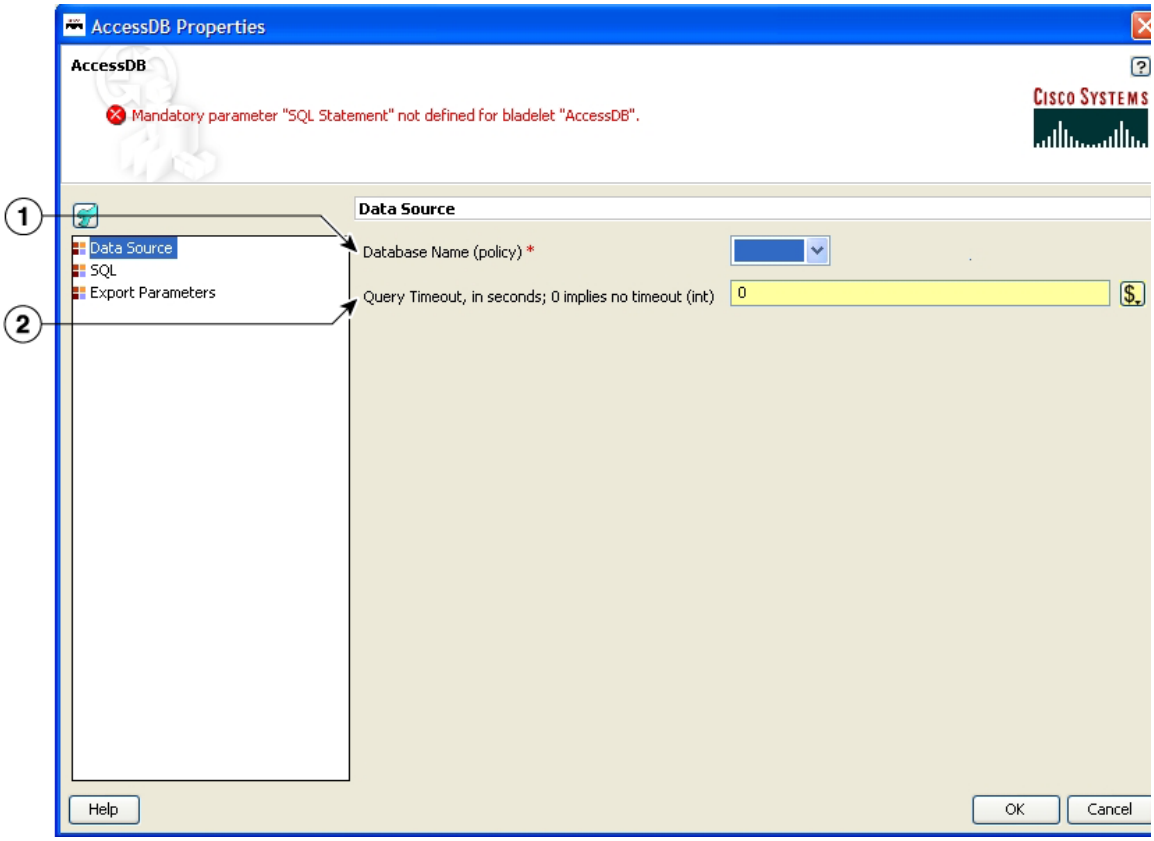
Use this Bladelet to make a SQL call out to a database.

Prerequisites, Dependencies, and Restrictions

Access DB does not work with binary objects; it only works with basic types, for example string, int, and so on.

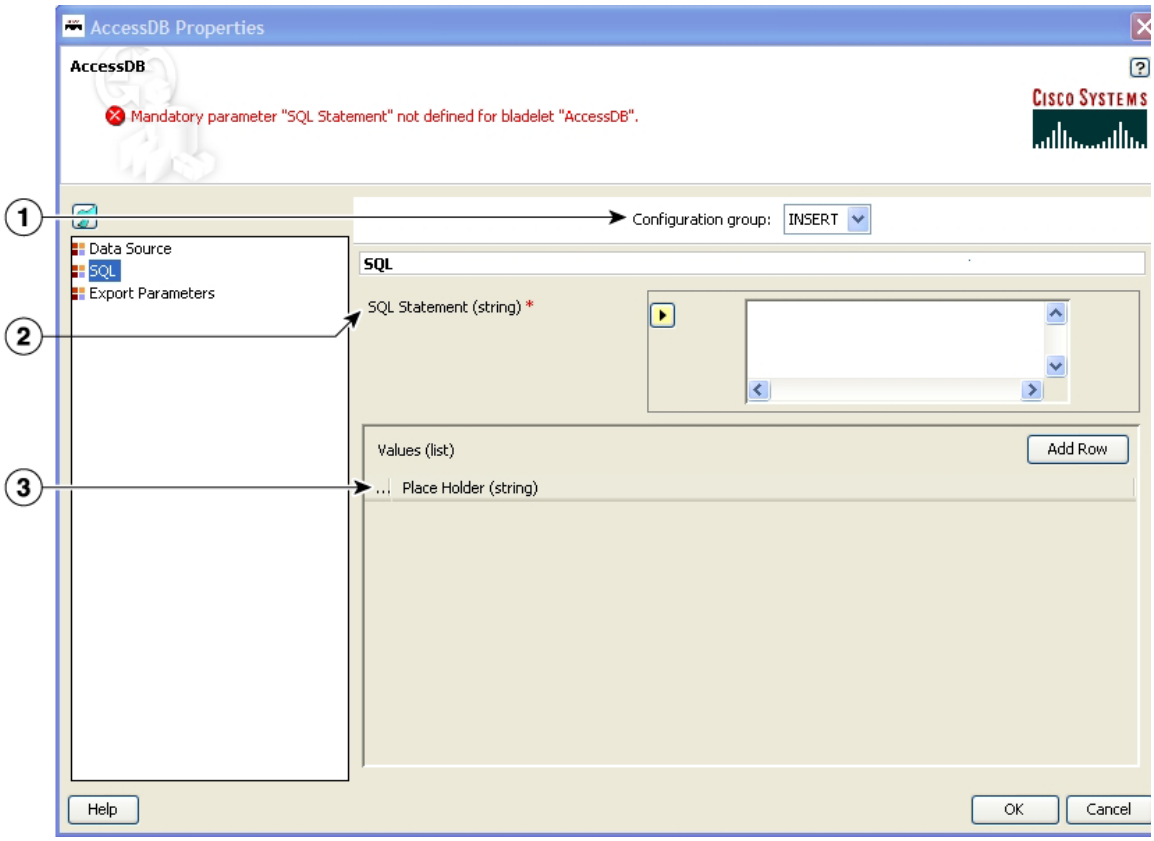
Details

Figure 3-9 Access DB Properties Window—Data Source



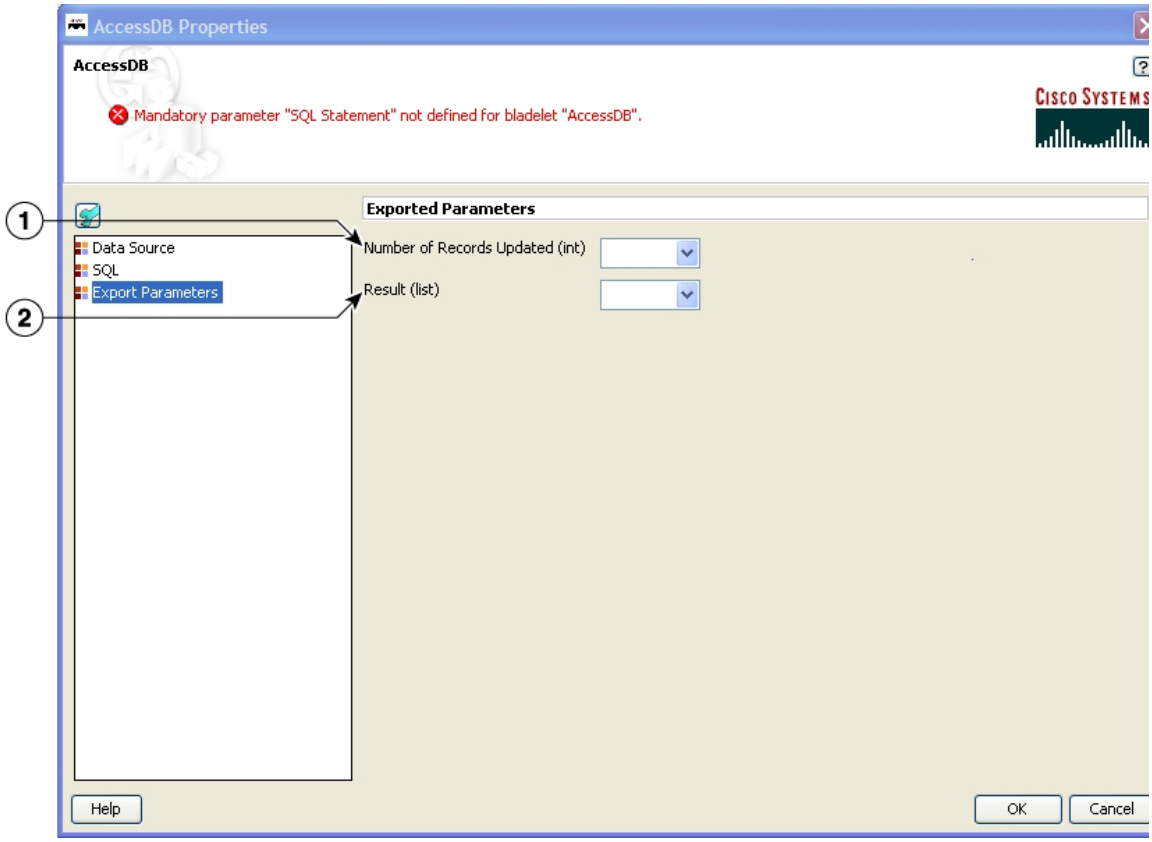
1	Database Name	Property set. After opening a project in AMC the path is **Properties** > **Application** > **Global** > **Databases**.
2	Query Timeout in Seconds	Desired waiting time set for a query to get executed.

Figure 3-10 Access DB Properties Window—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 3-11 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

Summary

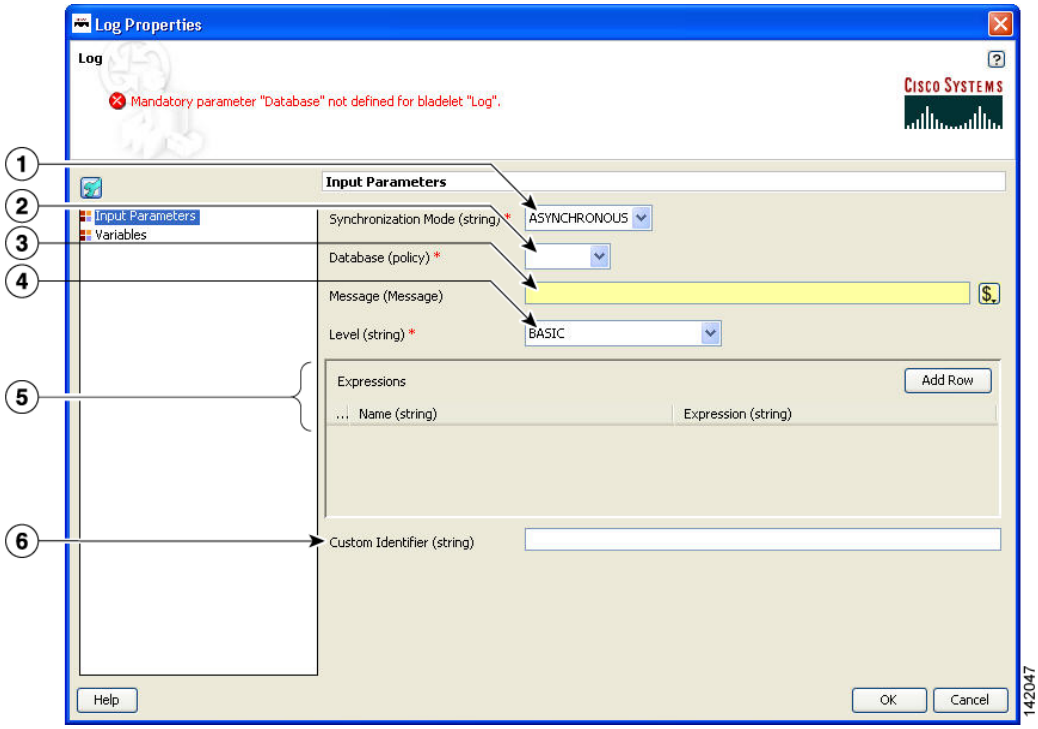
The Log Bladelet allows you to log message contents, PEP variables and other important data related to the message, message class, source, destination, time stamps, and PEP name.

Prerequisites and Dependencies

None.

Details

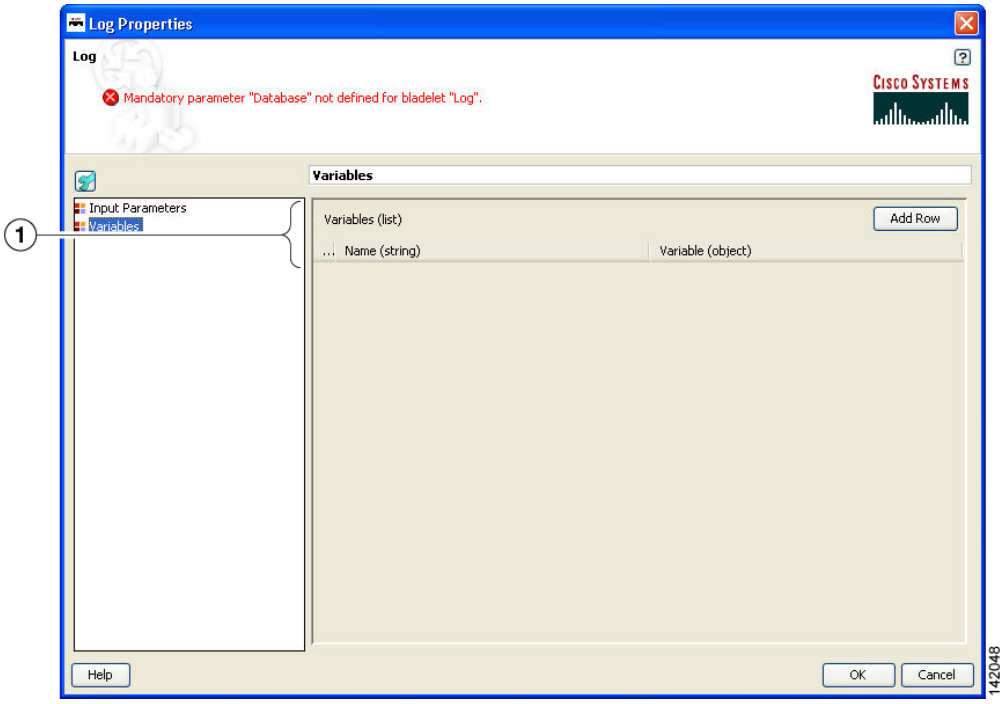
Figure 3-12 Log Properties Window—Input Parameters



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. After opening a project in AMC the full path is **Properties** > **Application** > **Node** > **Message Log Domain**.
3	Message	Auto complete message field.

4	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.)
5	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
6	Custom Identifier	Optional. String to identify this message log entry.

Figure 3-13 Log Properties Window—Variables



1	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

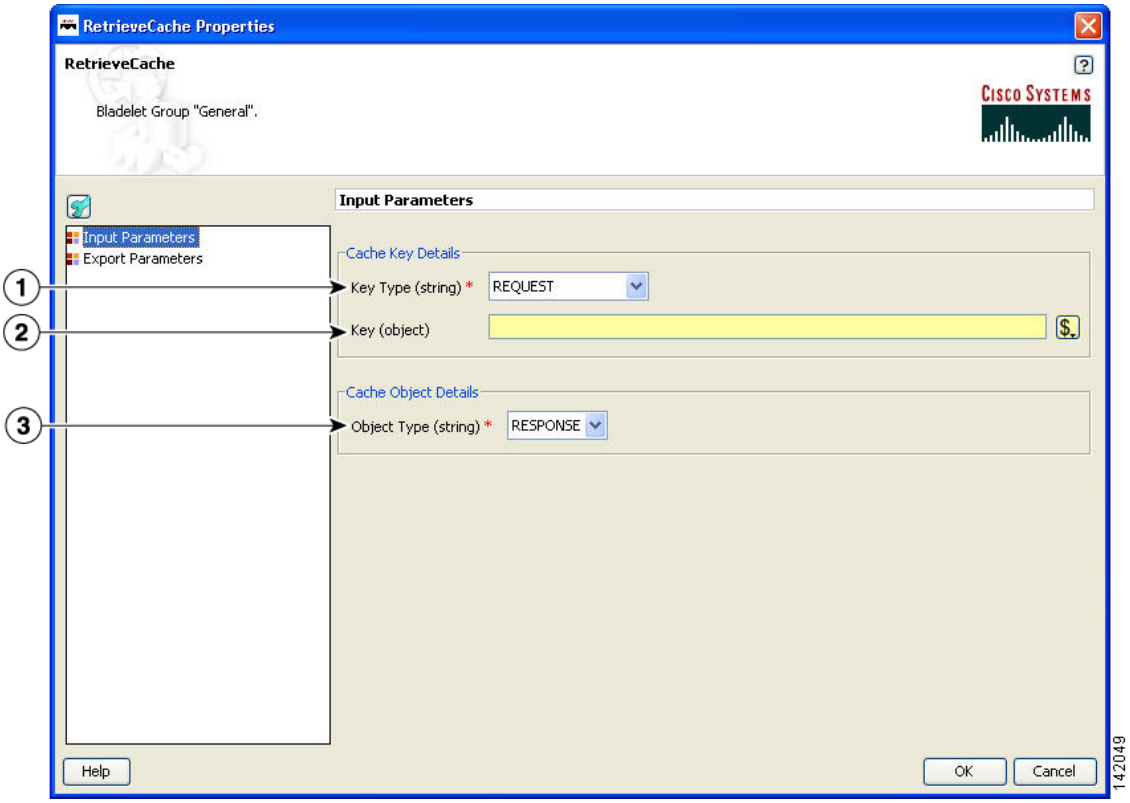
This Bladelet retrieves data from two named caches configured on an AON node. The named caches are response and variable. The response cache caches arbitrary messages and server responses. The variable cache caches PEP variables. The response cache is distributed and the variable cache is node local. Populate these named caches by using the CacheData Bladelet.

Prerequisites and Dependencies

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

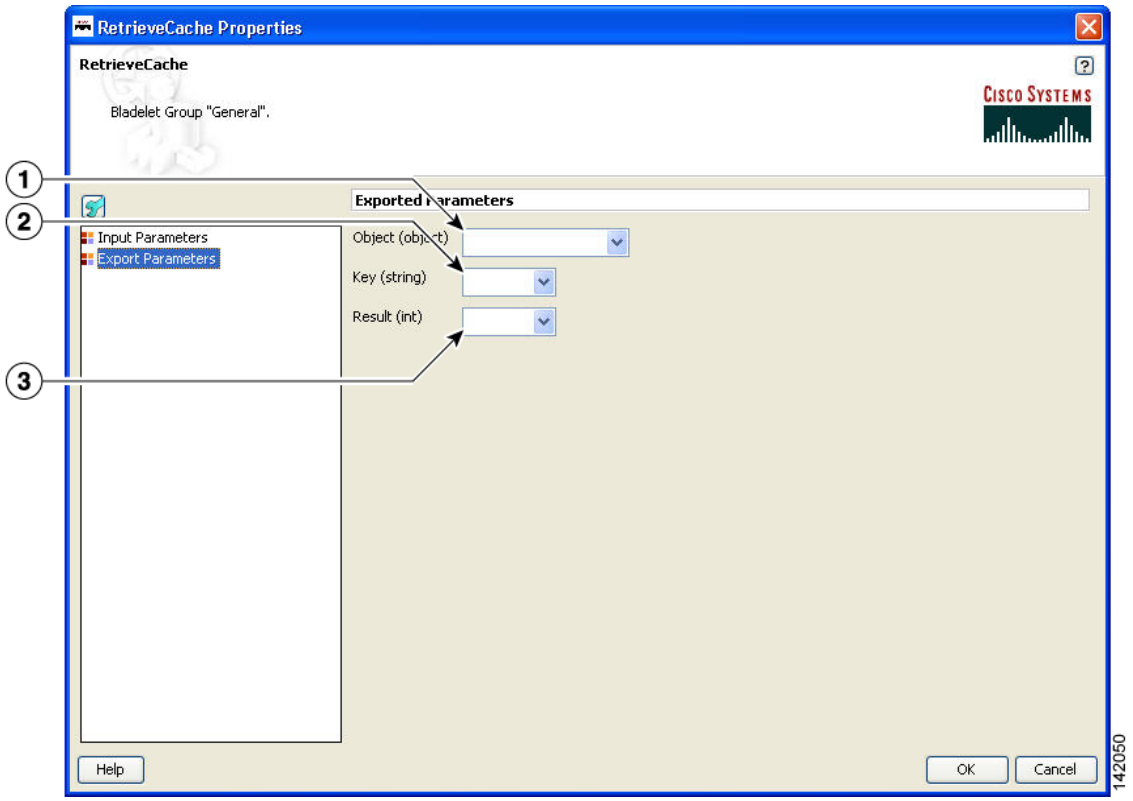
Details

Figure 3-14 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 • Message—Message cache

Figure 3-15 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 response type 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 arbitrary messages and server responses. The "variable" cache caches PEP variables. The response cache is distributed and the variable cache is node local. You can retrieve data from the named caches by using the RetrieveCache Bladelet. You can also set and retrieve data from the named and "variable" caches by using the Caching Service API exposed to Custom Bladelets.

Note Objects are put into the response cache asynchronously and the variable cache synchronously with PEP execution.

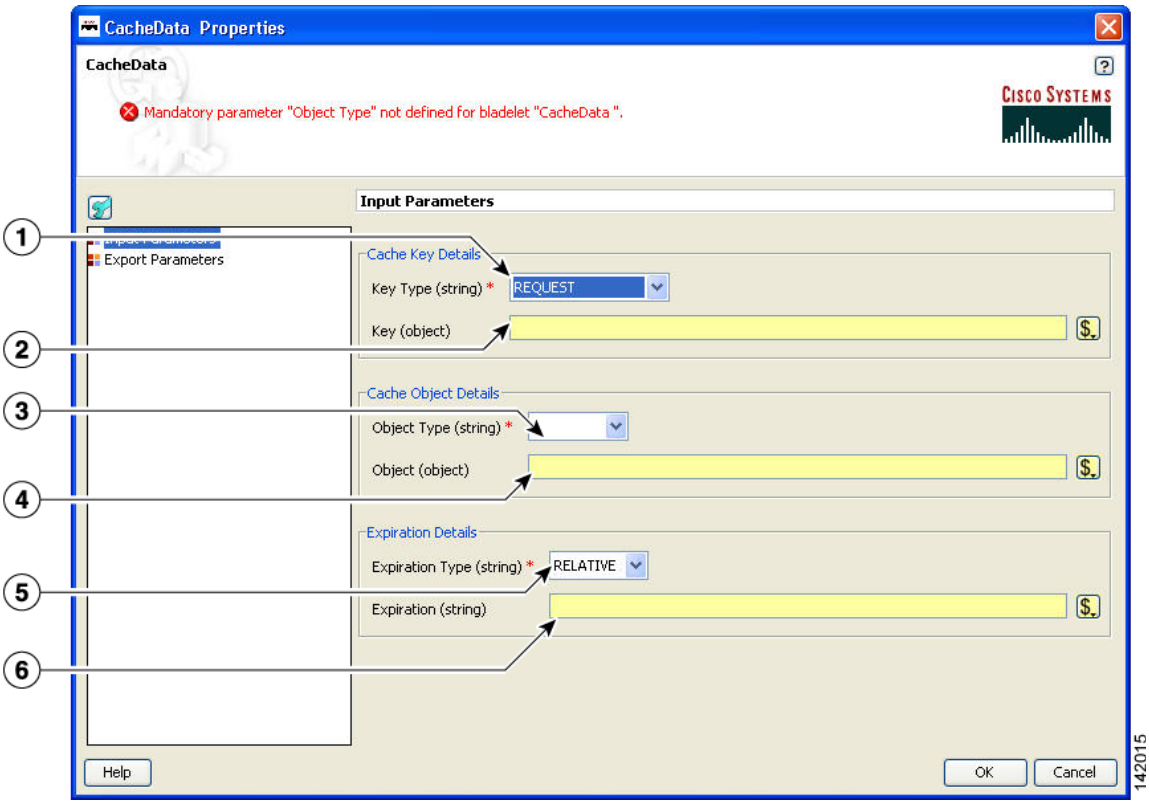
Prerequisites and Dependencies

- 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.

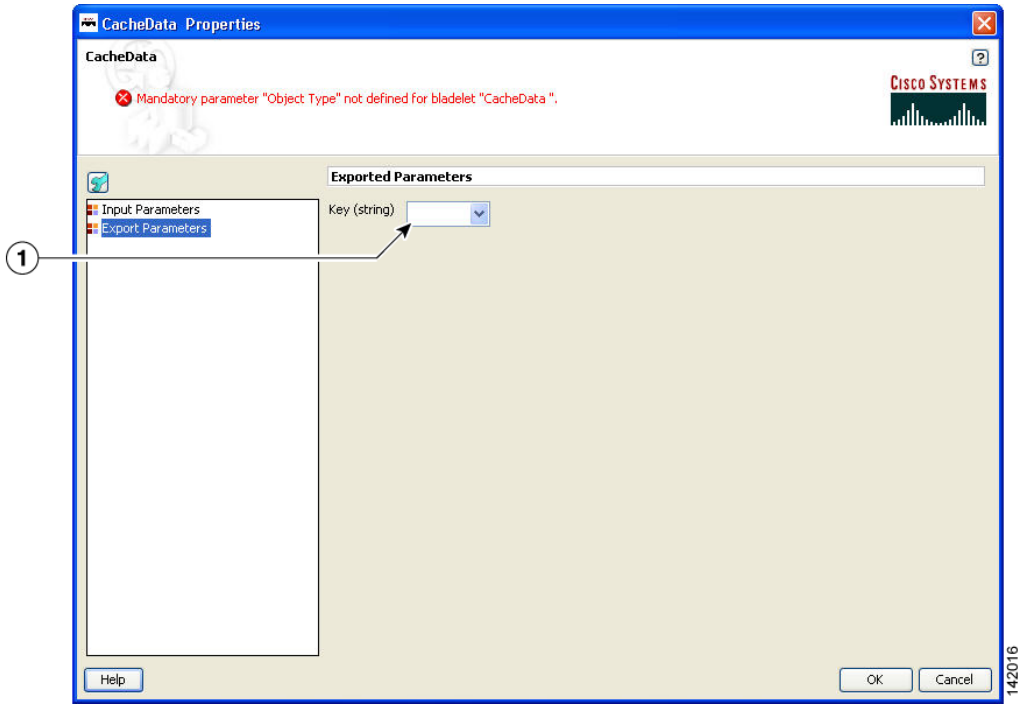
Figure 3-16 Cache Data Properties Window—Input Parameters



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. • Message—Caches the message specified in the Object parameter in the response cache.

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 3-17 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:

- Loop, page 3-26
- Scope, page 3-30
- Find, page 3-31
- Branch, page 3-35

Loop

Summary

The Loop Bladelet allows you to construct a PEP and apply repeated business logic processing based on the number of items in the data types—Counter, Iterator, or Map—over which the loop is performed. This construct is identical to a loop which is found in the Java or C programming languages. The Loop Bladelet is represented as a block in which other bladelets are placed.

Figure 3-18 Loop Bladelet

Prerequisites and Dependencies

None.

Details

A PEP can be viewed as a program expressed in the PEP Description Language (PDL). PDL is a programming language and defines the variable types used in a PEP as the fundamental data types. The PDL exposes a number of data types—List, Map, and Iterator. At runtime, these data types represent a collection of other data types (List and Map) or a cursor into this collection (Iterator). When handling these data types, it is almost always required to do some kind of repeated processing (loop) for each entry in the collection.

Note A Break marker is only allowed in a Loop Bladelet and cannot be used elsewhere.

The Loop Bladelet has three different data types—Counter, Iterator, and Map.

Counter

The Counter Loop is used when a given operation (for example, another bladelet needs to be executed) needs to be performed specific number of times. It is similar to a “for” loop with counter variable being initialized to a user specified value and incriminated by a user specified amount. The loop terminates at a user specified end value. See Figure 3-19 for the details of the Counter Loop parameters.

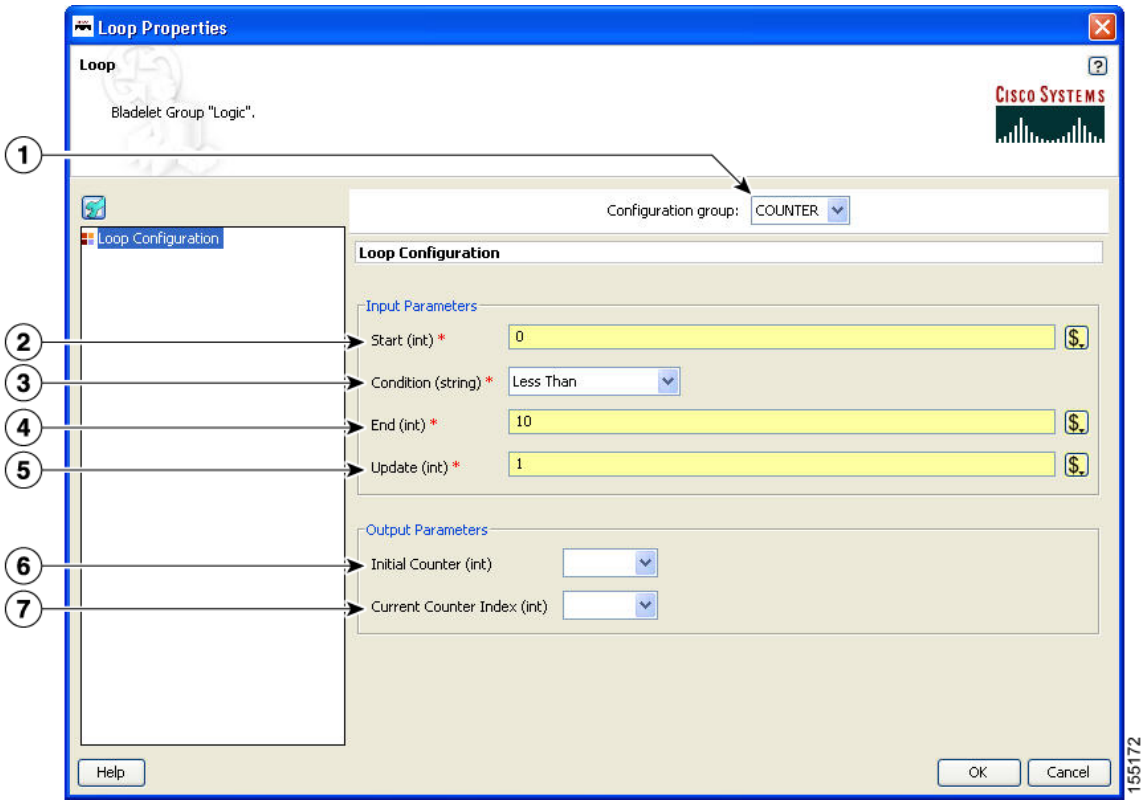
Iterator

The Iterator Loop must be used to loop through a list. The current object and index is available at each loop iteration and can be used by bladelets within the loop. See Figure 3-20 for the details of the Iterator Loop parameters.

Map

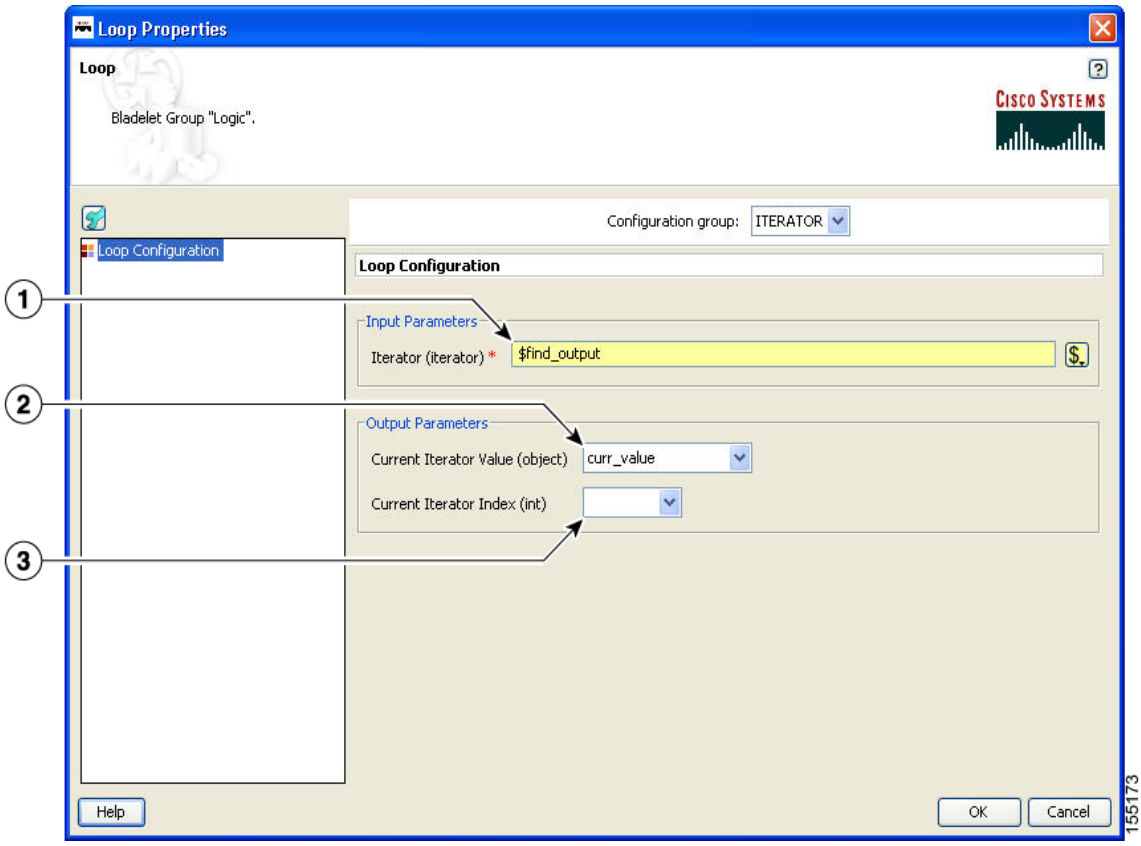
The Map Loop is used to iterate over a map. It exposes both the current key and current value at each loop iteration, both of which can be used by bladelets within the loop. See Figure 3-21 for the details of the Map Loop parameters.

Figure 3-19 Loop Properties Window—Counter



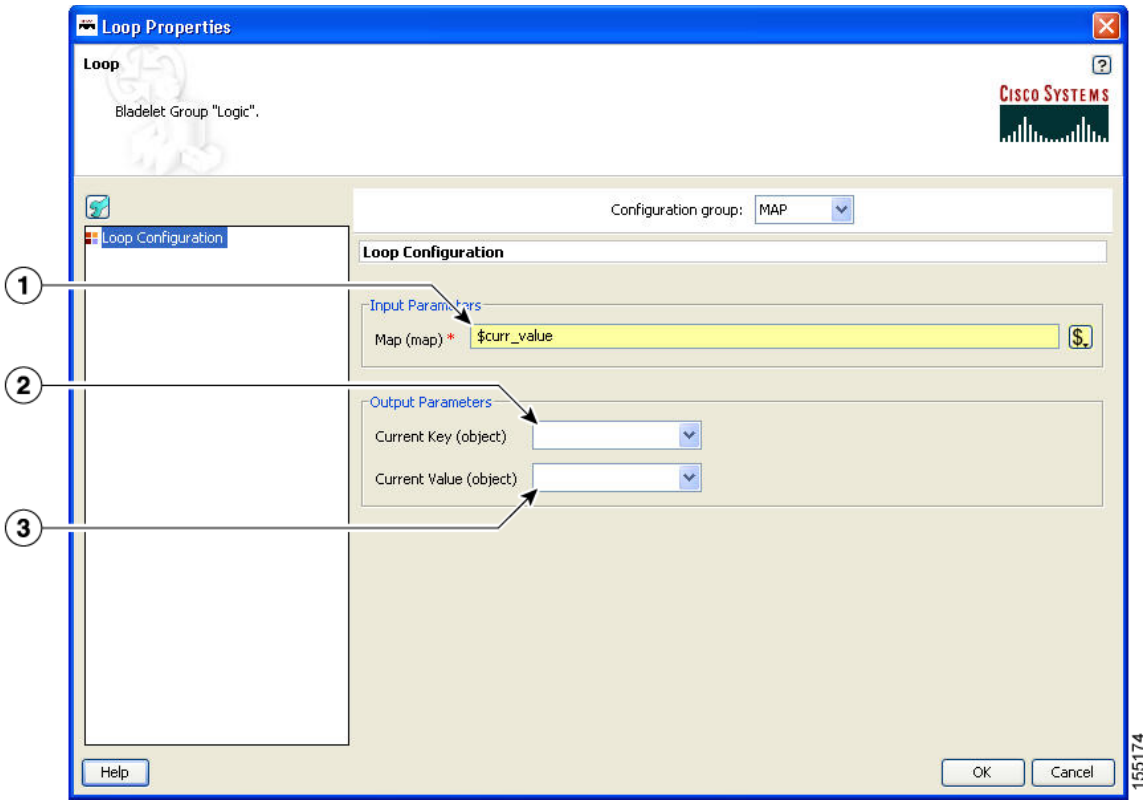
1	Configuration Group	Configuration Group, set here to Counter.
2	Input Parameters—Start	Initializes the loop with the number where we start counting; the index starts here.
3	Input Parameters—Condition	This condition must be satisfied at every iteration of the loop. Check if the current index is “less than,” “less than or equal to,” “greater than,” “greater than or equal to” than the End value, in order to determine whether to continue with the next iteration.
4	Input Parameters—End	Determines when the loop will terminate.
5	Input Parameters—Update	Number of updates of the index after each loop iteration.
5	Output Parameters—Initial Counter	The start index. It is always same as the start index of the input parameter.
7	Output Parameters—Index	Index at which loop iteration ends.

Figure 3-20 Loop Properties Window—Iterator



1	Input Parameter—Iterator	A pointer to the start of a list of elements. At every iteration of the loop, the subsequent element in the list will be traversed using this iterator. **Note** This is an auto complete field and provides a list of variables.
2	Output Parameter—Current Iterator Value	The element that is pointed to by the iterator at this iteration of the loop.
3	Output Parameter—Current Iterator Index	The index of the element that is pointed to by the iterator at this iteration of the loop.

Figure 3-21 Loop Properties Window—Map



1	Input Parameter—Map	A collection of elements. Each element is comprised of a key and a value. At every iteration of the loop, the subsequent element in the map will be traversed and the associated key and value will be exposed.
2	Output Parameter—Current Key	The key of the element that is being traversed at this iteration of the loop.
3	Output Parameter—Current Value	The value of the element that is being traversed at this iteration of the loop.

Outcome

Loops can be nested to arbitrary levels and there is no pre-defined limit to the number of loops that can be used in a PEP. For each type of data that the loop executes over, a different set of PEP variables are exported.

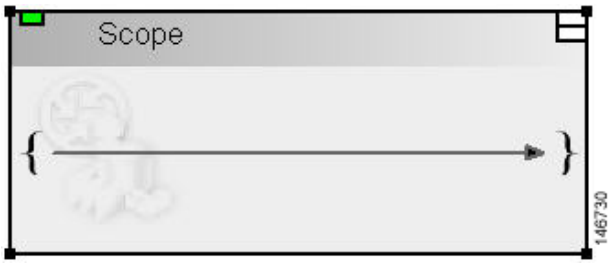
Exceptions

None.

Scope

Summary

The Scope Bladelet is used to define a physical block within a PEP that allow localized definition of variables and business logic.



Prerequisites and Dependencies

None.

Details

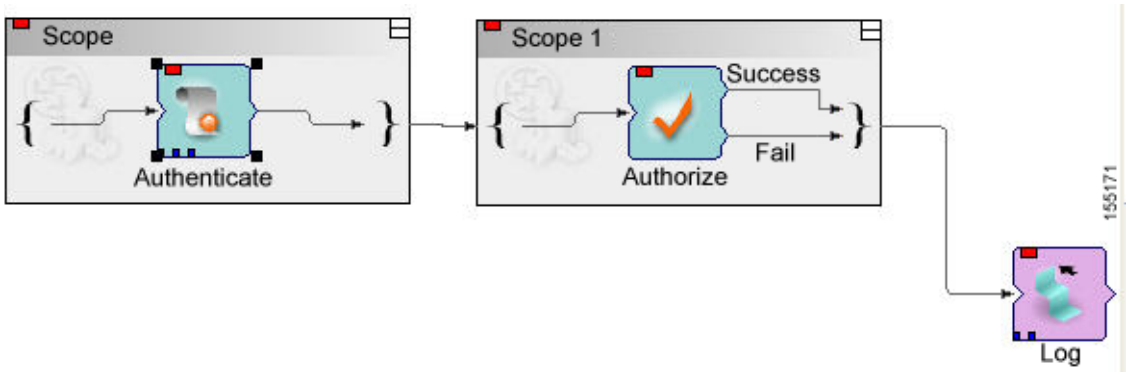
There are no properties to set for this Bladelet.

The scope construct in the PEP Description Language (PDL) allows you to define physical blocks within the PEP that allow localized definition of variables and business logic. This is similar to the {} operator in the Java or C++ programming languages where a block of execution bounded by the braces serves as a container for variables that are not visible outside the execution block.

A single scope block can only have one immediate parent scope (the scope block within which it is nested) and the global scope (always present) is the top level scope and is the root node in the tree representation of all the scopes in a PEP. There are some semantics that apply when using a scope block. In general a scope block is most useful when it is used to restrict the scope of a PEP variable ensuring that a variable defined in one part of the PEP is not available for use in other parts of the PEP. The restrictions on the variables in a Scope are dependent on the visibility of the variable in that block. A scope block can recognize those variables that are defined in its parent scope.

In Figure 3-22 the root node of the tree represents the default Global scope that is always present in a PEP. You can add additional scopes by dragging the scope construct from the palette to the canvas and including it at any point in the flow. Figure 3-22 shows a sample PEP containing two explicit scope blocks and the global scope (represented by the white background region on the canvas).

Figure 3-22 Sample Scopes within a PEP



In the first scope the Authenticate Bladelet can only use the variables defined with global and Scope. In the second scope, Scope 1, the Authorize Bladelet can only use variables defined with global or Scope 1. The Log Bladelet can only use variables defined with Global or Scope 1.

Outcome

Scopes can be nested within each other with no pre-defined limit on the number of scopes that can be included in a PEP.

Exceptions

None.

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.

The Regex evaluation engine used by the Find bladelet uses Java Regex API from Sun Microsystems, Inc. There are several APIs to chose from, so we recommend that you use the API that matches the whole input string rather than finding only a match. You need to use the API that matches the whole input string because Find Bladelet needs to save the result of Regex evaluation.

For details of Java Regex API from Sun Microsystems, Inc., see http://java.sun.com/docs/books/tutorial/extra/regex/.

Note You can use the Find Bladelet to get the FindResultMapListIterator and refer to the results in the Rules Wizard in Branch Bladelet.
Or
You can also use the Rules Wizard to perform Regex evaluation.

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.

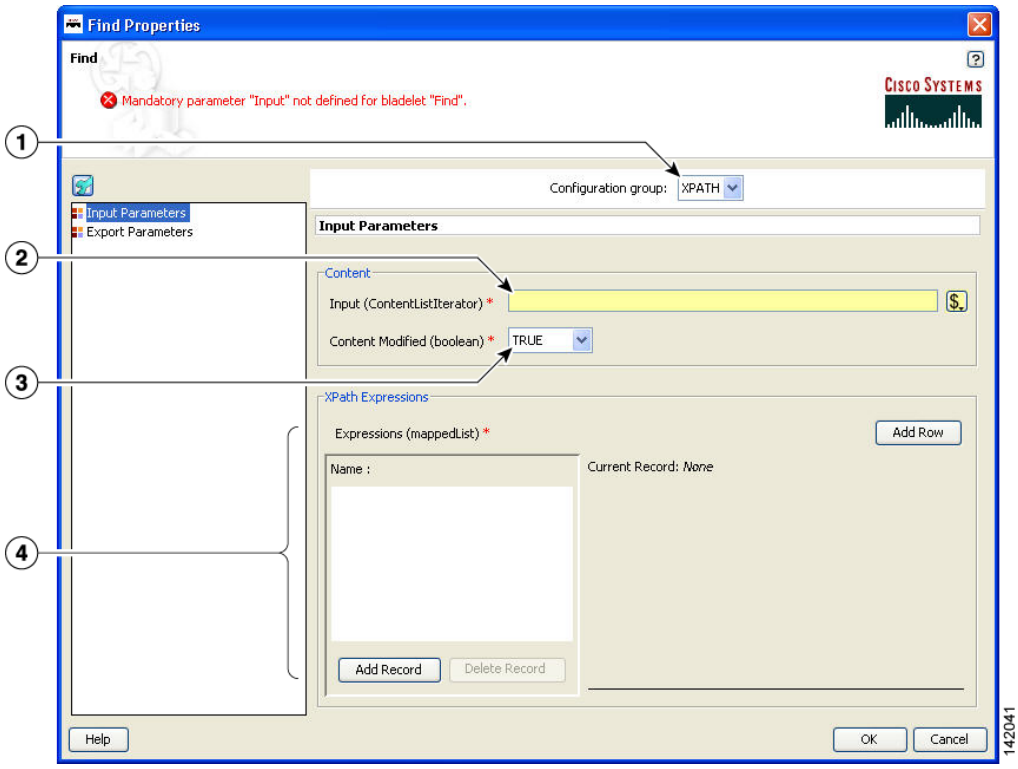
Note IMPORTANT: Today it is not possible to use the PEP variable-picker dialog to select values from the tree view. **You must enter a specific value to extract the returned results.**

```
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).
```

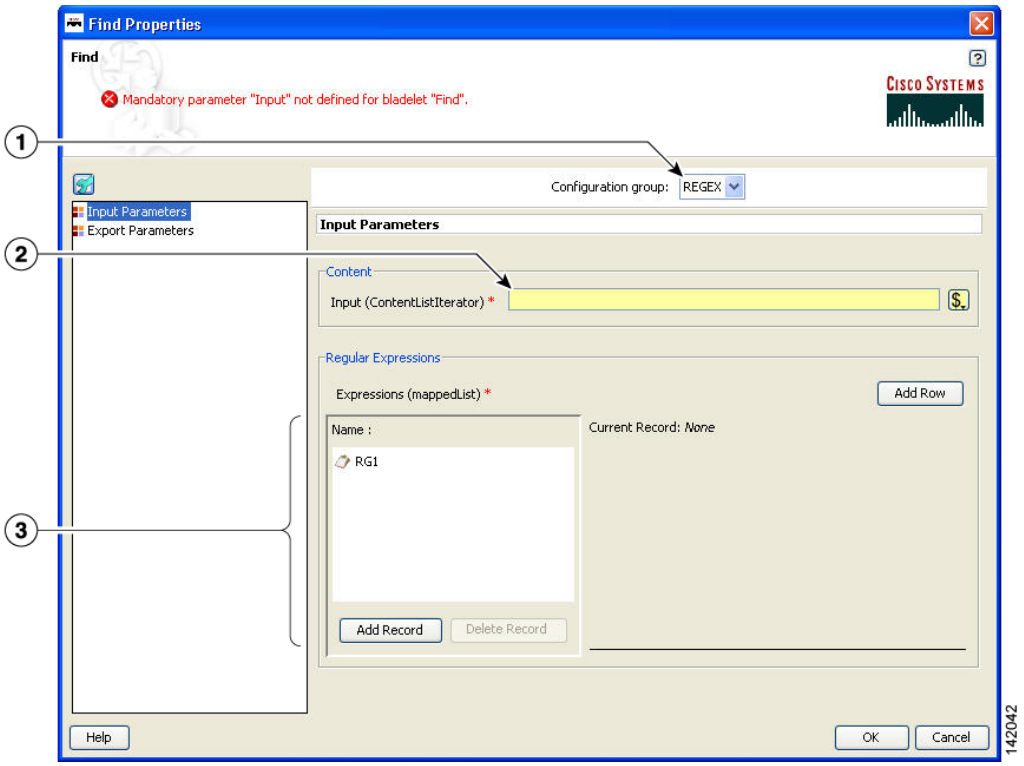
The input parameters for this Bladelet (configuration group is set to XPath) are shown in Figure 3-23. Input parameters for a Bladelet whose configuration group is set to Regex are shown in Figure 3-24.

Figure 3-23 Find Properties Window—Input Parameters, XPath



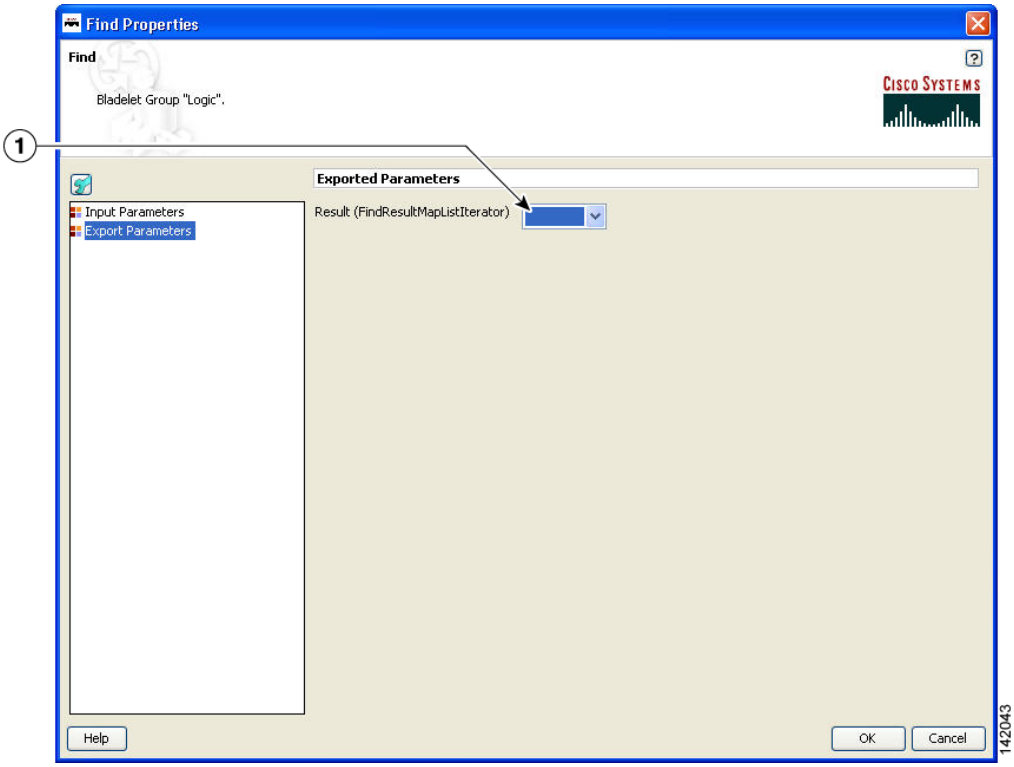
1	Configuration Group	Configuration group, set here to XPath. Valid values are XPath and Regex.
2	Input	Input, such as DVar.
3	Content Modified	Whether or not message content has been modified—for example, by a preceding encryption Bladelet or transformation Bladelet. If this is the first Find Bladelet 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 3-24 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 3-25 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 "Managing Variables" section on page 2-3, in "Setting Bladelet Properties, Variables, and Rules" chapter.

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

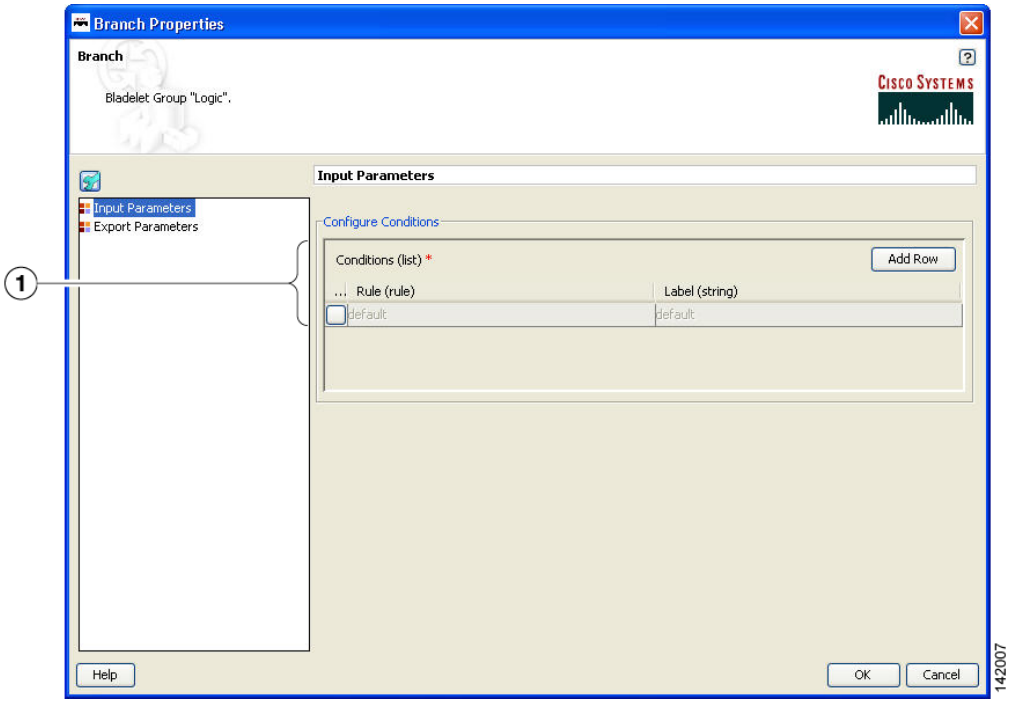
This Bladelet establishes conditions for message route branching based on rules and message labels. There are two main sections in the Branch Properties window.

Prerequisites and Dependencies

None.

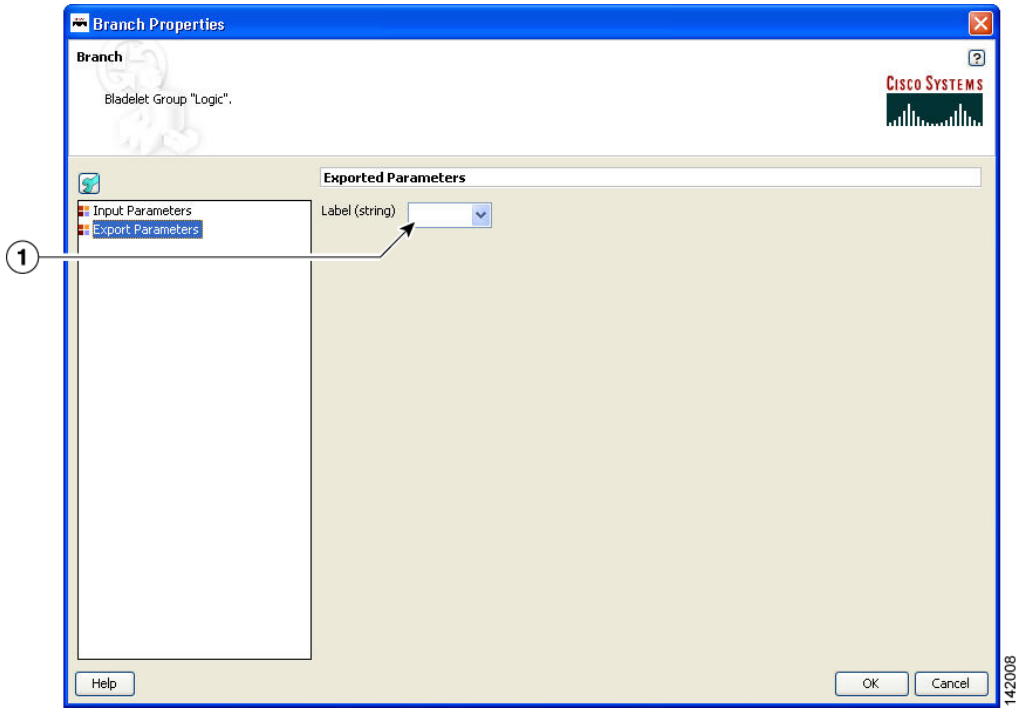
Details

Figure 3-26 Branch Properties Window—Input Parameters



1	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 3-27 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 nine 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.

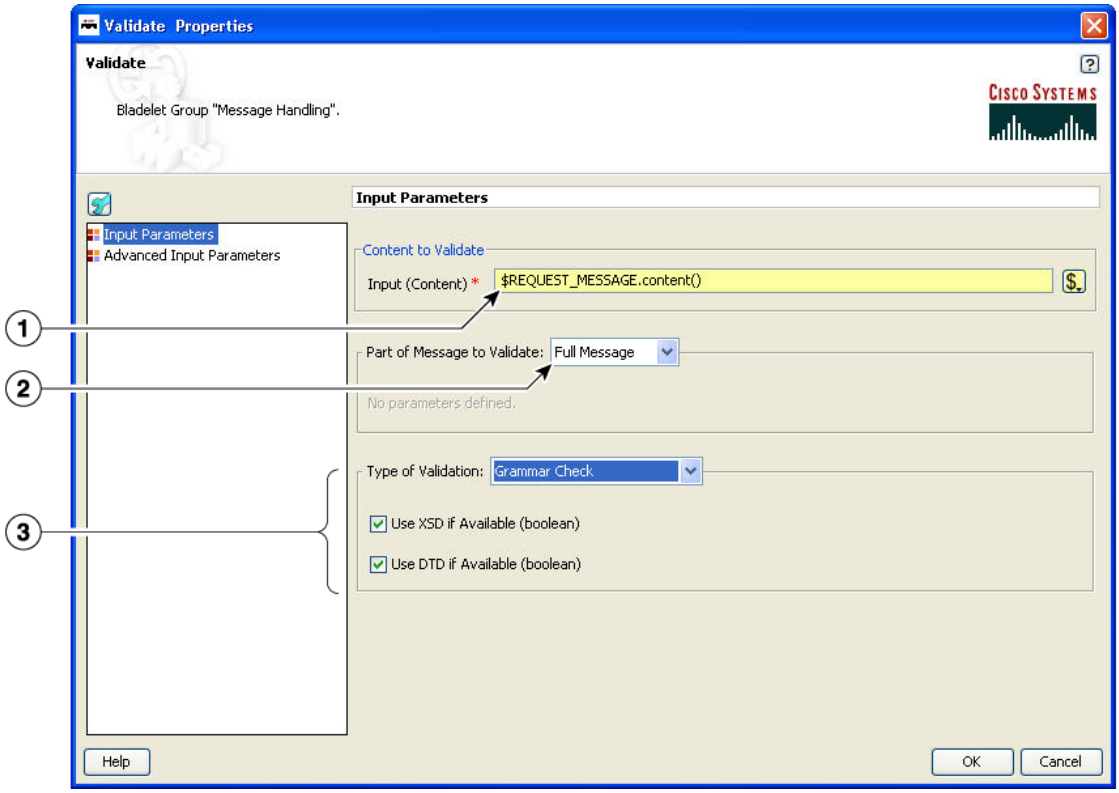
Prerequisites and Dependencies

- 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

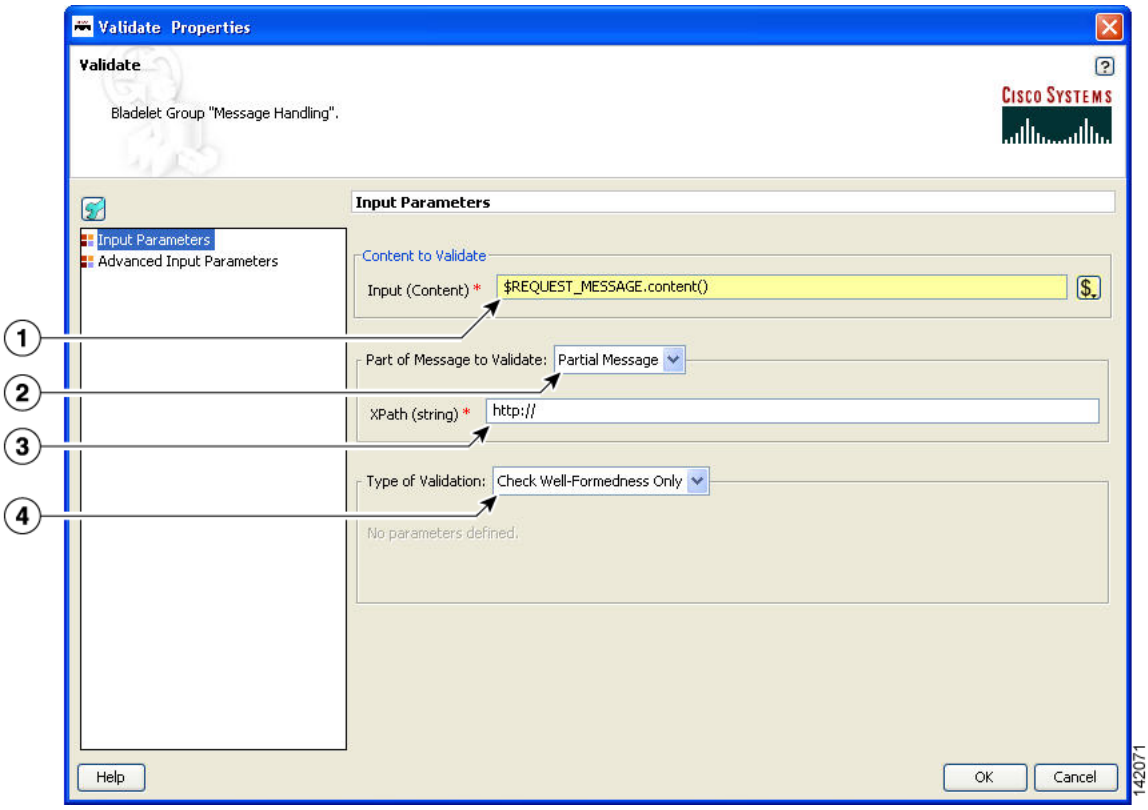
The Validate Bladelet has two main parts in its properties window: input parameters and advanced input parameters.

Figure 3-28 Validate Properties Window—Input Parameters, Validate



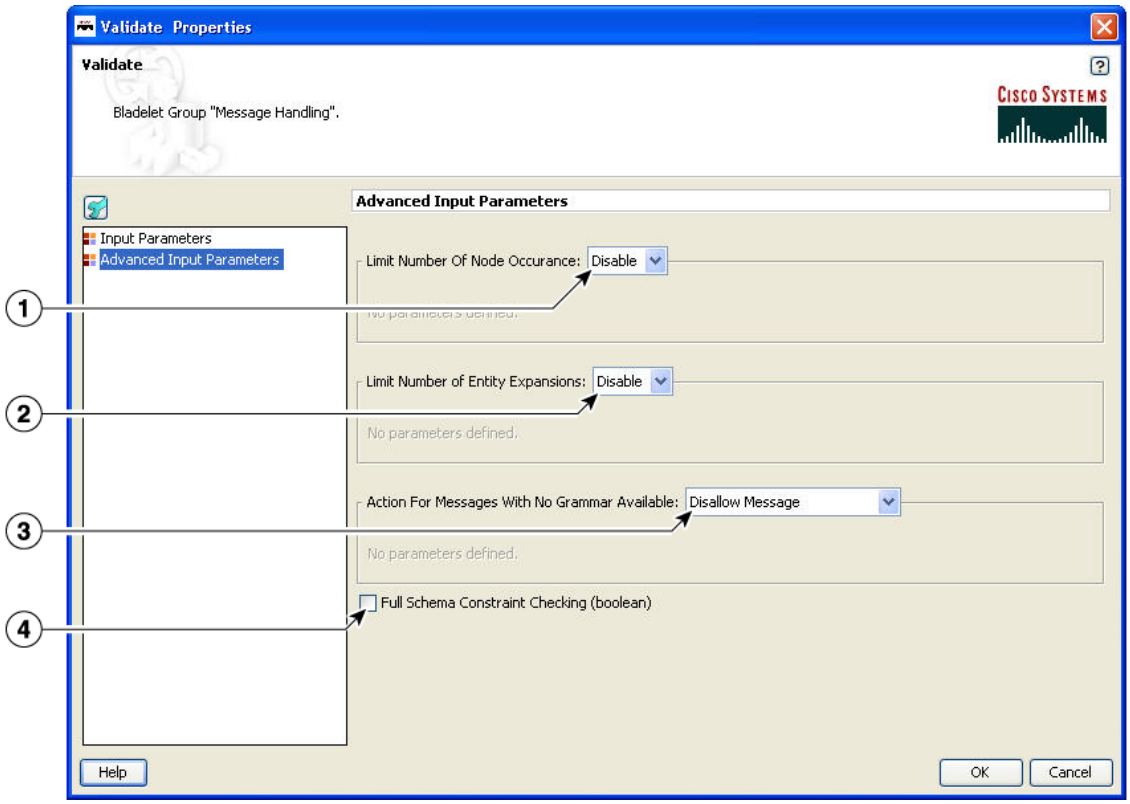
1	Input Content	Source-content input. XML message content to be validated by the Bladelet.
2	Part of Message to Validate	Full Message—Whole XML message needs to be validated.
3	Type of validation	Grammar Check—Whether or not to validate XSD in addition to DTD. Validate the input XML message against XSD, if the box is checked and DTD, if the box is checked. Check the box 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.

Figure 3-29 Validate Properties Window— Input Parameters, Check Well-Formedness Only



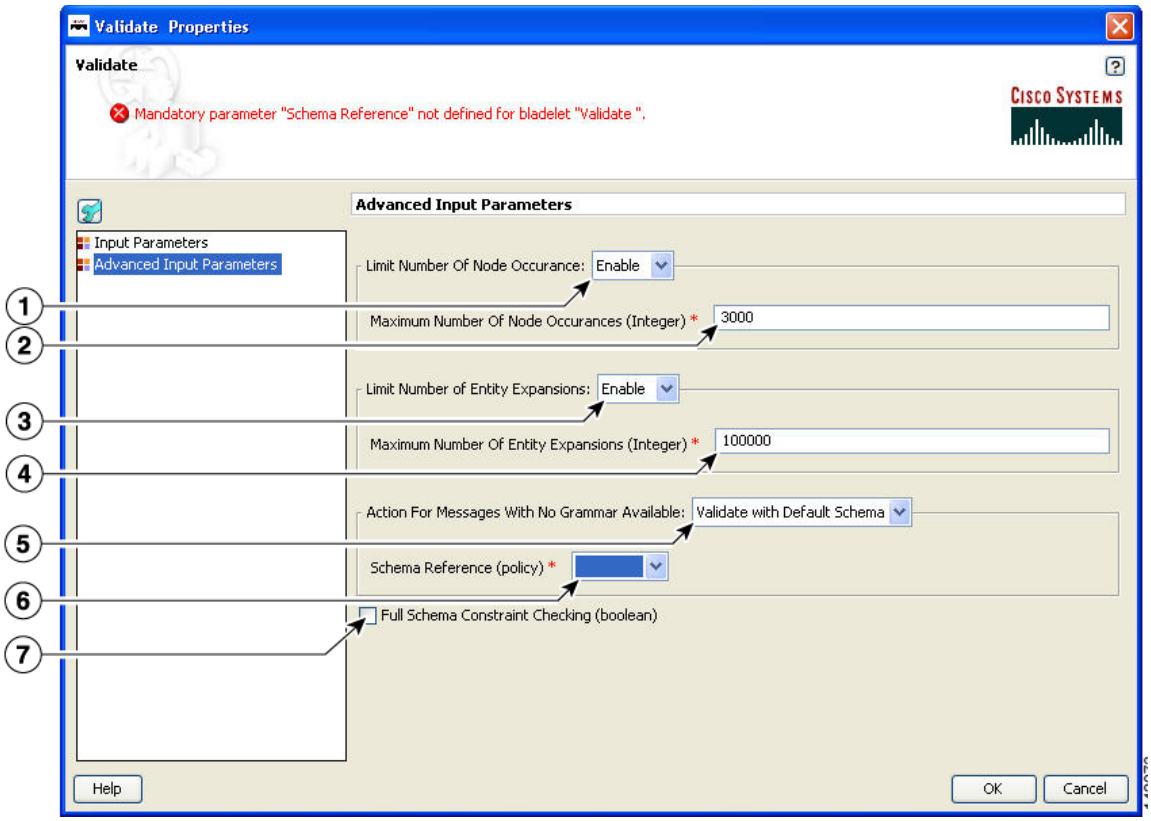
1	Input Content	Source-content input. XML message content to be validated by the Bladelet.
2	Part of Message to Validate	Partial Message—Only part of the input XML message is validated. This is determined by the XPath value entered in the XPath input field.
3	XPath String	XPath value.
4	Type of Validation	Check Well-Formedness Only—Whether or not to ensure the input XML message is formed according to XML standards.

Figure 3-30 Validate Properties Window—Advanced Input Parameters 1



1	Limit Number of Node Occurrence	Disable—No limit is set on the number of content model nodes in the XML message.
2	Limit Number of Entity Expansions	Disable—No limit is set on the number of entity expansions and parser can permit any number of entity expansions in the XML document.
3	Action For Messages With No Grammar Available	When the input XML message does not refer to any Schema or DTD to validate against. Disallow Message—Fail the validation and set the failure path in the PEP execution.
4	Full Schema Constraint Checking	(Optional) Determine if the schemas must be checked for well-formedness.

Figure 3-31 Validate Properties Window—Advanced Input Parameters 2



1	Limit Number of Node Occurrence	Enable—Set the limit on the number of content model nodes in the XML message. The limit is configured in the “Maximum Number of Node Occurrences” input field.
2	Maximum Number of Node Occurrences	The limit of the maximum number of node occurrences.
3	Limit Number of Entity Expansions	Enable—Set the limit on number of entity expansions that the parser should permit in a XML document. The limit is configured in the “Maximum Number of Entity Expansions” input field.
4	Maximum Number of Entity Expanses	The limit of the maximum number of entity expansions.
5	Action For Messages With No Grammar Available	When the input XML message doesn't refer to any Schema or DTD to validate against. Validate with Default Schema—Validate the input XML message with the default schema defined by the Schema Reference (Policy) input field.
6	Schema Reference Policy	Schema reference policy. Must already be set on the AMC server.
7	Full Schema Constraint Checking	(Optional) Determine if the schemas must be checked for well-formedness.

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

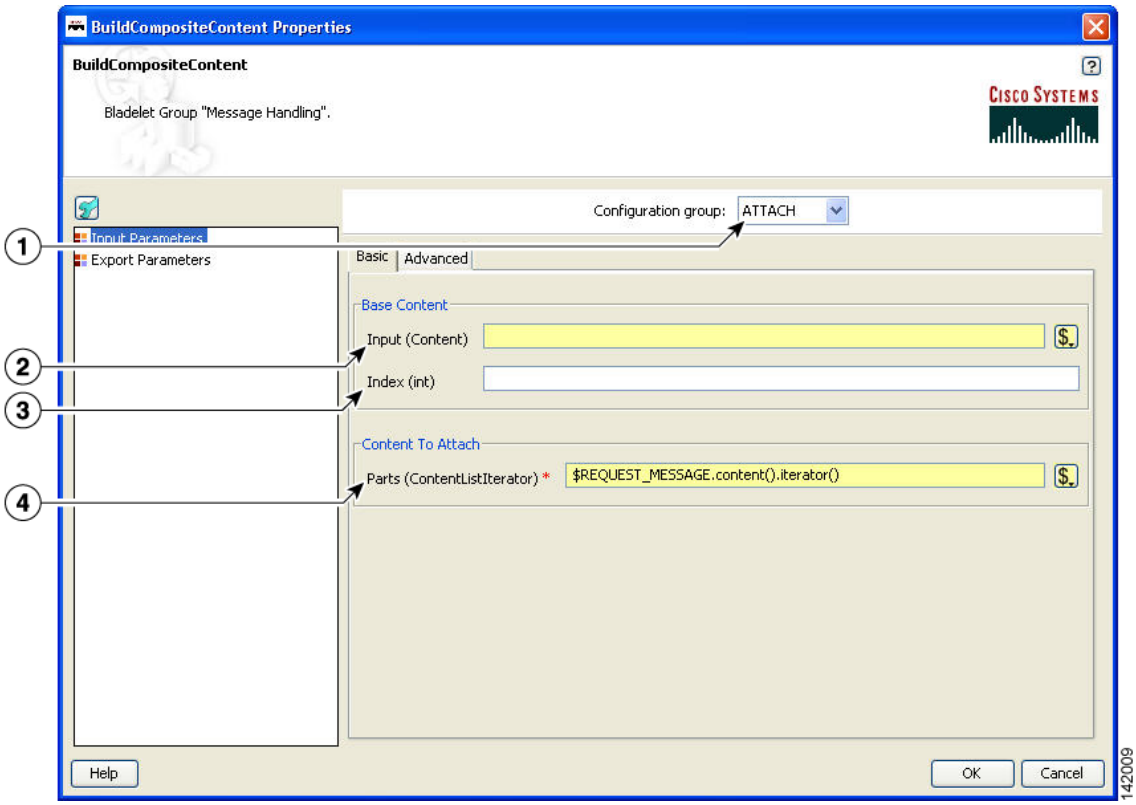
None.

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

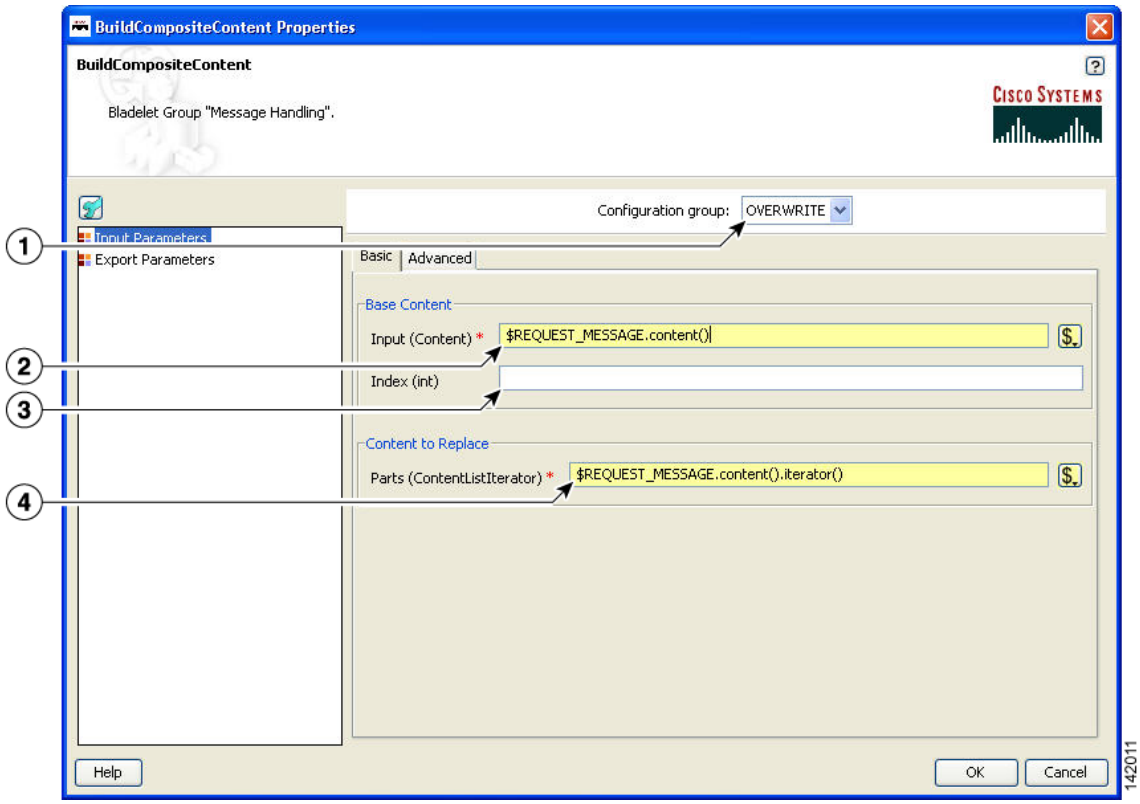
configuration group Overwrite, it overwrites the Input Content based on the Content-Id of the parts. In configuration group Delete, index and parts are mutually exclusive. Both cannot be specified. If the index is blank, the parts are deleted based on the Content-Id.

Figure 3-32 Build Composite Content Properties Window—Input Parameters, Attach



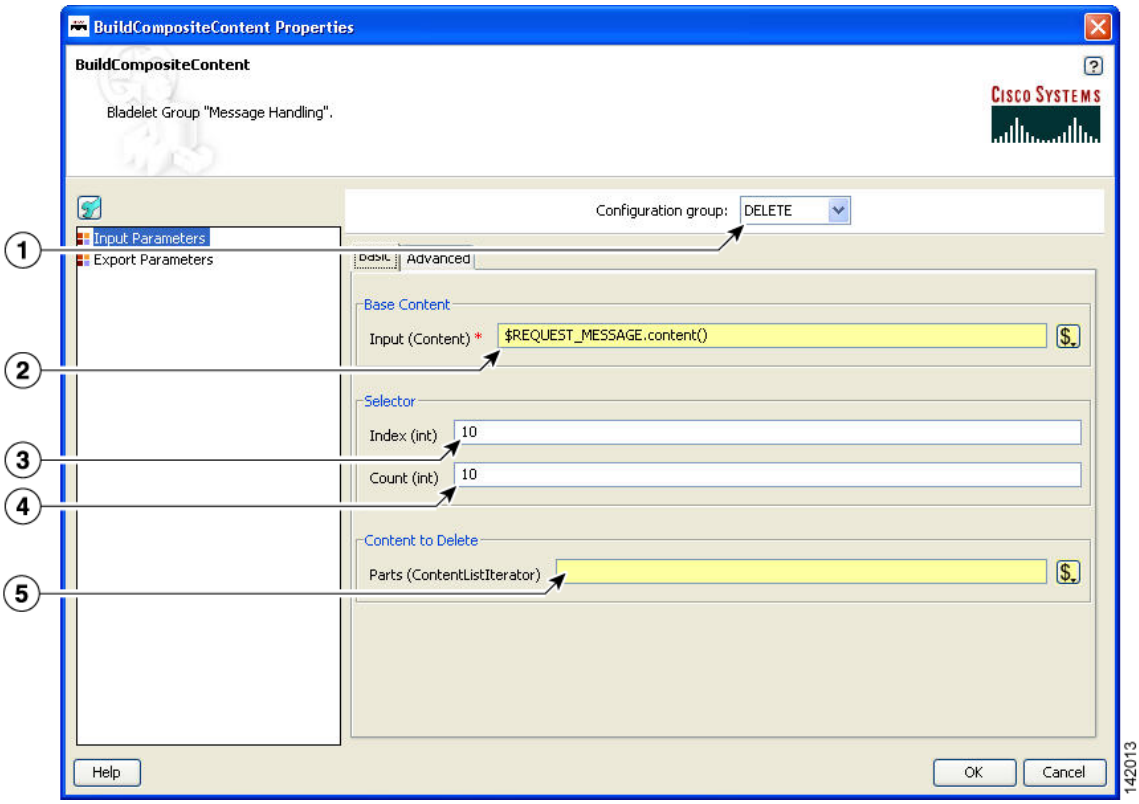
1	Configuration Group	Configuration group, set here to Attach.
2	Input	Base-content input message. Base content to which parts are attached and it has to be a multipart.
3	Index	Optional. Index to attach. If blank, attaches to the end.
4	Parts	List of contents to attach.

Figure 3-33 Build Composite Content Properties Window—Input Parameters, Overwrite



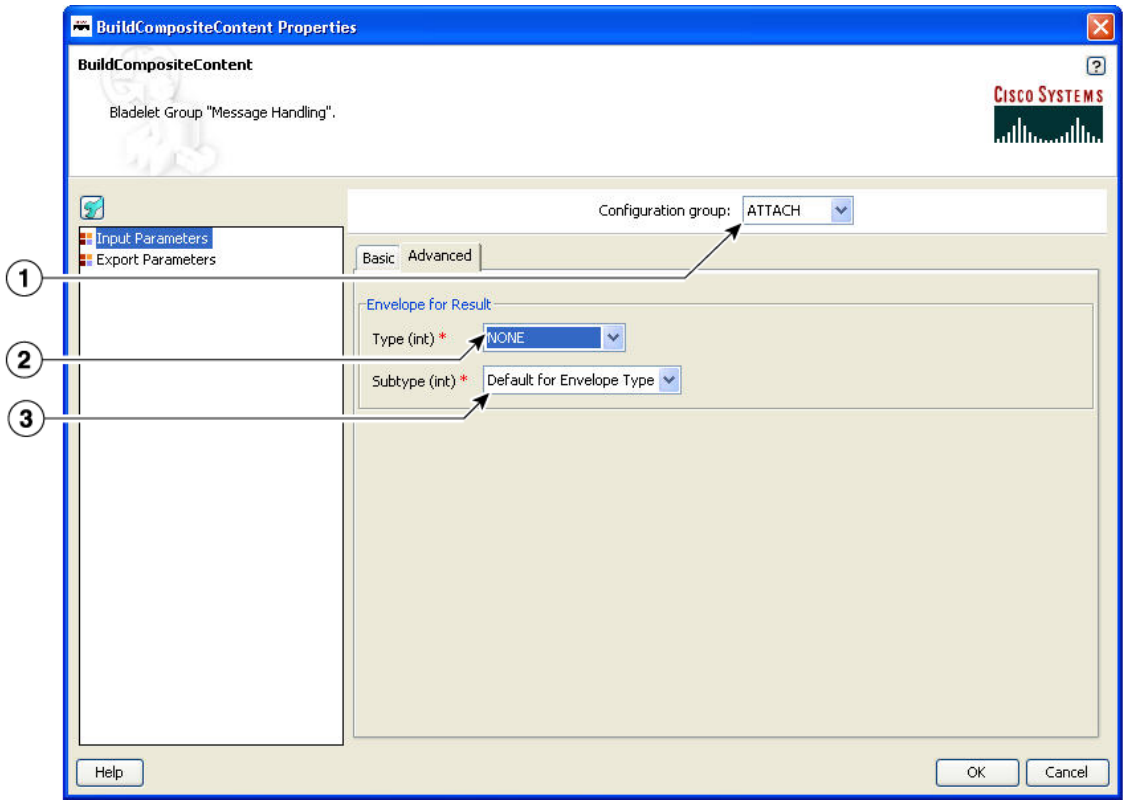
1	Configuration Group	Configuration group, set here to Overwrite.
2	Input	Base-content input message. Base content to which parts are overwritten. Must be a multipart content.
3	Index	Optional. Index to overwrite.
4	Parts	List of contents to Overwrite. Use to overwrite existing contents at the index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

Figure 3-34 Build Composite Content Properties Window—Input Parameters, Delete



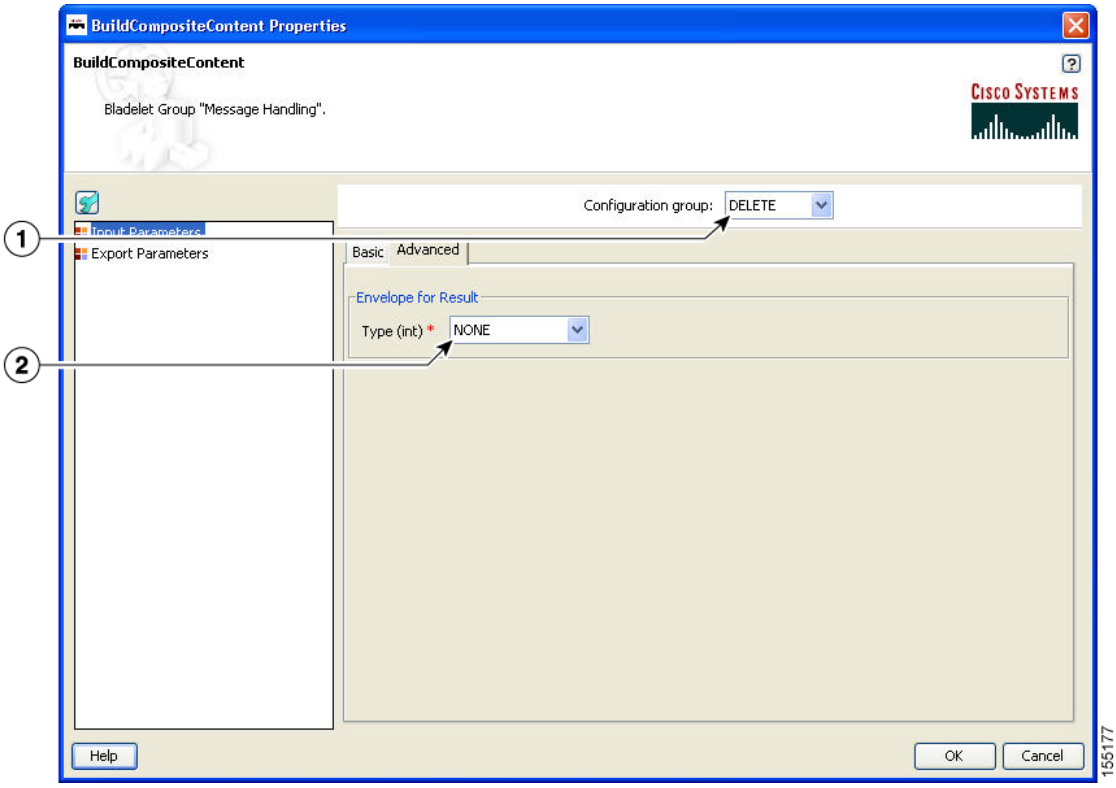
1	Configuration Group	Configuration group, set here to Delete.
2	Input	Base-content input message. Base content to which parts are overwritten. Must be a multipart content.
3	Index	Optional. Index to overwrite.
4	Count	Number of parts that need to be deleted from the index specified.
5	Parts	List of 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 3-35 Build Composite Content Properties Window—Input Parameters, Advanced, Attach



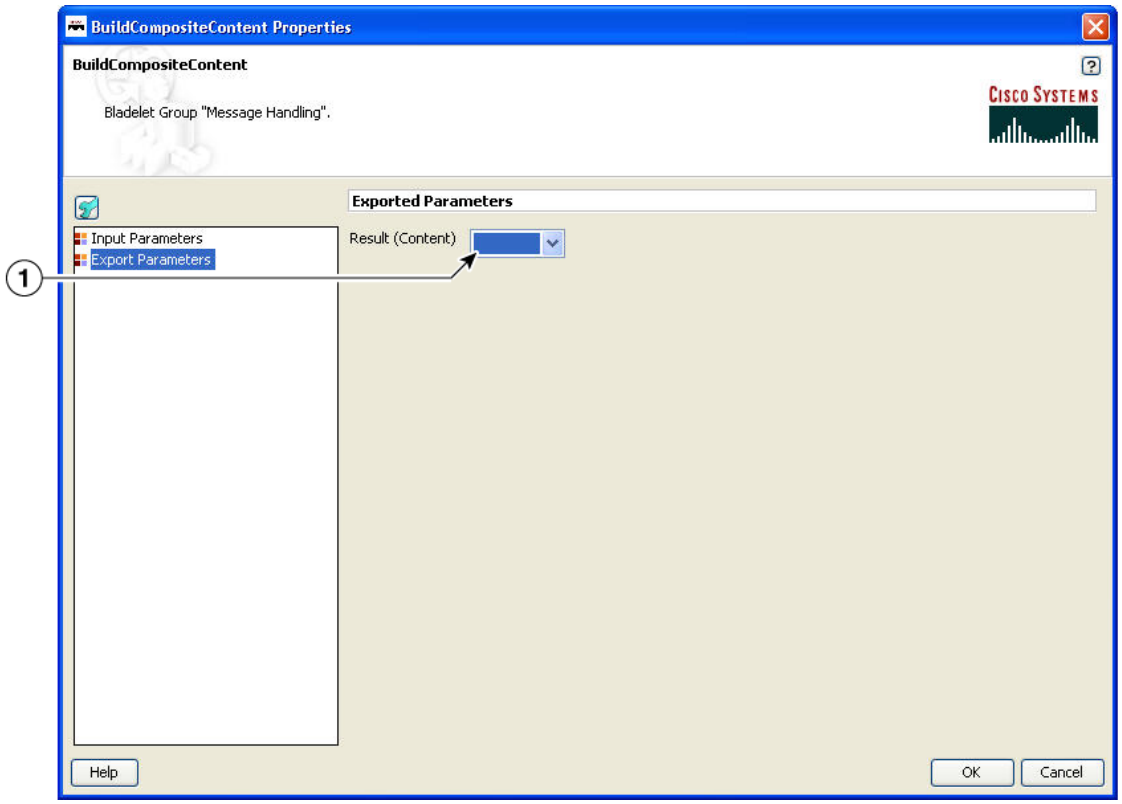
1	Configuration Group	Configuration group, set here to Attach.
2	Type	Output-message type. Default is None, which is the same as a regular MIME message. RosettaNet outputs the message in RosettaNet format.
3	Subtype	Header for subtypes when the input content is null. Can be set only when Configuration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

Figure 3-36 Build Composite Content Properties Window - Input Parameters, Advanced, Overwrite/Delete



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 3-37 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 to set 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.

Send Reply

Summary

The Send Reply Bladelet sends a reply from the PEP before the end of the flow is reached.

Prerequisites and Dependencies

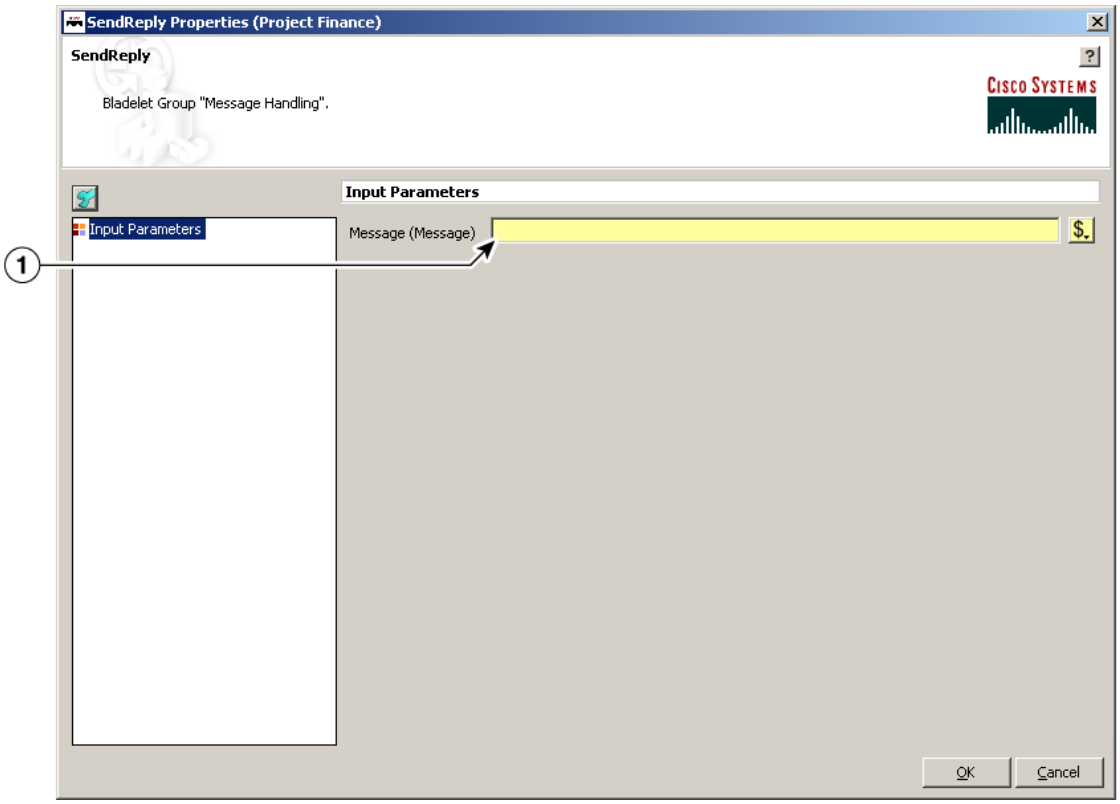
There should be a Send or a Create Response preceding this bladelet so that a response message is available to be sent out.

Details

The Bladelet takes the message to be sent as reply as input. The exception SendReplyFailed is generated if sending the reply out fails. If the Bladelet is used in a one-way PEP, a generic bladelet exception is given. If two Send reply bladelets are used in the same path of a two-way PEP, the second bladelet is ignored and a notice-level log message is printed.

In the normal usage scenario, the bladelet would cause the response message to be sent back to the client. Once the response is sent out, the PEP starts executing again starting with the bladelet after the send reply bladelet. If no Send Reply bladelet is added to the PEP path, then the response is sent back to the client at the end of the PEP (default behavior).

Figure 3-38 Send Reply Properties Window



1	Input send message	The message to be sent as a reply from the PEP.

Exceptions

SendReplyFailed exception is generated if there is a failure in sending the reply out.

Create Message

Summary

This Bladelet creates a message within 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.

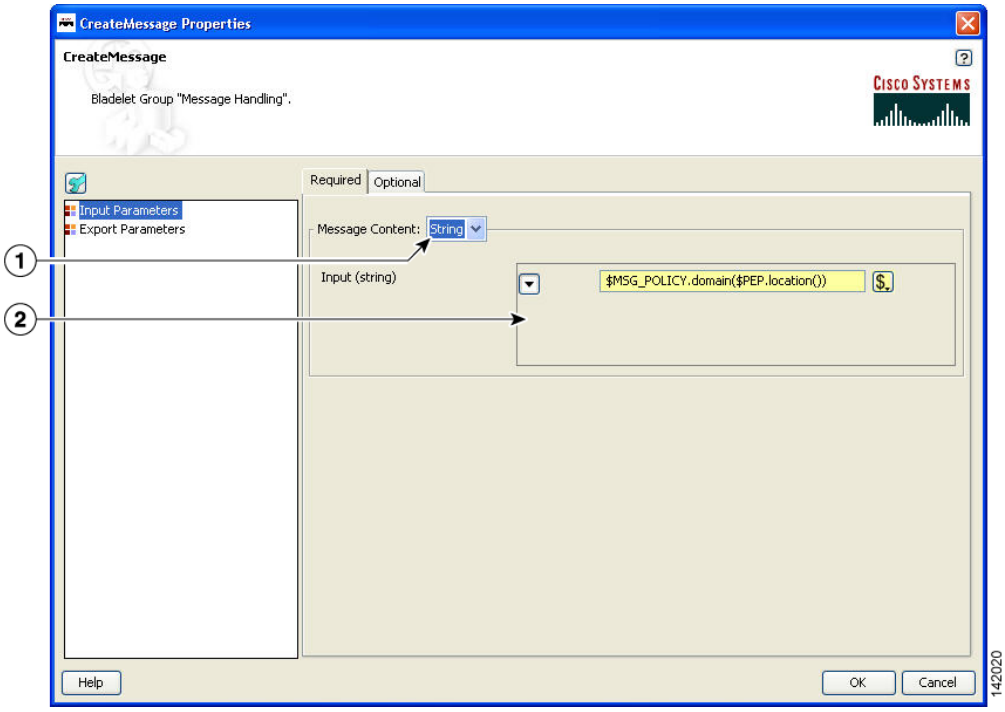
Prerequisites and Dependencies

None.

Details

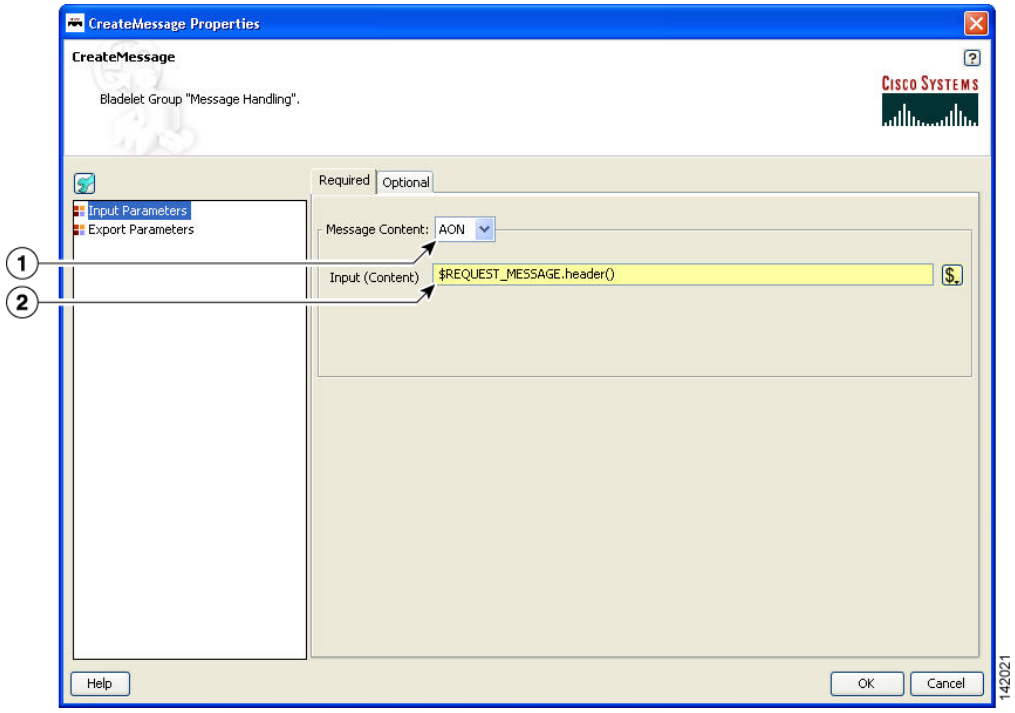
In the Create Message properties window under the Input Parameters section, tabs show required (Figure 3-39) and optional (Figure 3-41) settings.

Figure 3-39 Create Message Properties Window—Input Parameters, Required 1



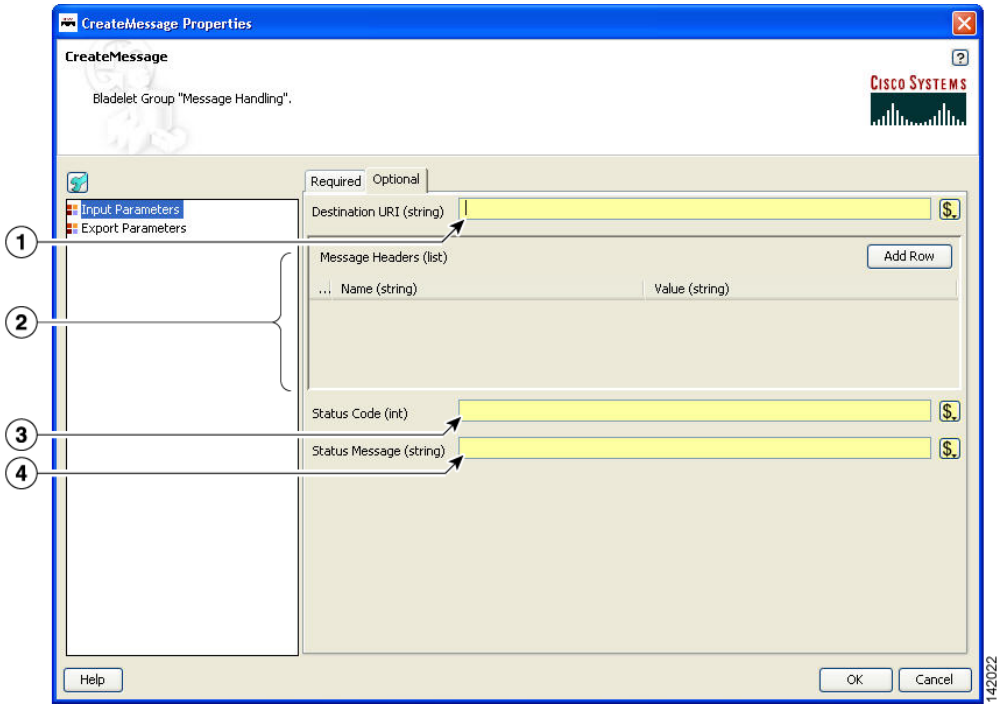
1	Message Content	Message content (string type). Required.
2	Input	Input content mentioned in 2 above.

Figure 3-40 Create Message Properties Window—Input Parameters, Required 2



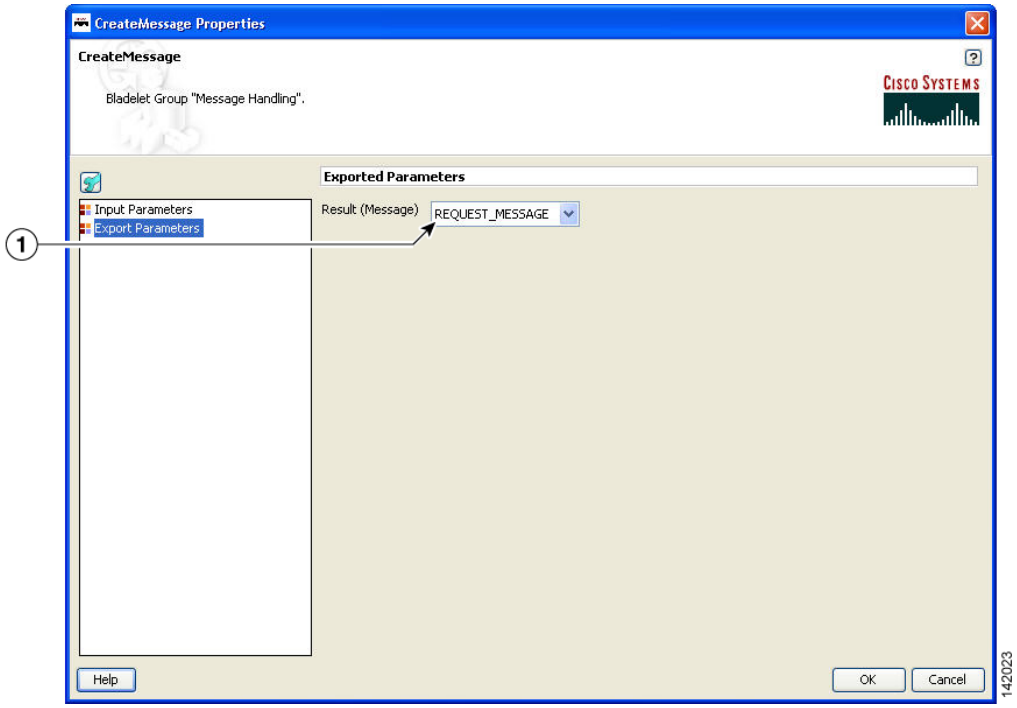
1	AON	AON content is created using the bytes in the input. Set as the message payload. Required.
2	Input	Input content to be set as the content of the message being created.

Figure 3-41 Create Message Properties Window—Input Parameters, Optional



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	Message Headers	Optional. One or more headers of the created message. Add rows as needed and enter a header name and value (string types).
3	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.
4	Status Message	Optional. Status message string.

Figure 3-42 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

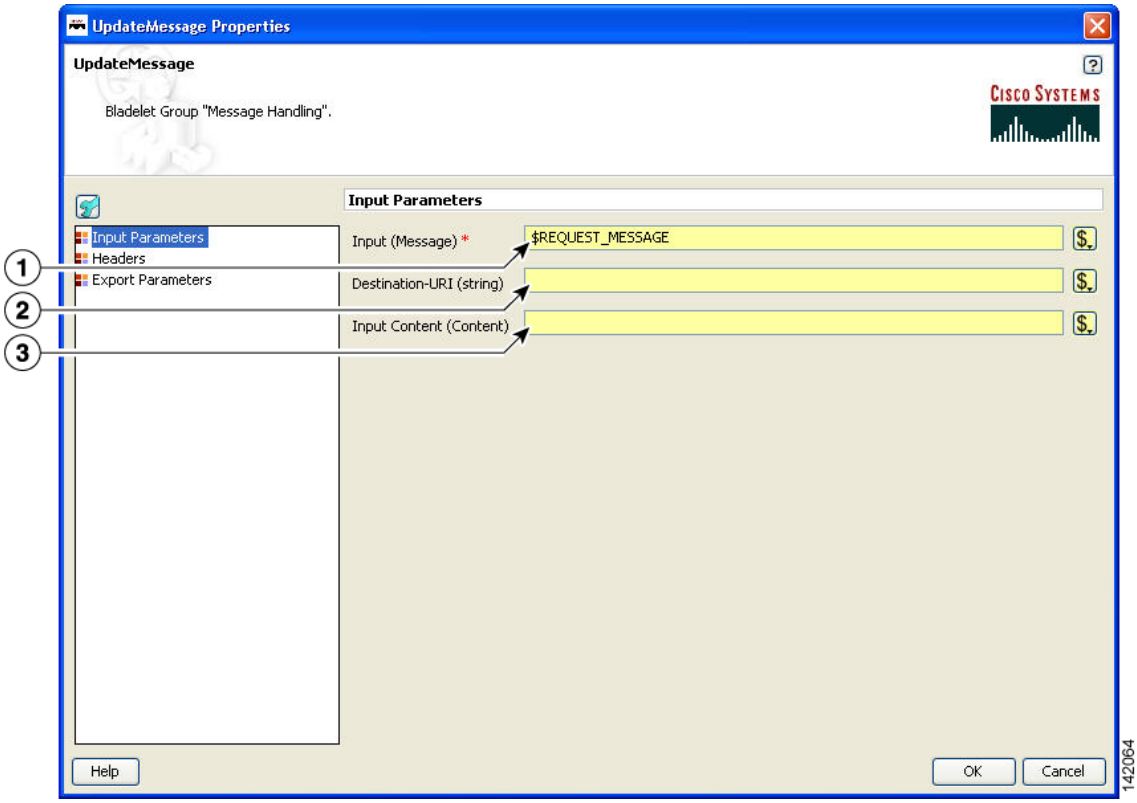
The UpdateMessage Bladelet updates an existing AON message in the PEP. User can optionally update the destination, content or the headers of the message. You can use this Bladelet to update the payload of the incoming message or modify some header information as it forwards on to an endpoint or to the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

Prerequisites and Dependencies

None.

Details

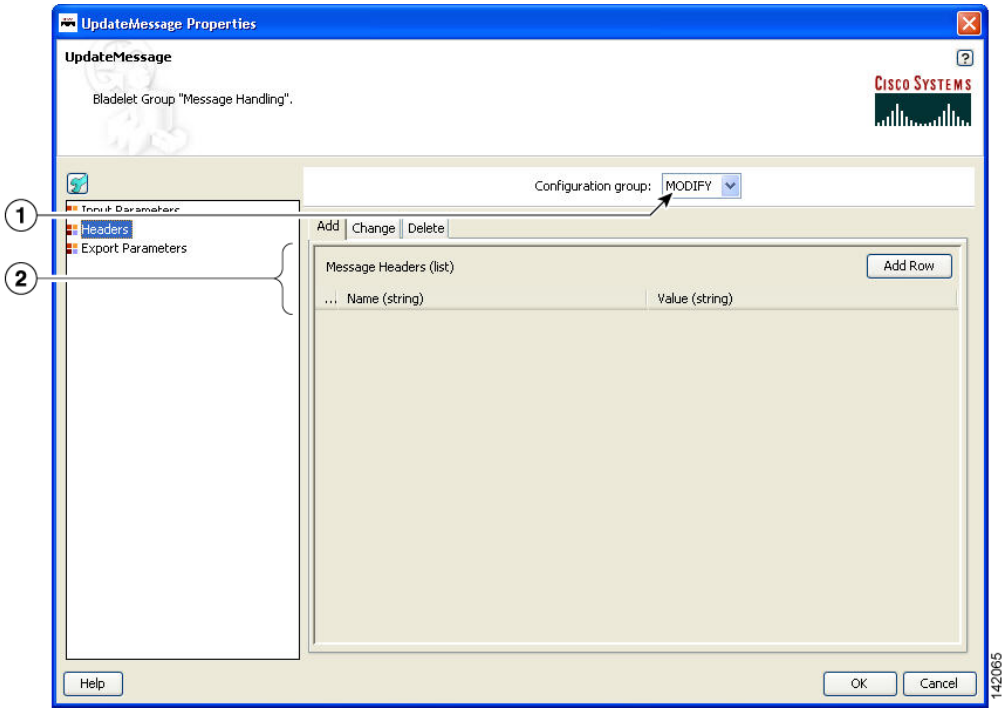
Figure 3-43 Update Message Properties—Input Parameters



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.

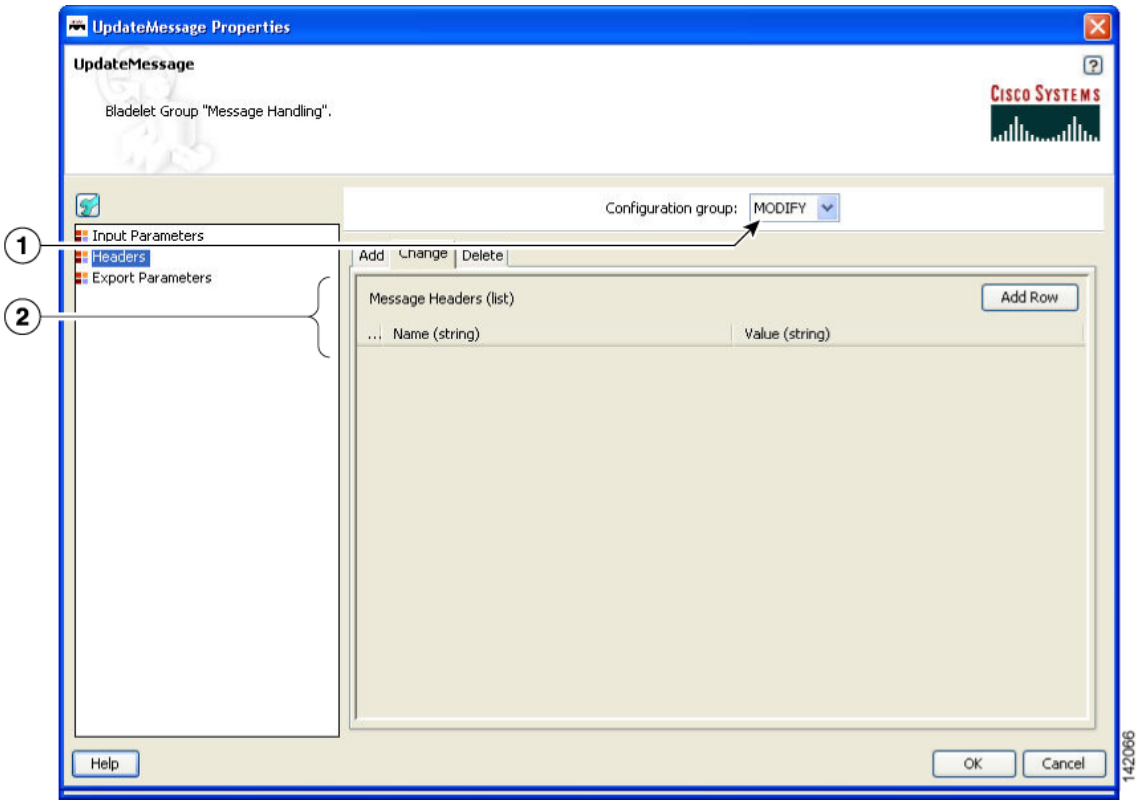
The Headers section has three tabs when the Configuration group is set to Modify (Figure 3-44 to Figure 3-46). You can set the Configuration group to Replace (Figure 3-47).

Figure 3-44 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.

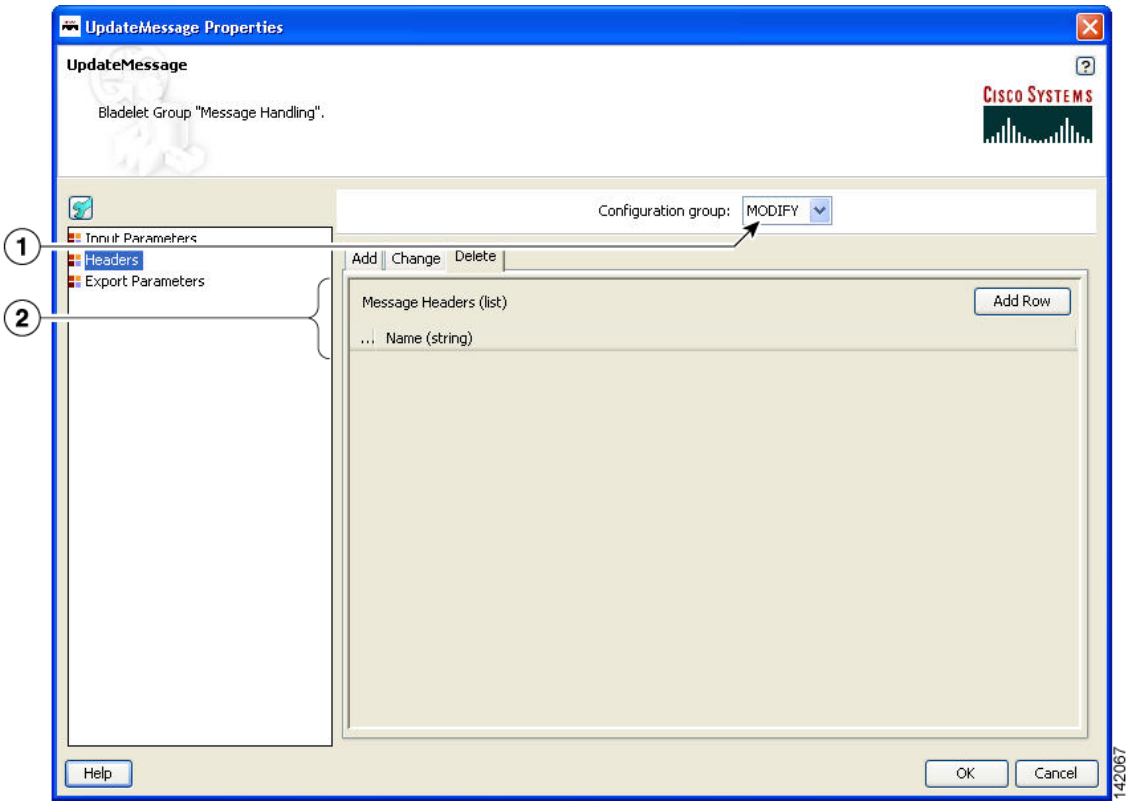
Figure 3-45 Update Message Properties Window—Headers, Modify, Change Tab



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.

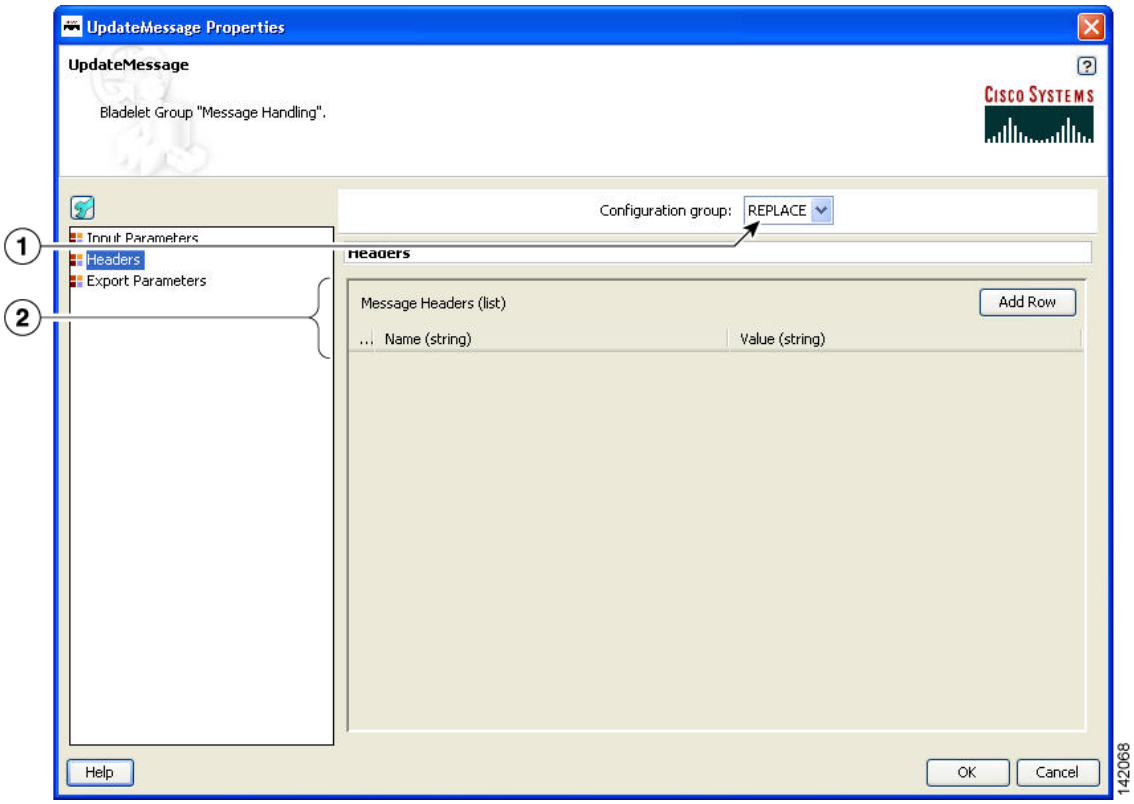
While the Configuration group is still set to Modify, delete one or more header names as necessary.

Figure 3-46 Update Message Properties Window—Headers, Modify, Delete Tab



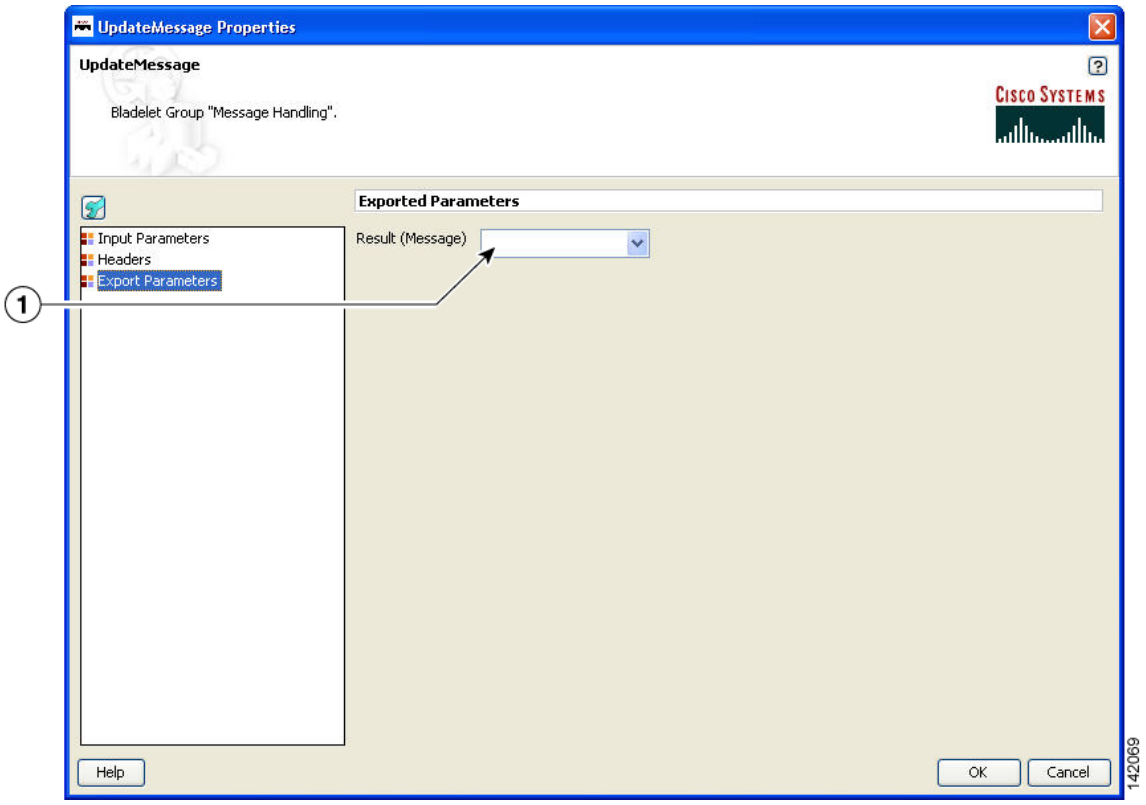
1	Modify	Configuration group, set here to Modify.
2	Delete Message Headers	Names of the headers to be deleted from the message being updated (string type). Delete as many header names as required by adding rows.

Figure 3-47 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 3-48 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.

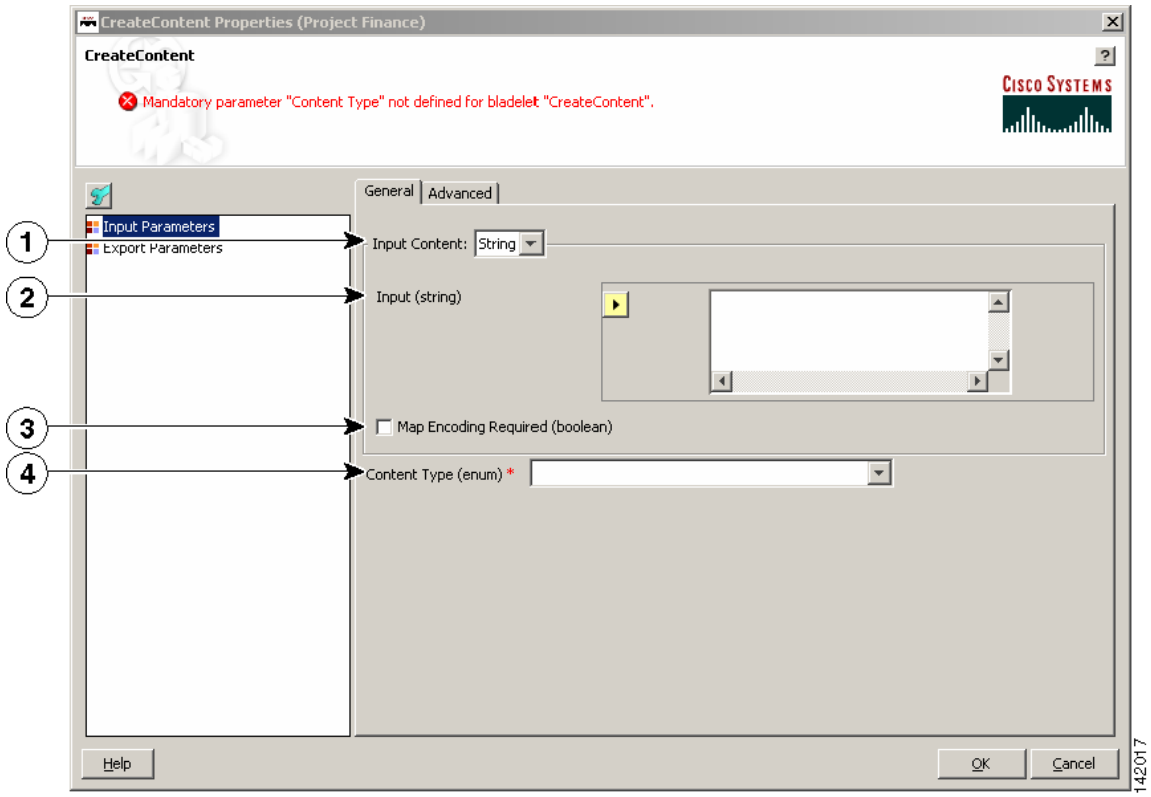
If the content type needs to be more specific than what is shown in the Required/Content-Type drop-down list, you can specify it as one of the headers. For example: Content-Type (header name) and “application/xml” (header value). The entry in the header overrides the default content type.

Prerequisites and Dependencies

None.

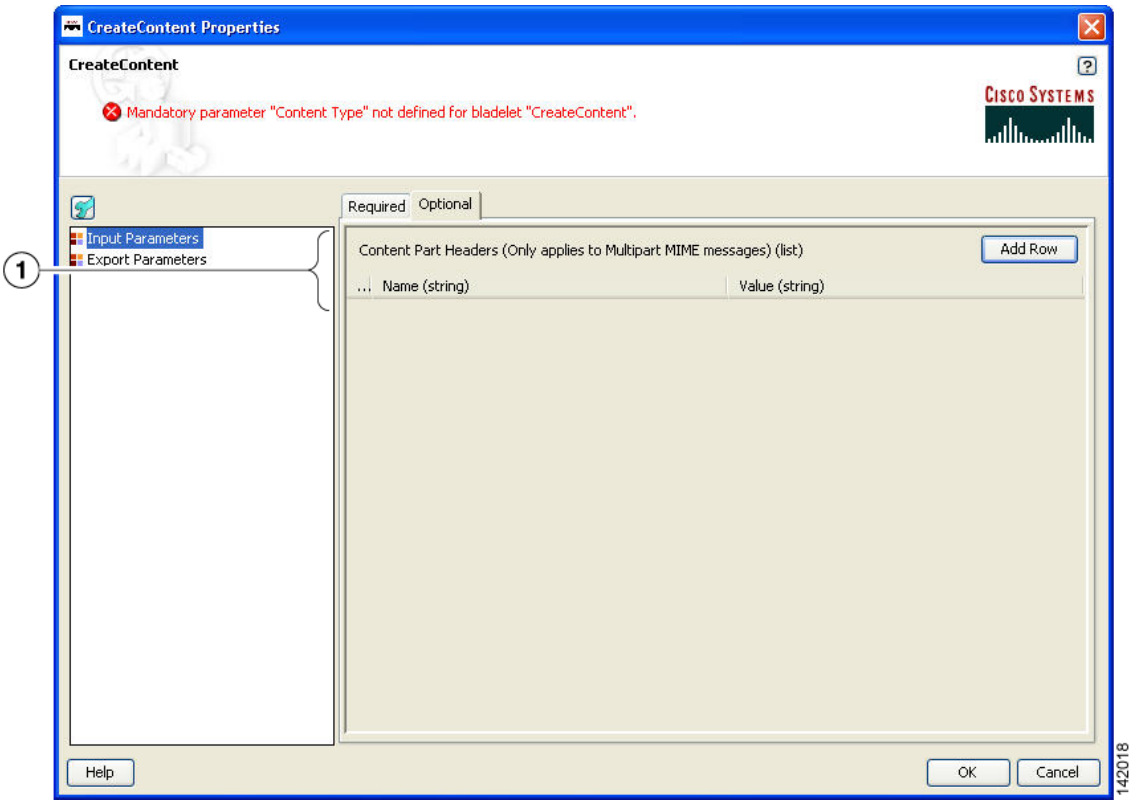
Details

Figure 3-49 Create Content Properties Window—General, Required Parameters



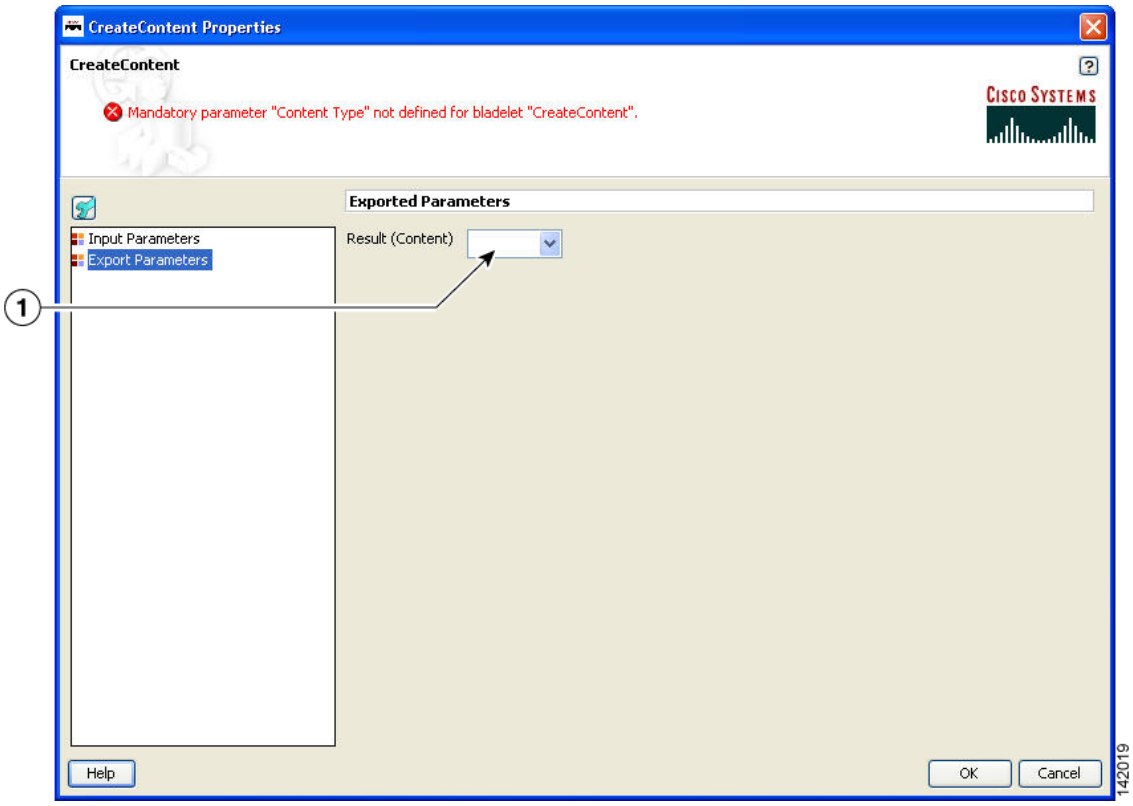
1	Input Content	Input content: • String—AON content is created from the bytes in the string. • AON content—AON content is created from the bytes in the input content. Useful to convert one type of content (say Stream) to another (say SOAP). The input content should be convertible to the desired output type.
2	Input (string)	The input string.
3	Map Encoding Required	Selecting this option requires map encoding for content creation.
4	Content Type	Content type. Choices: Stream Content, XML Content, SOAP Content, and Map Content.

Figure 3-50 Create Content Properties Window—General, Optional Parameters



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

Figure 3-51 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

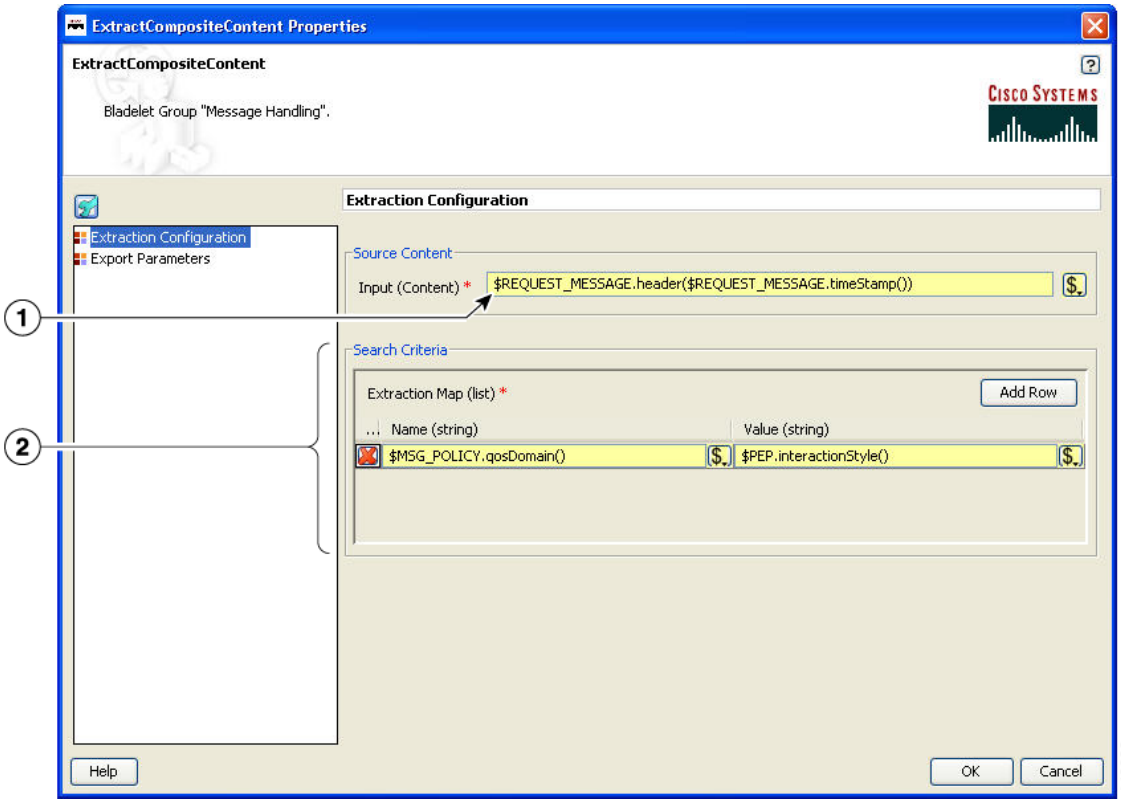
The Extract Composite Content Bladelet extracts the contents from a multipart content message.

Prerequisites and Dependencies

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

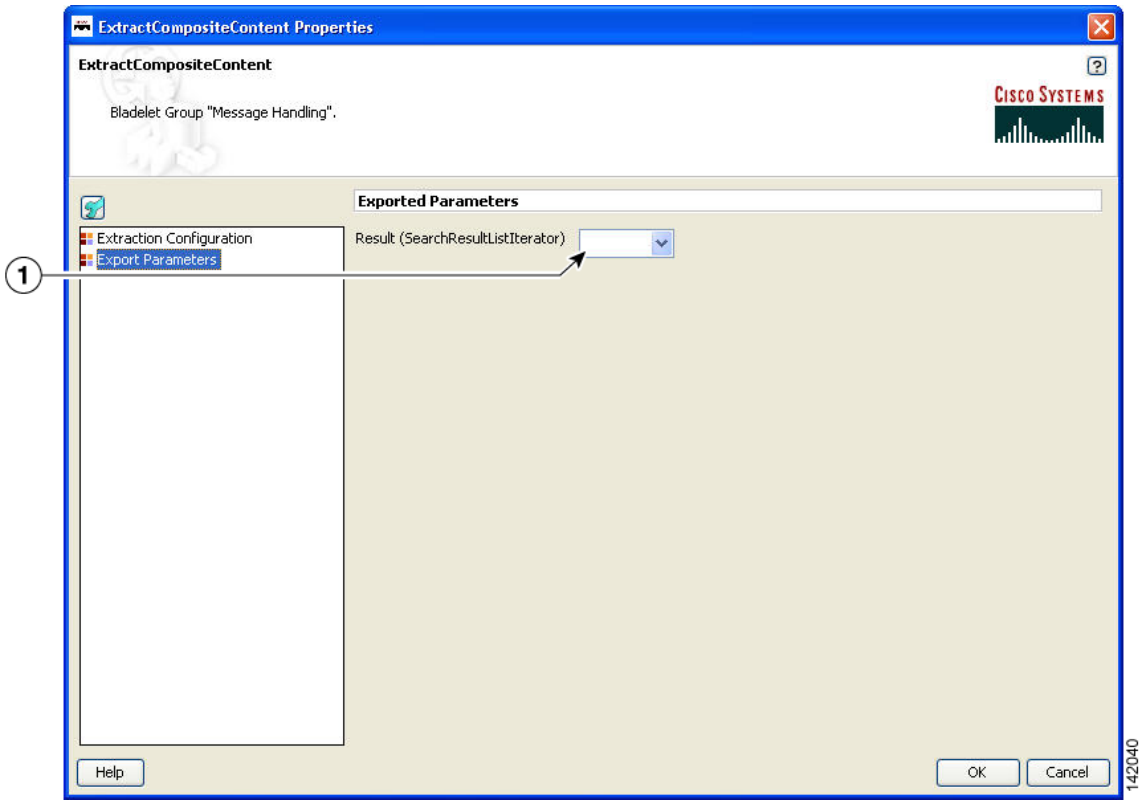
Details

Figure 3-52 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 3-53 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.

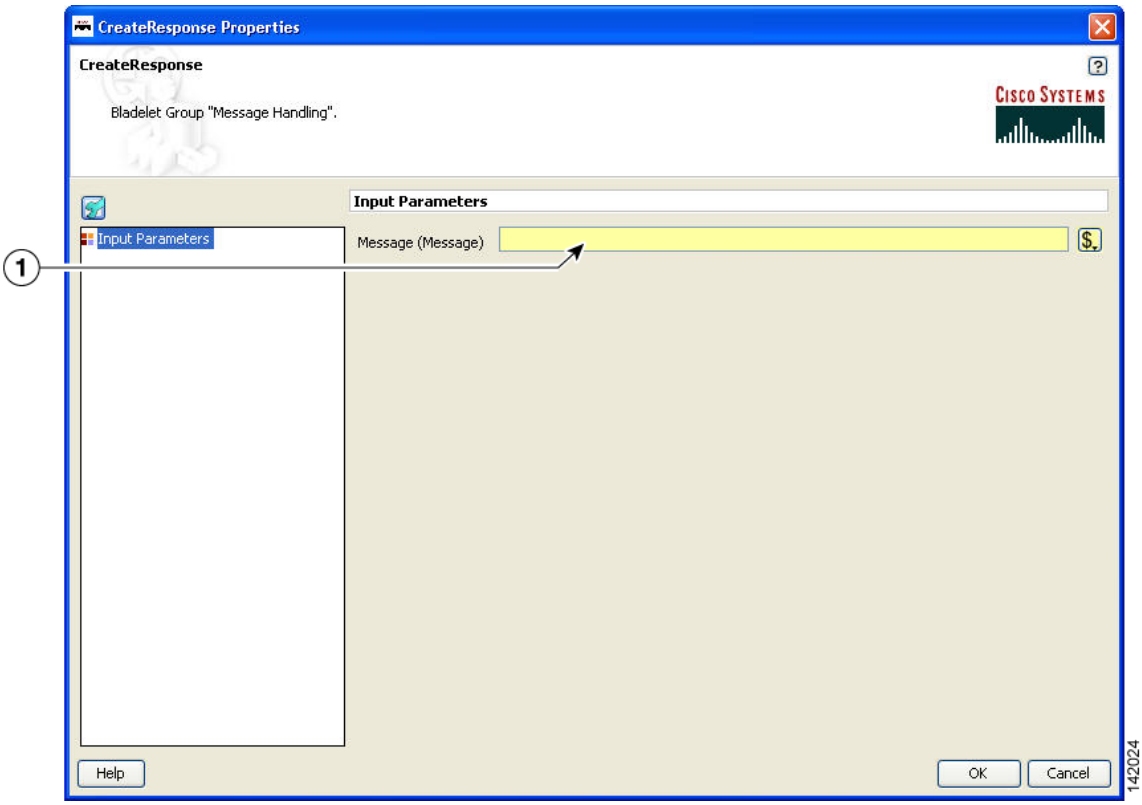
This Bladelet does not have to be used after Send or Distribute Bladelets. This Bladelet is typically used in conjunction with CreateMessage or RetrieveCache. It may also be used when the PEP has multiple Sends and based on some logic, one of the replies needs to be picked.

Prerequisites and Dependencies

None.

Details

Figure 3-54 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.

Application QoS

Summary

The Application QoS Bladelet enables the network to provide true application Quality of Service (QoS) by inspecting the message content and context. The application QoS functionality is implemented within an Application QoS Bladelet combined with other AON bladelets, such as Encrypt and Sign forms a Policy Execution Plan (PEP). The Application QoS Bladelet makes use of AON PEP variables to extract information from the message to set DSCP values.

Prerequisites and Dependencies

None.

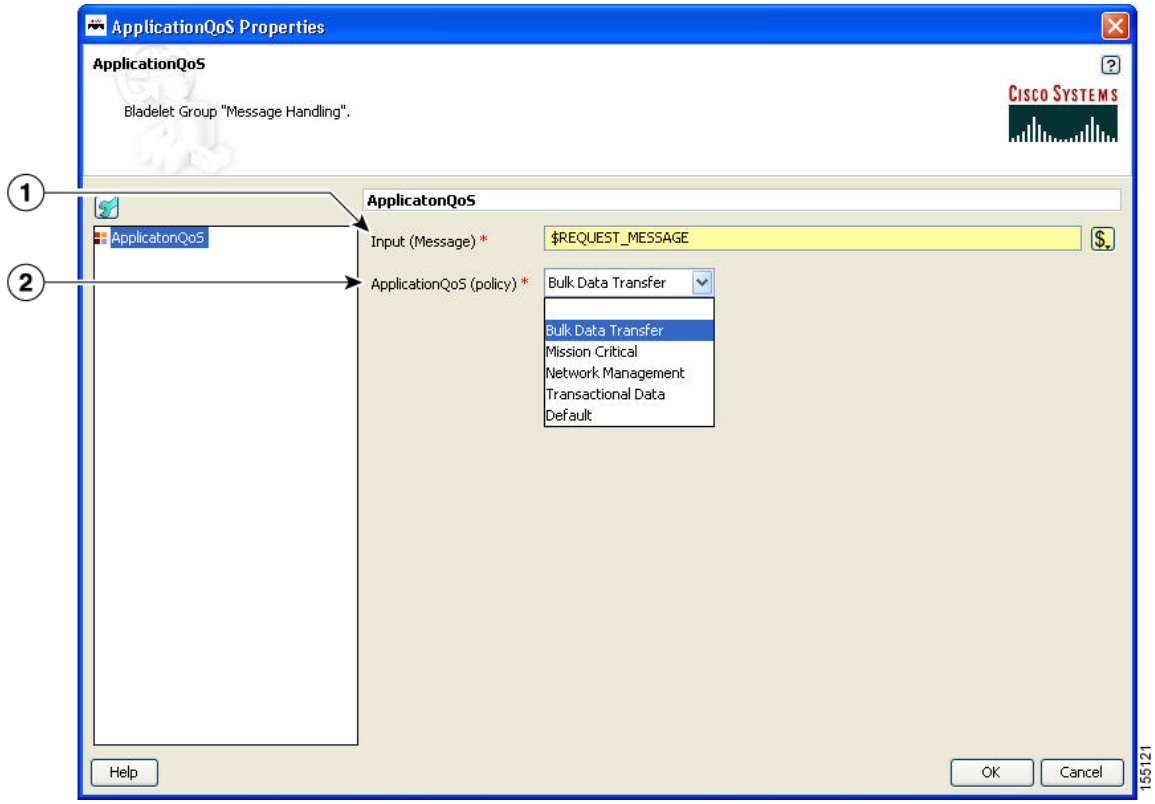
Details

- Application QoS values are assigned based on message classification. Based on classification, a message can be assigned the following Application QoS values:
  - Bulk Data Transfer—AF11 <001010>
  - Mission Critical—AF31 <011010>
  - Network Management—CS2 <010000>
  - Transactional Data—AF21 <010010>
  - Default—0 <000000>

  For classified packets, Mission Critical would have the highest priority and Bulk Data Transfer would have the lowest priority. Based on this information, it is easy to map the DSCP values (from the baseline document, default values), extending QoS beyond the L3 and L4. You can configure these DCSP mapping values and override the default as needed. Application QoS can be assigned on one AON node and be carried forward to all downstream AON nodes.
- All message processing on a given AON node is prioritized based on the Application QoS in a message, through the use of Queues and Queue Priority.
- A message that does not have any application QoS is processed with a default priority.

Note Mapping of QoS to DSCP is done in AMC and message classification in a PEP using an Application QoS Bladelet is done in ADS.

Figure 3-55 Application QoS Properties Window



1	Input Message	Application QoS Bladelet acts on this message. By default, the variable is REQUEST_MESSAGE. Application QoS can also be applied to RESPONSE_MESSAGE.
2	ApplicationQoS Policy	Policy to be applied to ApplicationQoS Bladelet: • Bulk Data transfer • Mission Critical • Network Management • Transactional Data • Default These policies can be mapped to the Application QoS value in AMC. This policy is available under Application. This must already be configured on the AMC server. The full path in AMC is **Properties > Application > Global > QoSMapping**. **Note** If you do not choose an application QoS value for an AON message then by default Application QoS is set to the DSCP value of 0 (zero).

Outcome

- Once an application QoS is assigned to a message, it is put in a prioritized Inbox queue to be picked up by the PEP management subsystem. Instead of having a single Inbox queue, a separate Inbox is used for each application QoS value that can be assigned to a message.

Exceptions

None.

Routing Category

In the Routing Category, there are four Bladelets:

- Distribute, page 3-70
- Set Destination, page 3-74
- Send, page 3-76
- Balance Load, page 3-80

Distribute

Summary

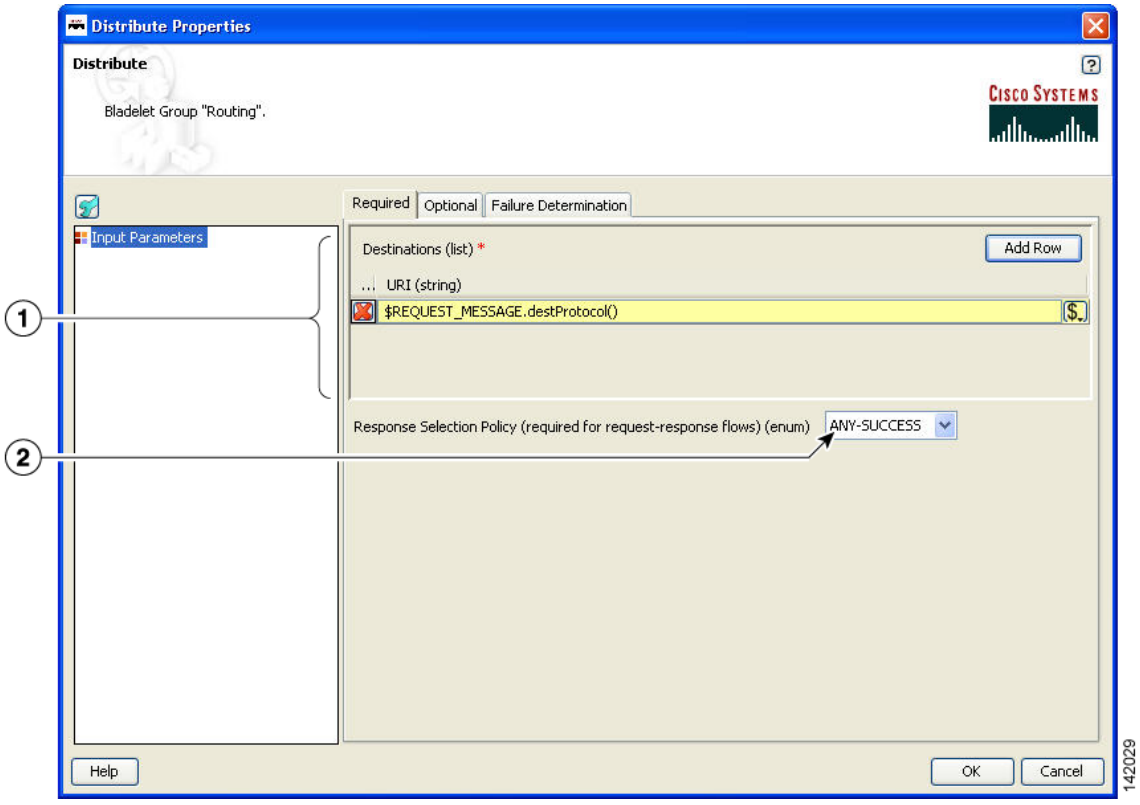
Distribute sends the same message over to multiple endpoints. In case of request-response PEPs, it gathers the responses, chooses one based on a selection criteria, and sets it as the response message. Distribute sets the response message of the PEP to the one chosen. It is a terminal Bladelet, so no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

Prerequisites and Dependencies

None.

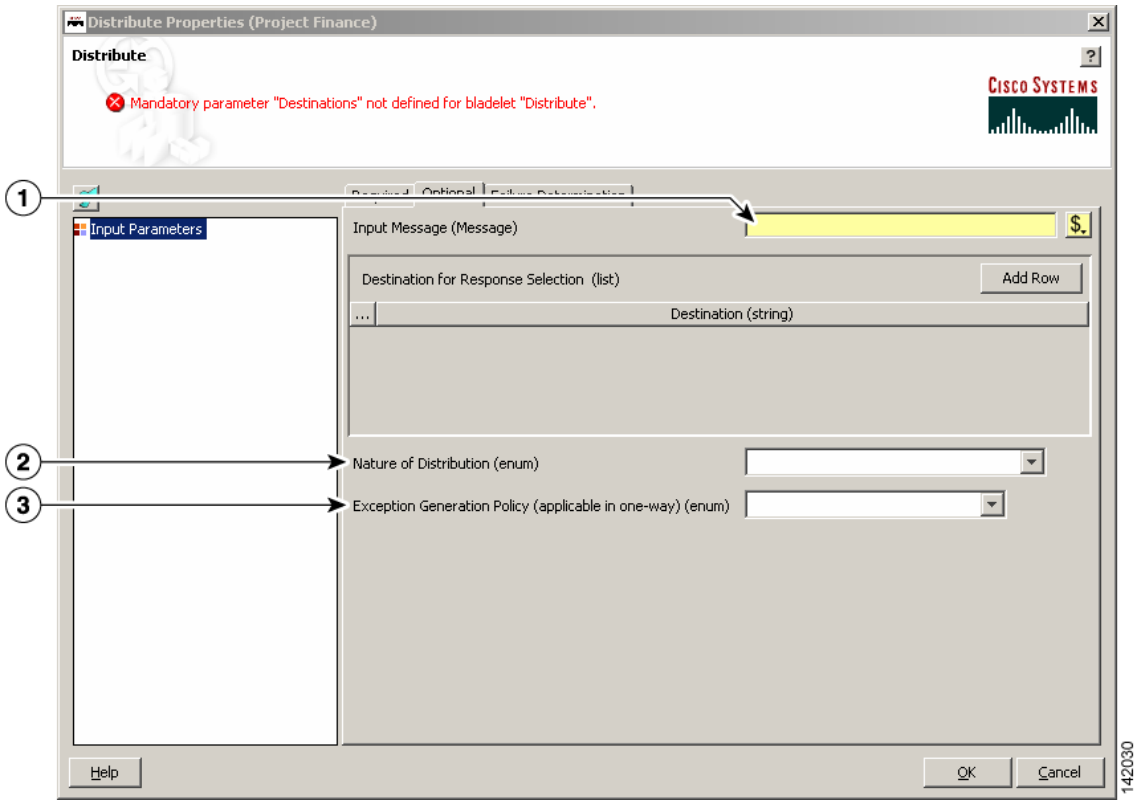
Details

Figure 3-56 Distribute Properties Window—Input Parameters, Required



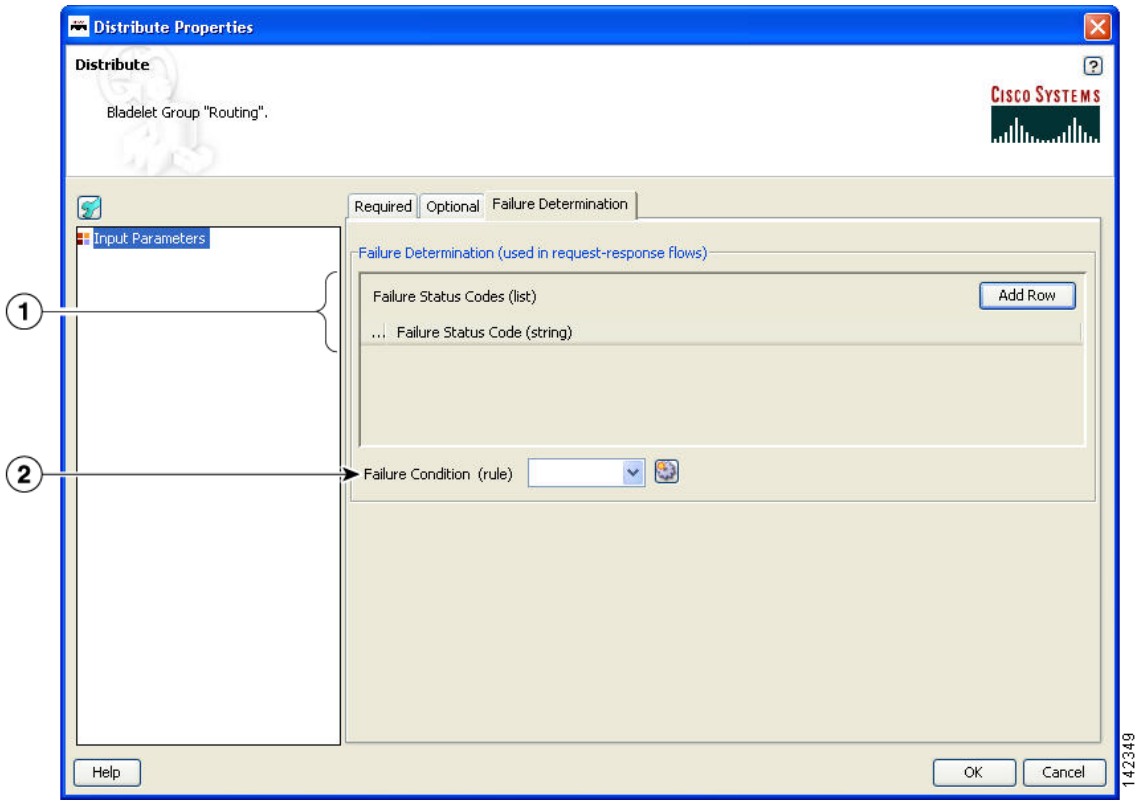
1	Destination	One or more destination URIs (string type) to which the message should be distributed. Required.
2	Response Selection Policy	Message to be considered as the response message. Required in request response PEPs. Not required for request-only PEPs. • Any-Success—First successful message to 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 3-57 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.
	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.
2	Nature of Distribution	Optional. Specifies the nature of the distribute operation. It can be one of one-way, two-way or inherit from the PEP's interaction style. The default is inherit.
3	Exception Generation Policy	Optional. Specifies when the exception is generated. The choices are: *never*; *when any send fails*; *when all sends fail*. The default is *never*. In the case of *when any send fails*, an exception is generated when any single send fails. In the case of *when all sends fail*, an exception is generated only if all the sends fail.

Figure 3-58 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

If the Distribute operation fails, a DistributionFailed exception is raised. A failure in a two-way distribute operation occurs when none of the responses are qualified under the specified Response Selection Policy. A failure in a one-way distribute operation depends on the Exception Generation Policy setting and the results of individual sends performed as part of the distribute operation. Because Distribute is terminal, you cannot continue PEP execution by connecting a bladelet to the exception path.

Set Destination

Summary

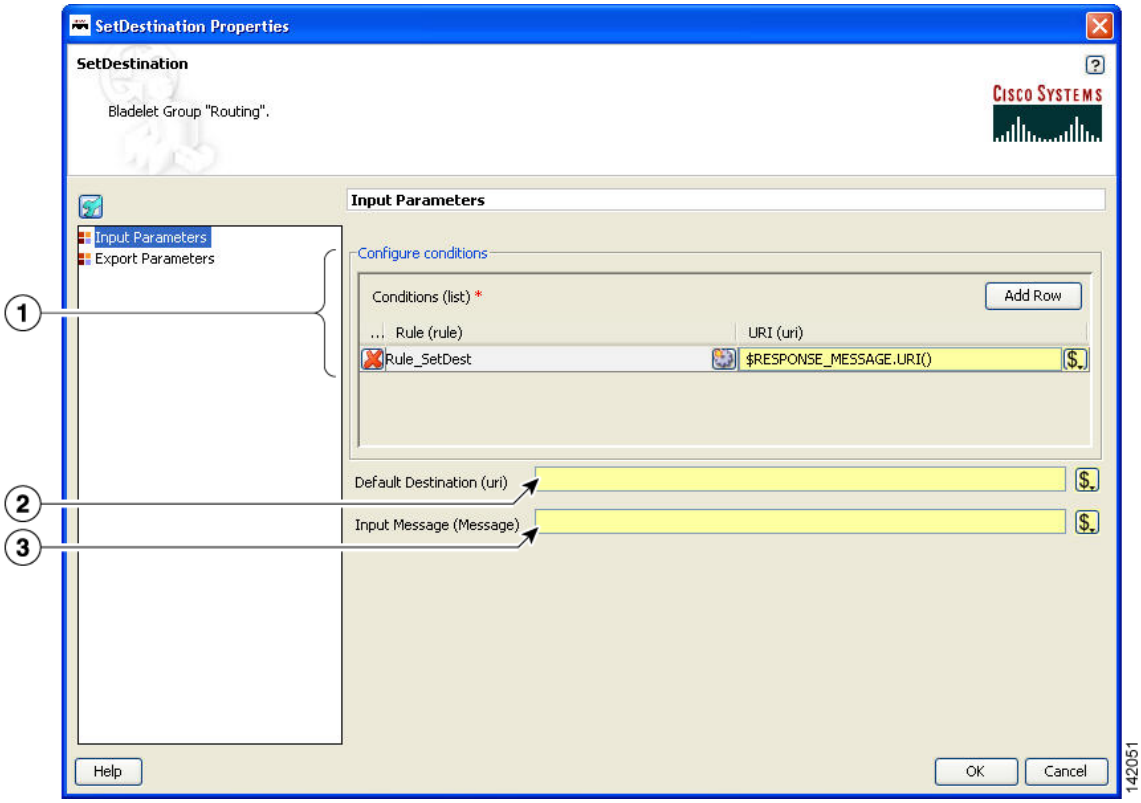
This Bladelet routes messages to destinations based on rules. It determines the final endpoint (URI) destination of the message being processed by the PEP.

Prerequisites and Dependencies

None.

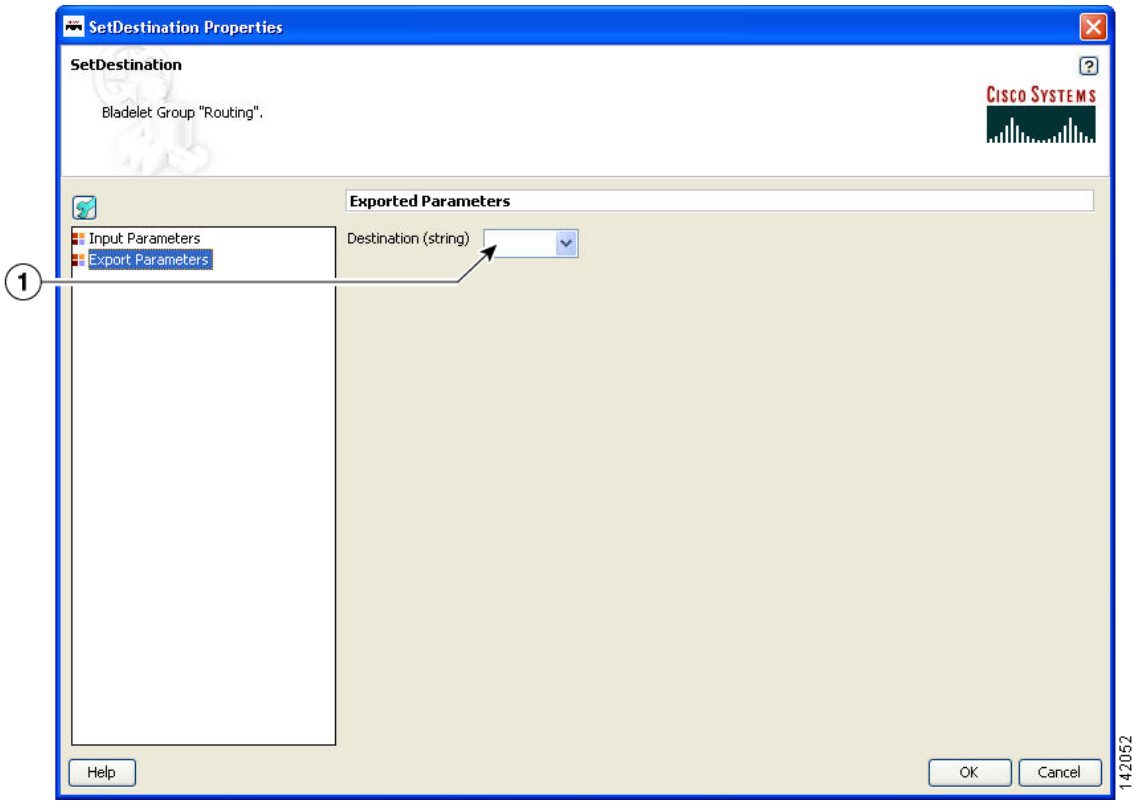
Details

Figure 3-59 Set Destination Properties Window—Input Parameters



1	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.
2	Default Destination	Destination to be used if none of the rules evaluates to true.
3	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.

Figure 3-60 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

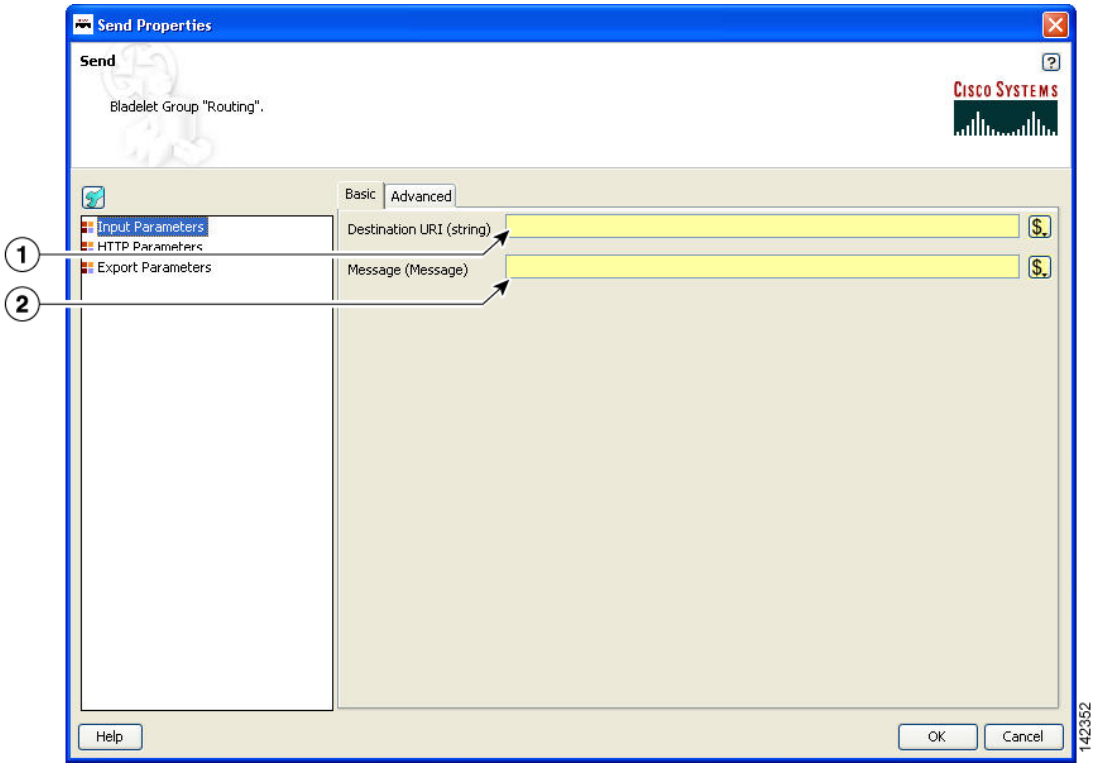
The Send Bladelet is the last item in a message request PEP and sends a message to a selected destination. The Bladelet performs protocol translation if the destination URI of the message to be sent out has to go out via an adapter that is different from the one that received the message.

Prerequisites and Dependencies

None.

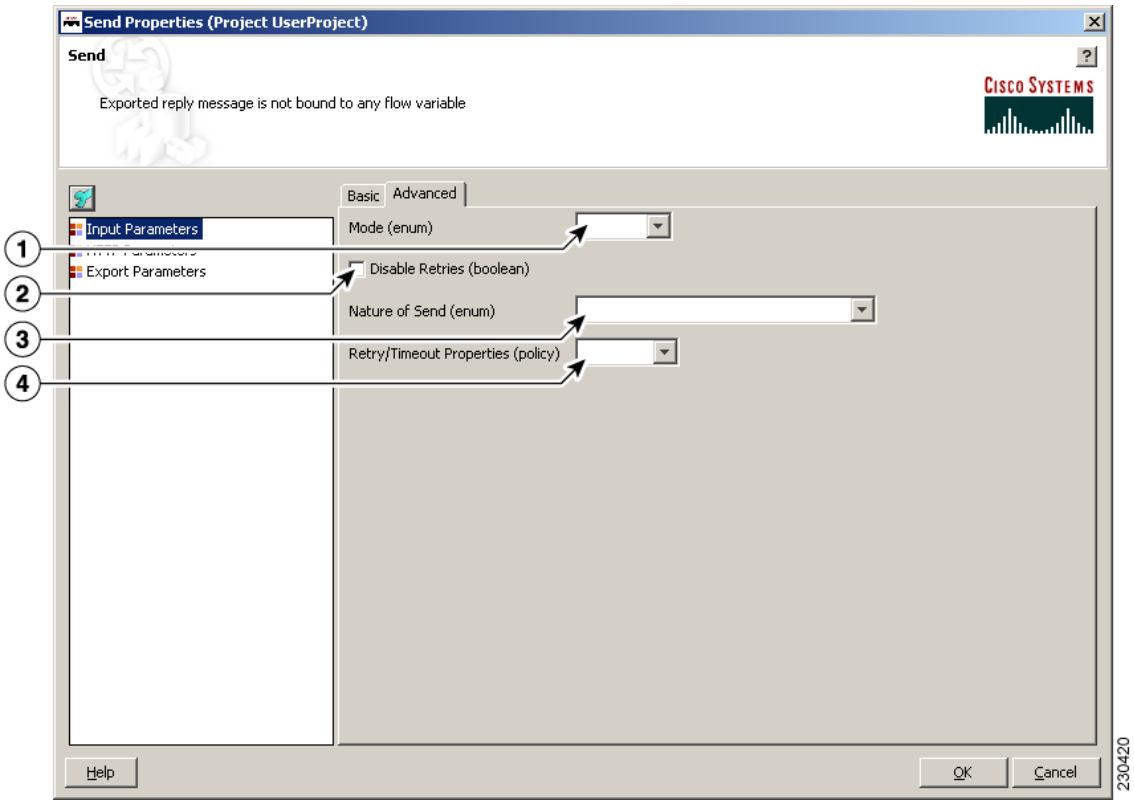
Details

Figure 3-61 Send Properties Window—Input Parameters, Basic



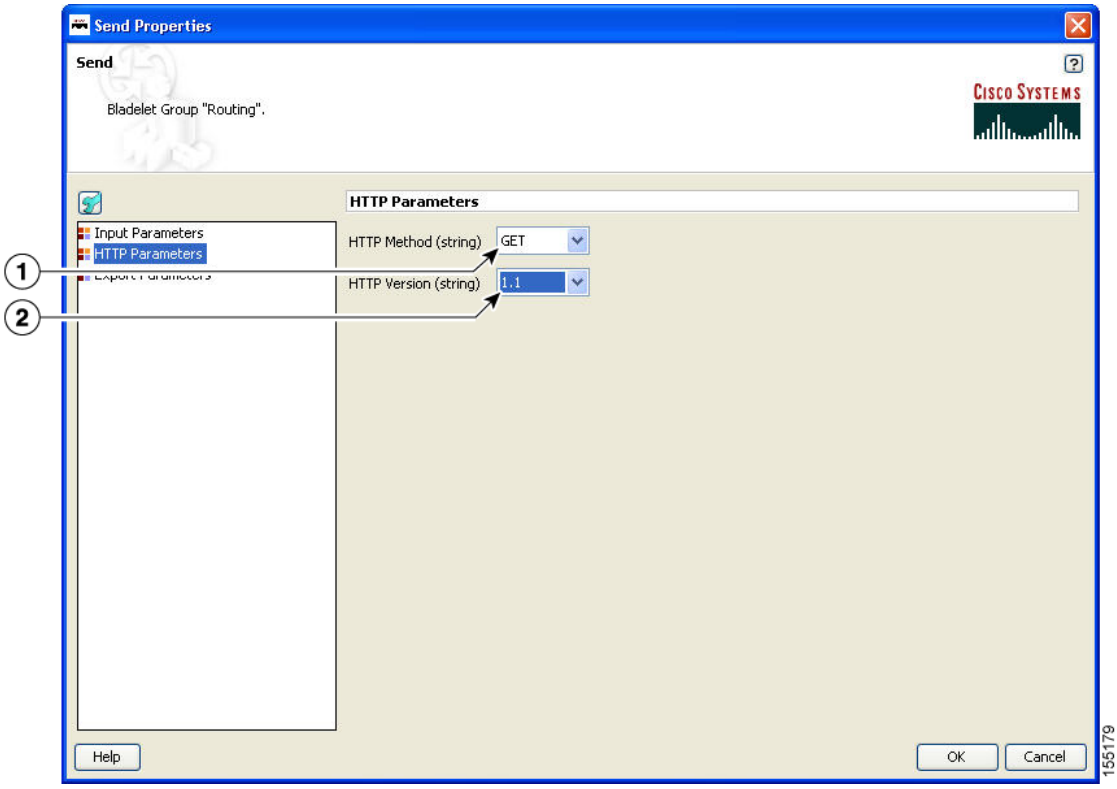
1	Destination URI	Destination URI to be set as the destination of the message being sent out. Overwrites the destination set in the input message. If this field is not set and the message does not have a valid URI, send fails and the client is sent an error message.
2	Message	Message to be sent. If none is specified, the message associate to the REQUEST_MESSAGE variable is used as the input.

Figure 3-62 Send Properties Window—Input Parameters, Advanced



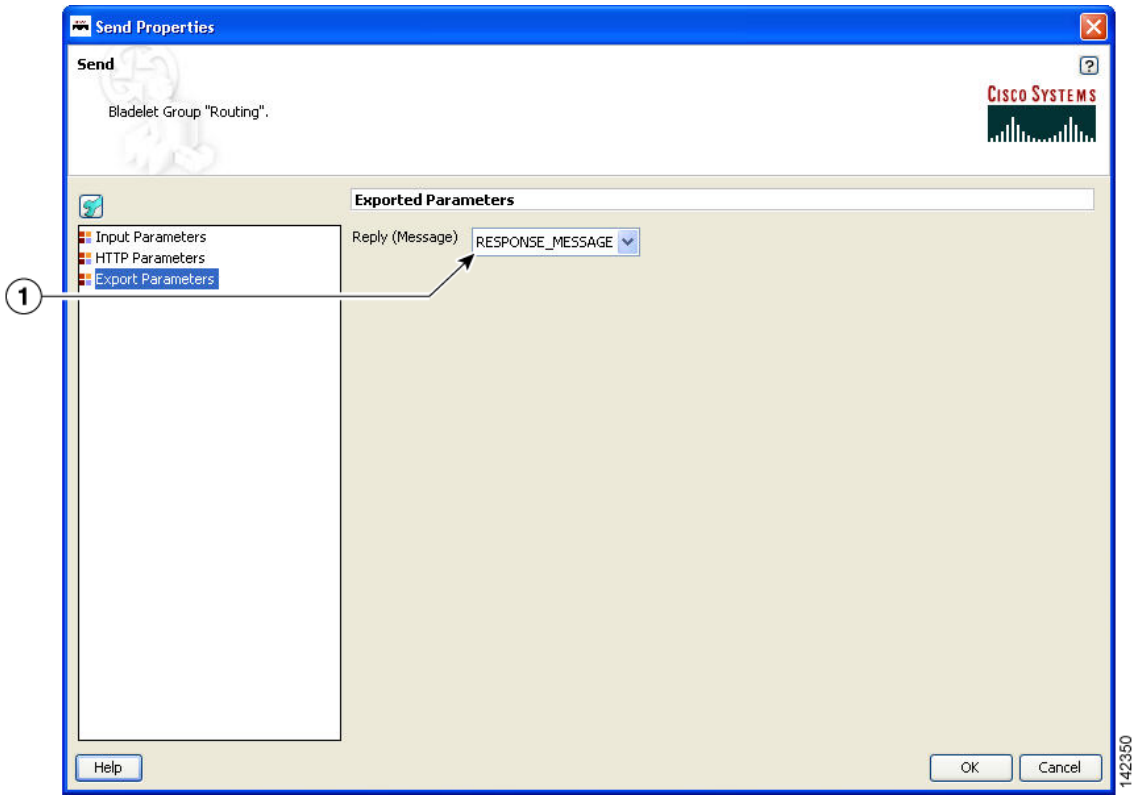
1	Mode	Mode. Choices: Clear or SSL.
2	Disable Retries	If enabled, no retries are attempted during the send operation if there is a failure. If disabled (the default), retries are attempted in case of a failure and the retry behavior is determined by the "Retry/Timeout Policy" selection.
3	Nature of Send	The default is equivalent to having no value selected, in other words the nature of Send (either one-way or two-way) is same as the PEP's interaction style. If the PEP's interaction style is *Request-Only*, then the send is one-way; if it is *Request-Response*, the send is a two-way. For a one-way send, the *Reply* export parameter is be populated at the end of send operation. For a two-way send, the export parameter will be populated. You can override the default behavior by selecting either ONE-WAY or TWO-WAY explicitly. In this case the nature of the send doesn't depend on the PEP's interaction style and is governed by whatever is specified for this parameter.
4	Retry/Timeout Properties (policy)	By selecting *Send Properties*, you can specify the values of the request timeout, the number of retries, and the interval between retries that need to be used for this operation. If none are specified, the default *Send Properties* property set is used to obtain these values.

Figure 3-63 Send Properties Window—HTTP Parameters



1	HTTP Method	Method. Choices: Get or Post. Used only if the message is sent out via HTTP.
2	HTTP Version	1.0 or 1.1. Used only if the message is sent out via HTTP.

Figure 3-64 Send Properties Window—Export Parameters



1	Reply	Message from the end point if the send is two-way. This is not set if the send is one-way. This behavior is governed by the *Nature of Send* parameter under the Advanced tab in the Input Parameters.

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

If the send operation fails, a SendFailure exception is raised. The failure of a send operation may be due to a variety of factors including connection failures and timeouts in retrieving a response. You can chain any bladelet (including more Send bladelets) to the exception path to handle the failure. The ExceptionInfo flow variable (*$PEP.exeception()*) has the error code information corresponding to the failure.

Balance Load

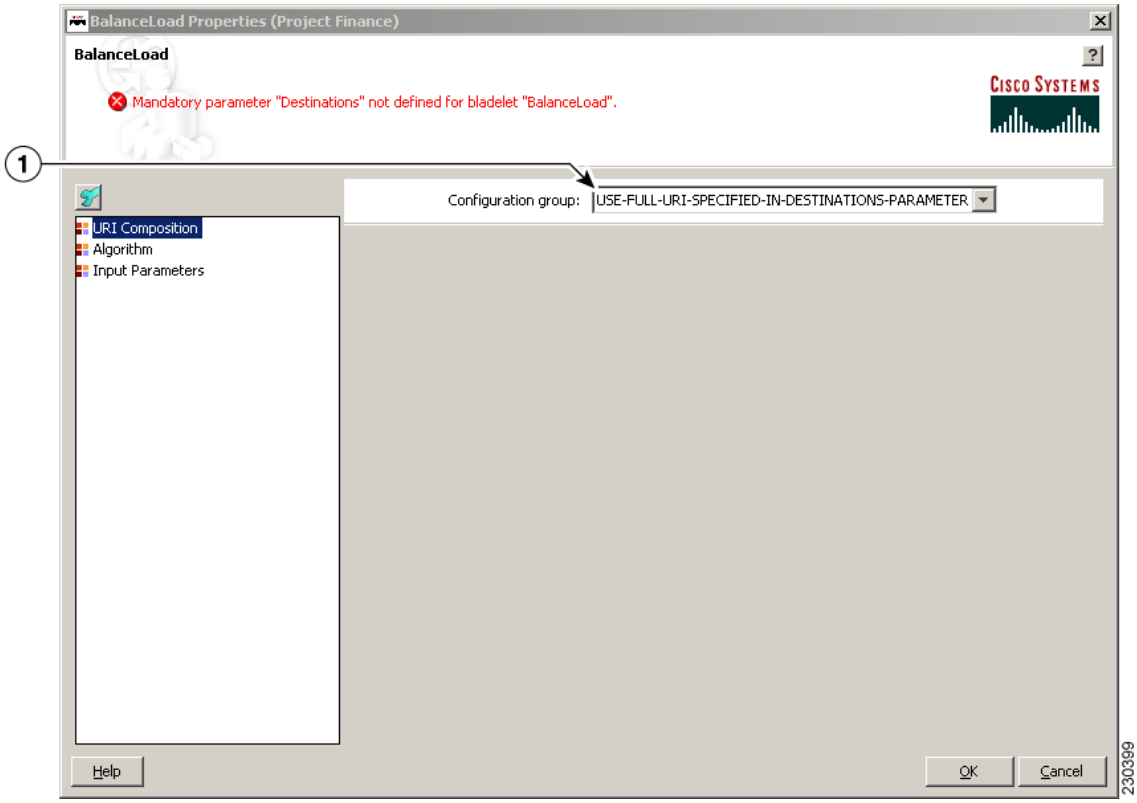
Summary

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.

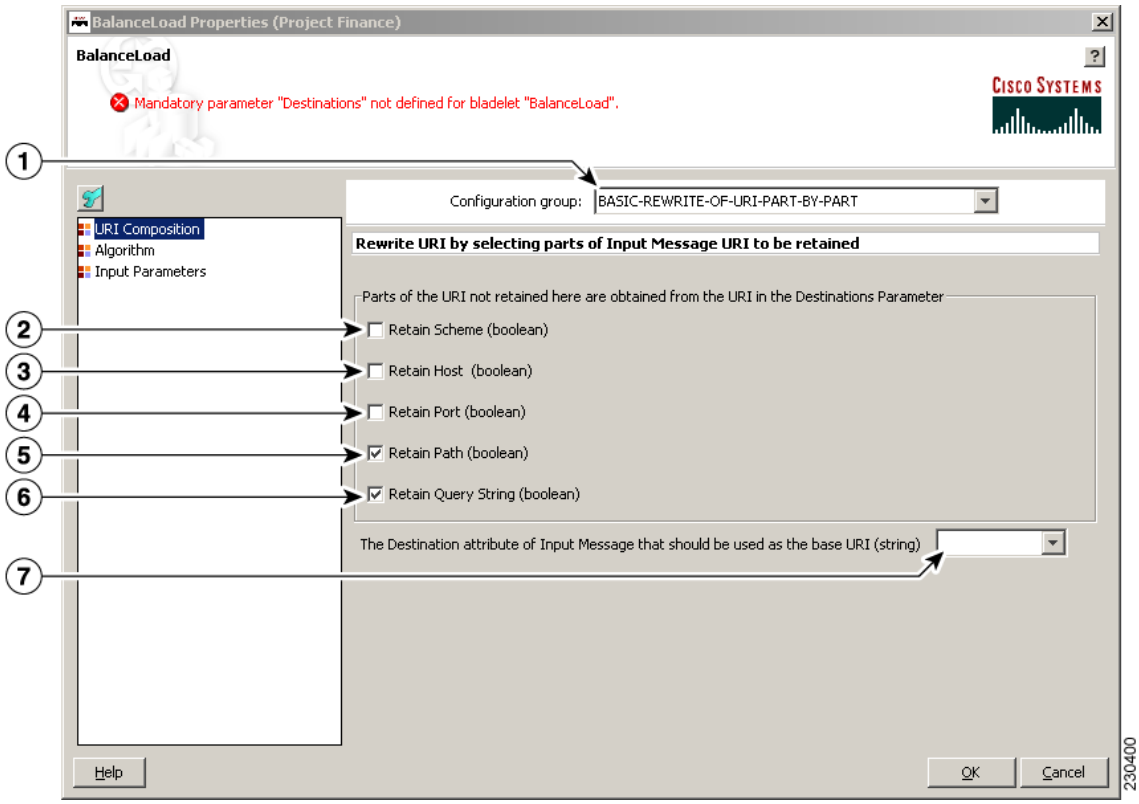
Using URI composition choices, you can specify a portion of the eventual destination URI in the bladelet and can also use information in the URI of the incoming message to formulate the whole URI. One of the typical use cases is to specify the details of the actual servers for load balancing in the bladelet (such as scheme, host and port), while letting the path and query string be picked up from the request message.

Figure 3-65 Balance Load Properties Window—URI Composition Window, Choice 1



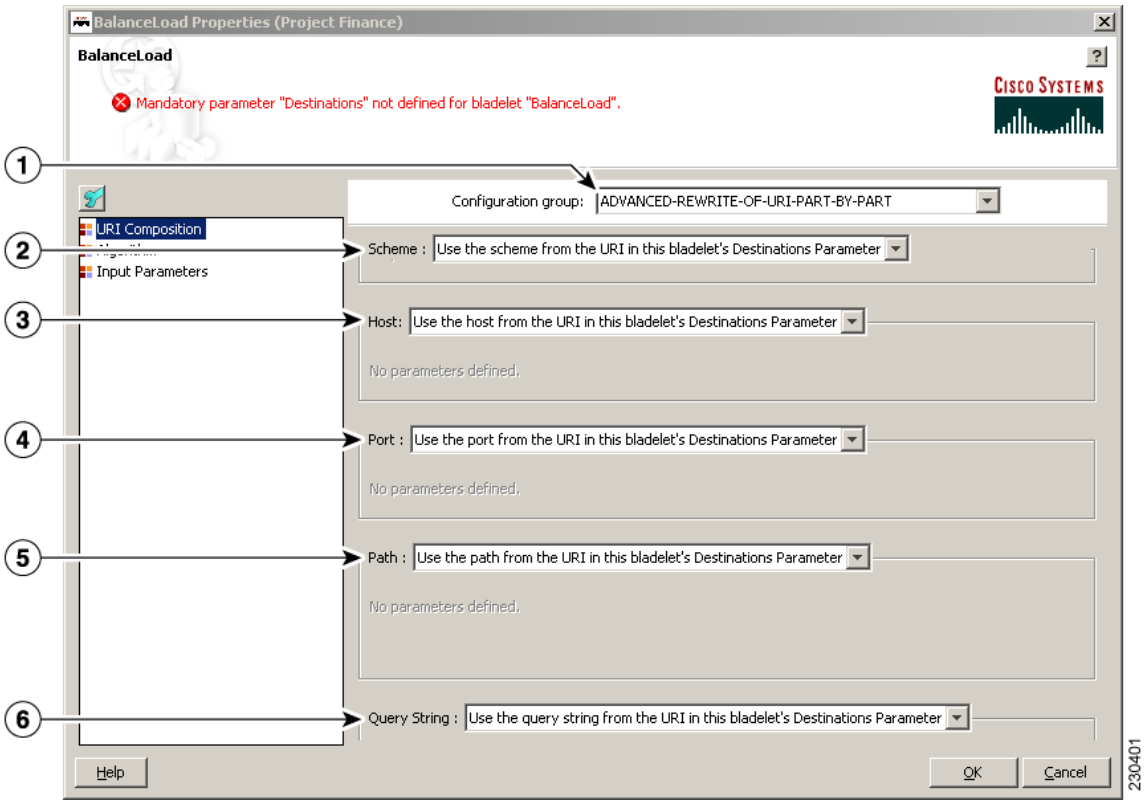
1	Configuration group: USE-FULL-URI-SPECIFIED-IN-DESTINATIONS-PARAMETER	The URIs in the destination parameter are used as is. No rewriting is performed.

Figure 3-66 Balance Load Properties Window—URI Composition Window, Choice 2



1	Configuration group: BASIC-REWRITE-OF-URI-PART-BY-PART	Allows you to retain some parts of the URI of the input message. The parts not retained are picked from the selected destination from the list of configured destinations.
2	Retain Scheme	Selecting this option retains the *scheme* portion of the URI of the input message.
3	Retain Host	Selecting this option retains the *host* portion of the URI of the input message.
4	Retain Port	Selecting this option retains the *port* portion of the URI of the input message.
5	Retain Path	Selecting this option retains the *path* portion of the URI of the input message. This option is selected by default.
6	Retain Query String	Selecting this option retains the *query string* portion of the URI of the input message. This option is selected by default.
7	Destination attribute	Allows you to choose which destination to use as the source. By default, the current destination is used.

Figure 3-67 Balance Load Properties Window—URI Composition Window, Choice 3



1	Configuration group: ADVANCED-REWRITE-OF-URI-PART-BY-PART	Allows you multiple ways of configuring each portion (scheme, *host*, *port*, *path*, and *query string*) of the URI of the input message. Each individual portion can be configured to one of the following: • Use the [*portion*] from the URI in the bladelet's destinations parameter • Use the [*portion*] from the Destination URI of the Input Message • Specify the [*portion*] as a variable or literal
2	Scheme	Allows you to set the *scheme* portion of the input message.
3	Host	Allows you set the *host* portion of the input message.
4	Port	Allows you to set the *port* portion of the input message.
5	Path	Allows you to set the *path* portion of the input message.
6	Query String	Allows you to set the *query string* portion of the input message.

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 3-72 on page 3-89)
- Weighted round-robin (Figure 3-73 on page 3-90)
- Adaptive (Figure 3-74 on page 3-91)
- Highest Priority (Figure 3-75 on page 3-92)

Prerequisites and Dependencies

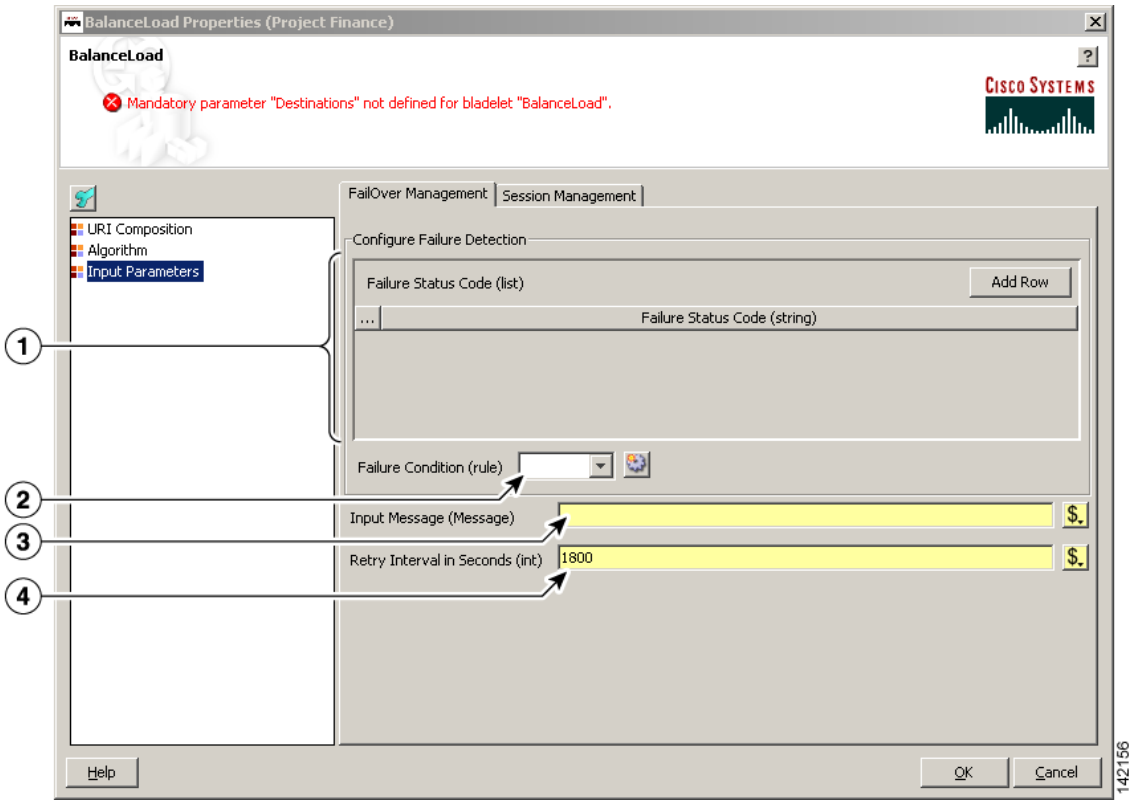
None.

Details

Two tabs, FailOver Management and Session Management, are under the Input Parameters section (Figure 3-68 to Figure 3-71).

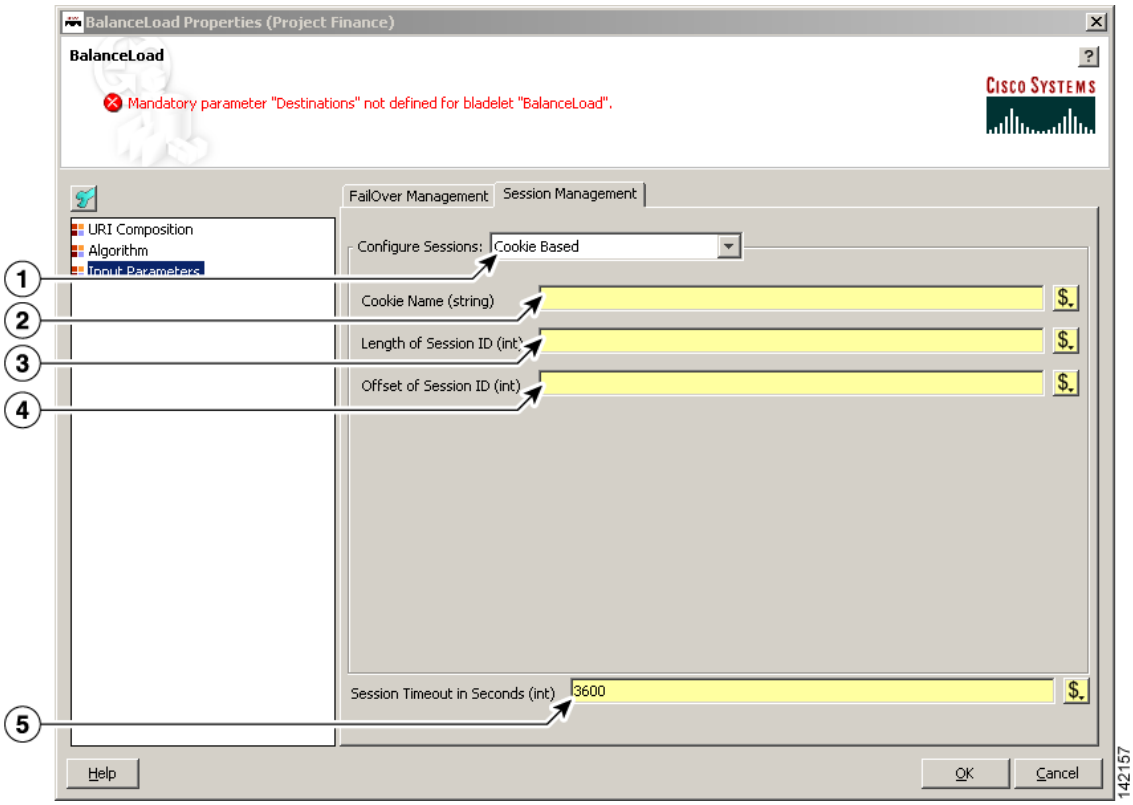
Note Each required field in the Bladelet Properties window is marked by a red asterisk. Until all required fields are completed with the correct value, an error message appears on top of the Bladelet Properties window to indicate which field remains to be completed or indicates that there is a parameter type mismatch and so on before the Bladelet is completely configured.

Figure 3-68 Balance Load Properties Window—Configure Parameters, FailOver Management



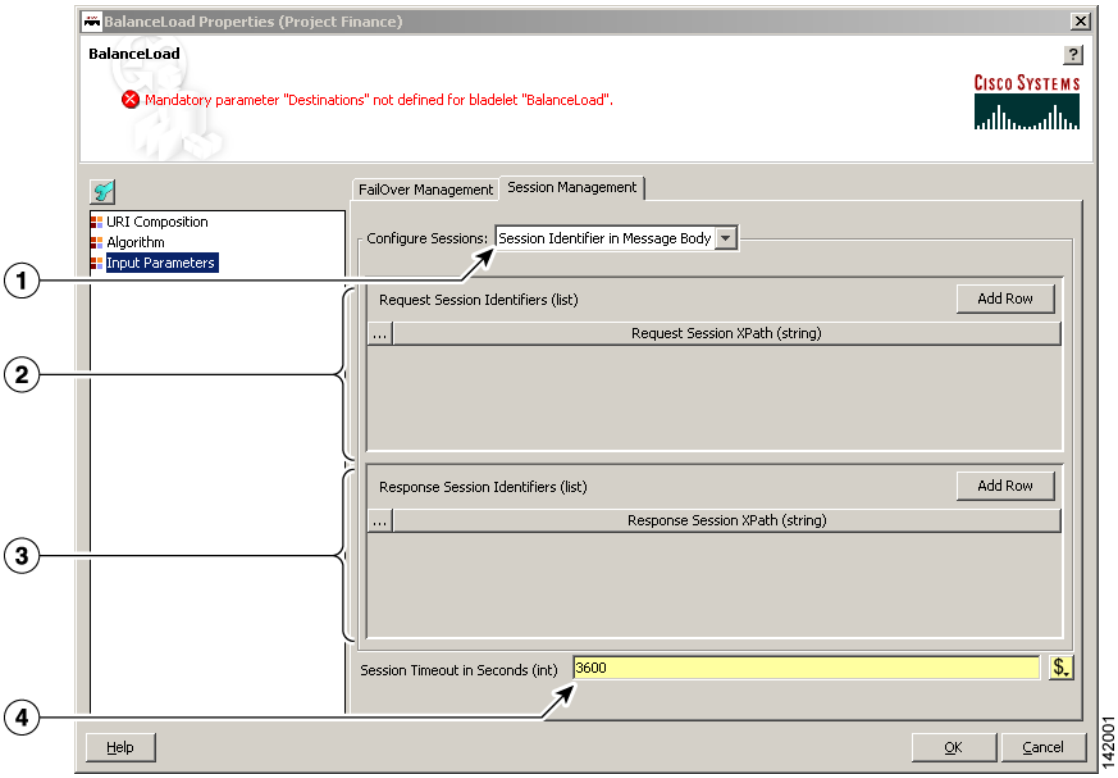
1	Failure Status Code	Optional. One or more failure error codes (examples: 404, 500) that indicate a failed endpoint. If none specified, any error code in the 400-600 range is considered to be a failure. Specifying particular error codes helps if only some of these errors should be considered fatal. 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).
2	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.
3	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.
4	Retry Interval in Seconds	Time for which a destination is not used again once it is considered to have experienced a failure.

Figure 3-69 Balance Load Properties Window—Input Parameters, Session Management 1



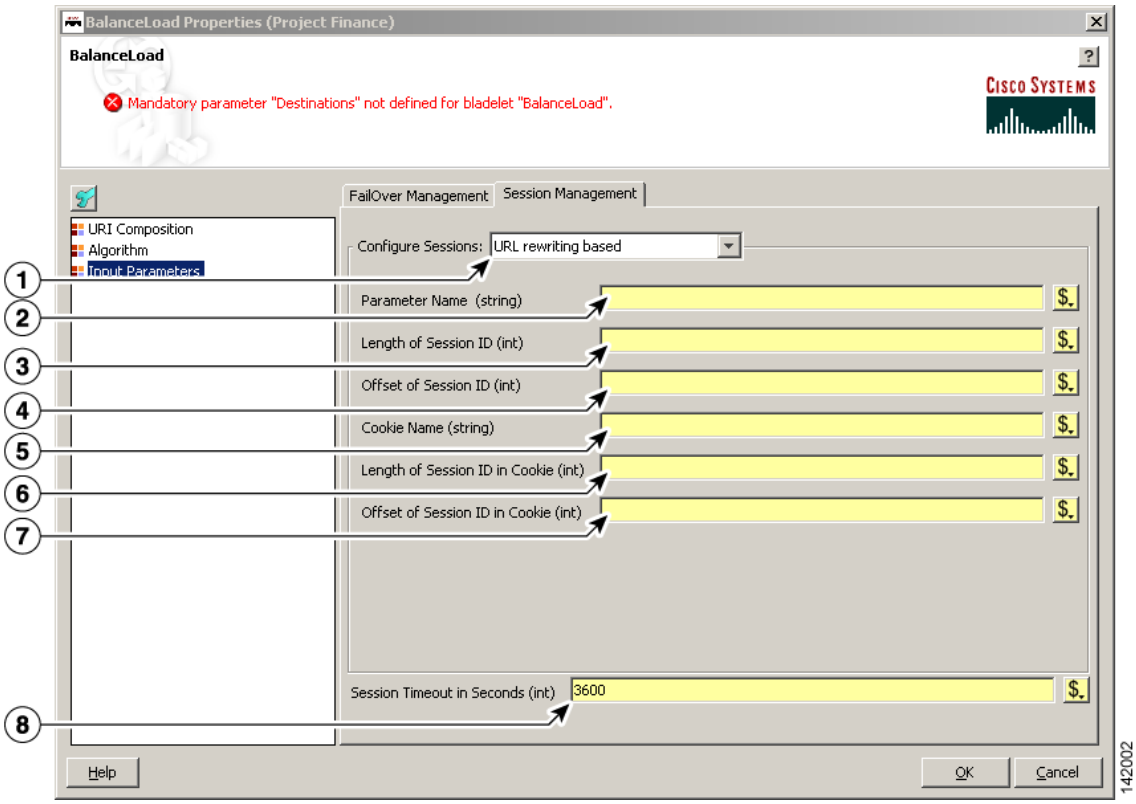
1	Cookie Based	Select the session type. In the figure, Cookie Based is selected.
2	Cookie Name	Name of the cookie that carries the session information (example: in Unit3 this is JSESSIONID) in both request and response.
3	Length of Session ID	If the session ID is only a part of the cookie value (as opposed 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).
4	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).
5	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.

Figure 3-70 Balance Load Properties Window—Input Parameters, Session Management 2



1	Session Identifier in Message Body	Configuration session type. In the figure, Session Identifier in Message Body is chosen.
2	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.
3	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.
4	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.

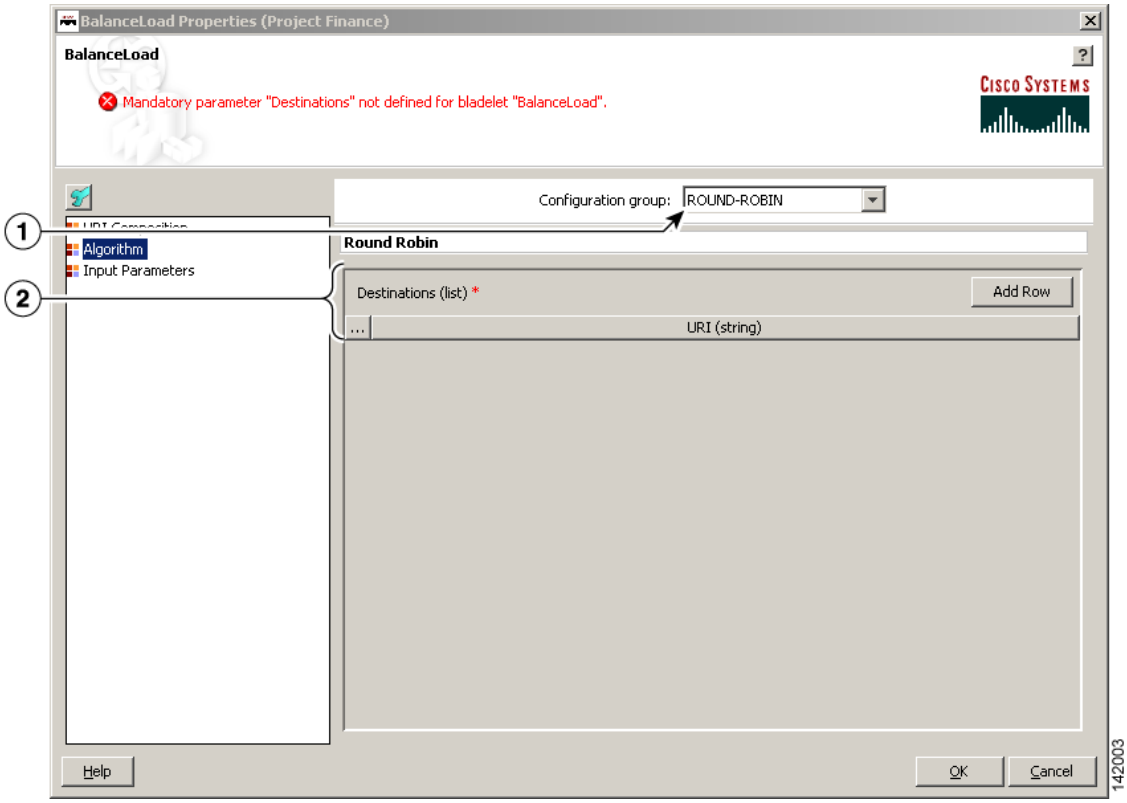
Figure 3-71 Balance Load Properties Window—Input Parameters, Session Management 3



1	URL Rewriting Based	Configuration session type. In the figure, URL rewriting based is chosen.
2	Parameter Name	Name of the parameter in the rewritten URL that carries the session information (for example, in Unit3 this is `jsessionid=').
3	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.
4	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.
5	Cookie Name	Name of the cookie that carries the session information (example: in Unit3 this is JSESSIONID) in the response message headers.

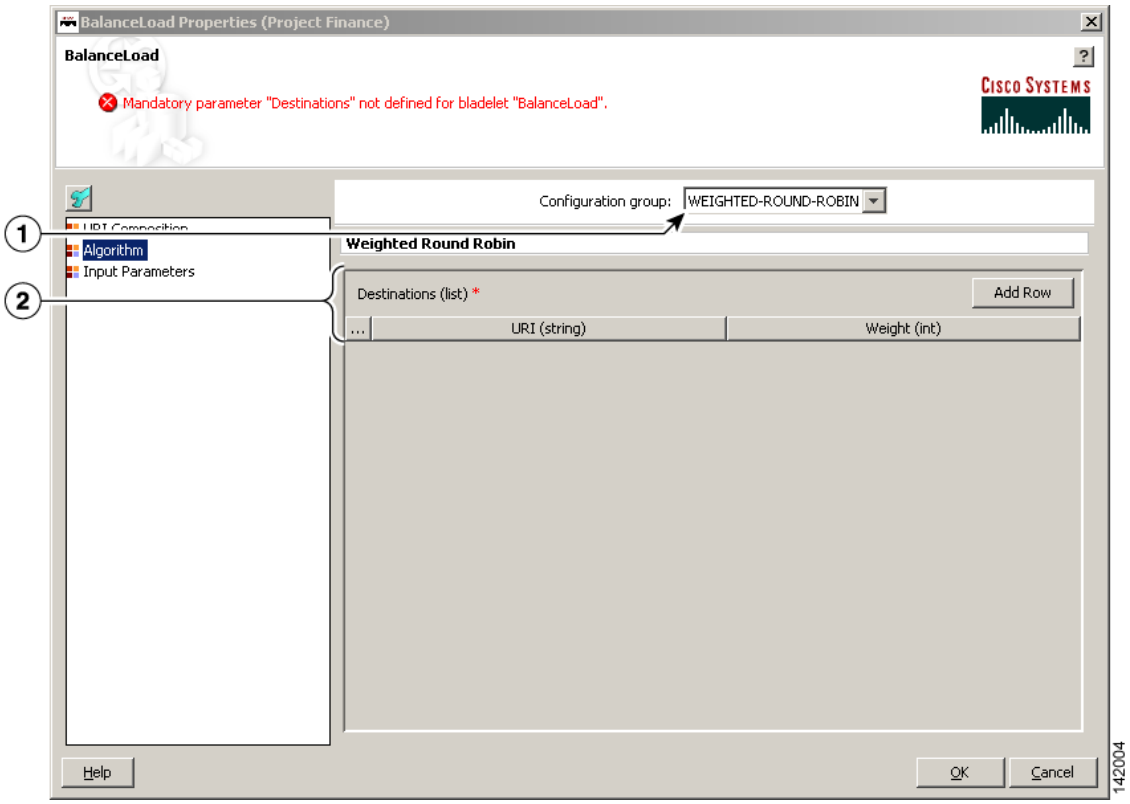
6	Length of Session ID in Cookie	If the session ID is only a part of the cookie value (as opposed to being the whole cookie value), length of the session ID within the cookie value. Need not be specified if the session ID is the entire cookie value such as Unit3. Applies only to response message headers, associated with the Cookie Name parameter.
7	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.
8	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.

Figure 3-72 Balance Load Properties Window—Algorithm, Round-Robin



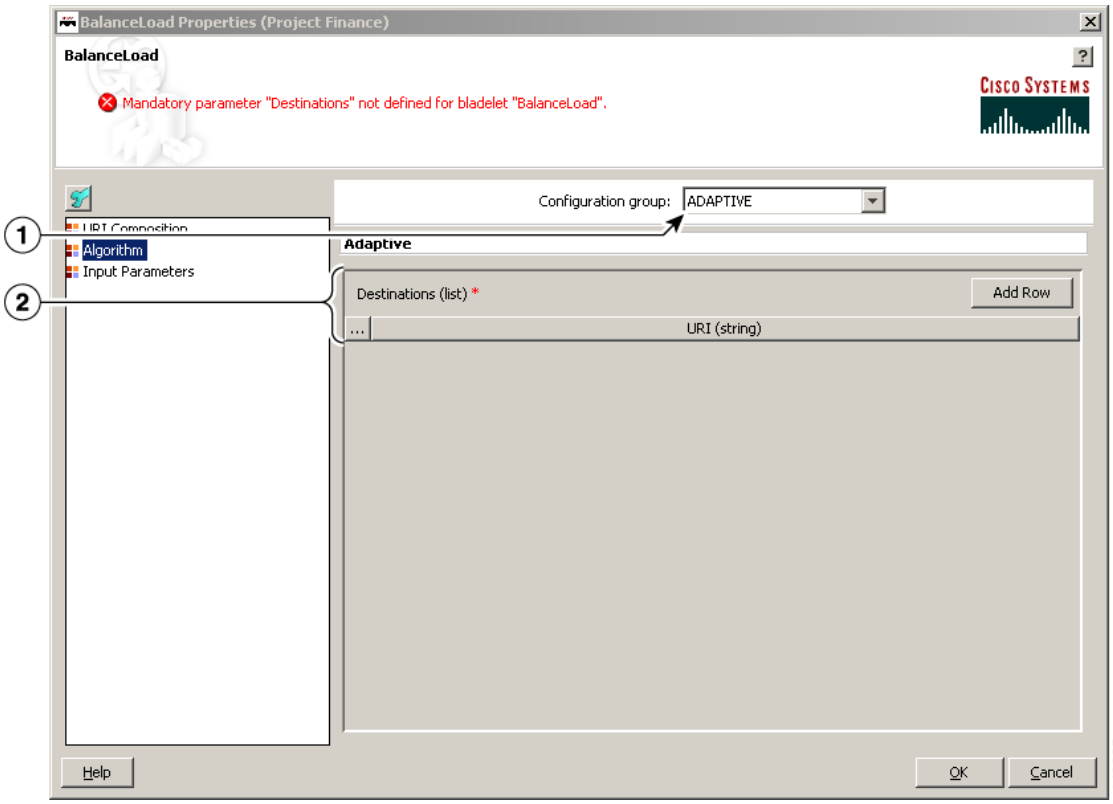
1	Configuration Group	Configuration group, set here to Round-Robin.
2	Destinations	One or more destination URIs to be load-balanced, based on the following: • Endpoint with least response time • Endpoint with least average wait time (when concurrency > 1)

Figure 3-73 Balance Load Properties Window—Algorithm, Weighted Round-Robin



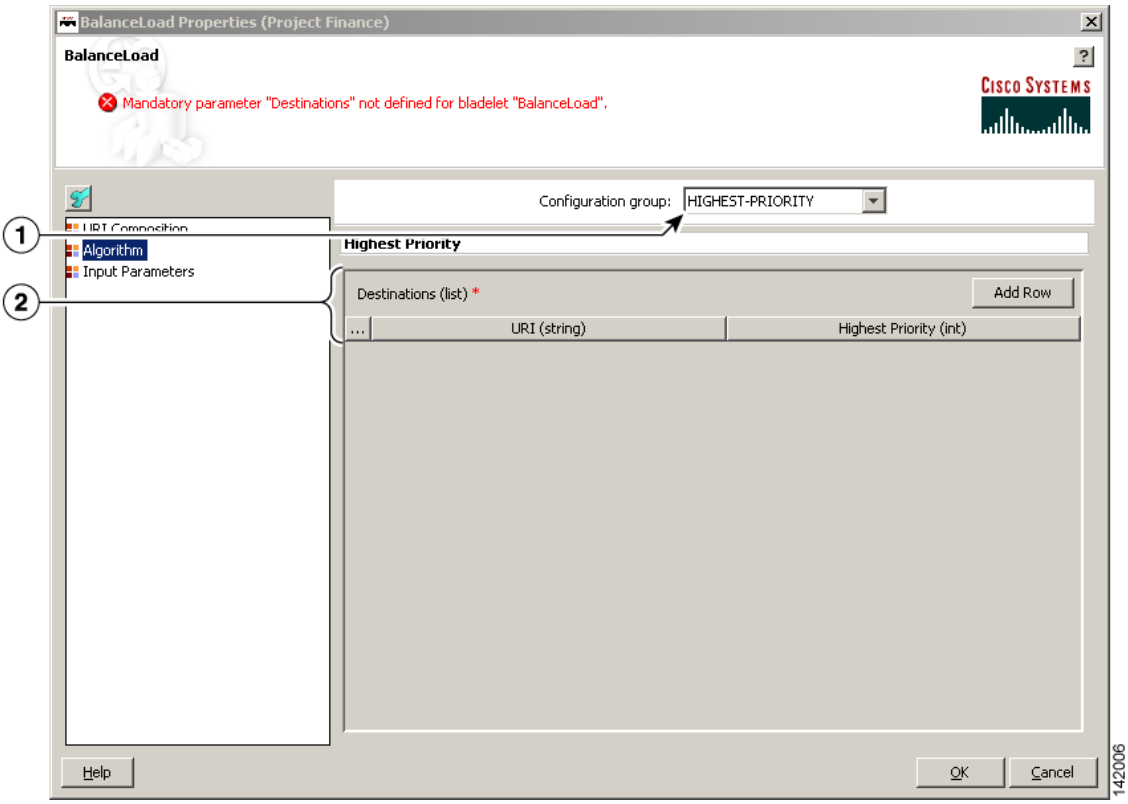
1	Configuration Group	Configuration group, set here to Weighted-Round-Robin.
2	Destinations	One or more URIs (string) and weight (int). This algorithm picks the destination based the corresponding weights. Distribution of messages to the destination is in proportion to the corresponding weight. So, if 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 3-74 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 3-75 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

If all the end points and destinations specified in BalanceLoad are unreachable, the bladelet raises a LBFailed exception. The correct way to achieve BalanceLoad functionality is to have a BalanceLoad followed by a Send. When an LB failure occurs due to the failures in the send operation, the exception path of BalanceLoad is activated and not that of send.

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 **AMC > 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.
- For SiteMinder 5.5 authorization, use the AMC server to define SiteMinder configuration parameters. The full path in AMC is **AMC > Properties > Application > Netegrity SiteMinder**.

  The user info must be defined in SiteMinder Policy Server or LDAP repository configured in SiteMinder Policy Server for Authorize to recognize. The identity of the user must be extracted using Identify Bladelet before Authorize can be used in a PEP. Before a user can be authorized to access a resource using SiteMinder 5.5 method, user must be authenticated using SiteMinder 5.5 authentication method because SiteMinder authorization requires user to be authenticated.

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.

SiteMinder 5.5

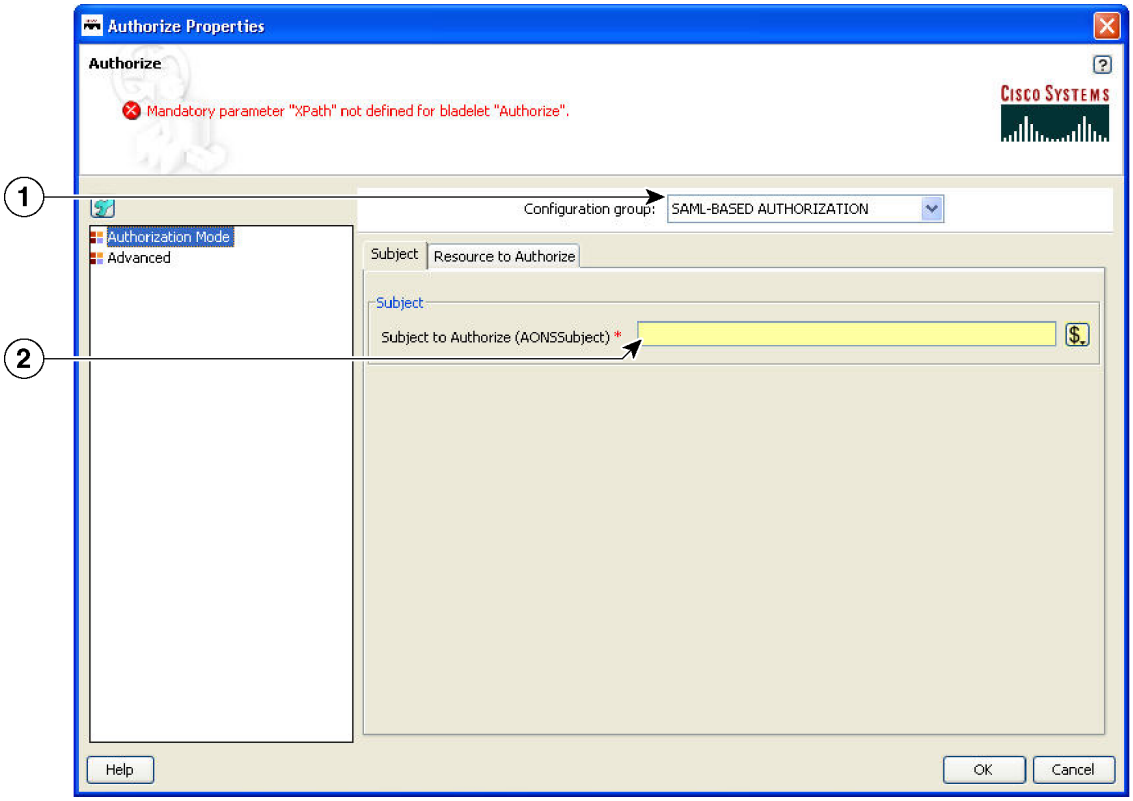
SiteMinder 5.5 method for authorization uses SiteMinder 5.5 Policy Server for authorization. This method authorizes a user's access to a web application resource configured in the Authorize Bladelet. Access method for accessing the resource usually can be GET or POST which is determined from the request message being handled by AON.

Further resource being accessed must be a protected resource in SiteMinder Policy Server. If resource is not protected, it will result in function failure and access will be denied.

SiteMinder authorization requires that user to authorize is authenticated using SiteMinder authentication method. Authenticate bladelet can be used to authenticate a user using SIteMinder Policy Server before the user is authorized by Authorize bladelet.

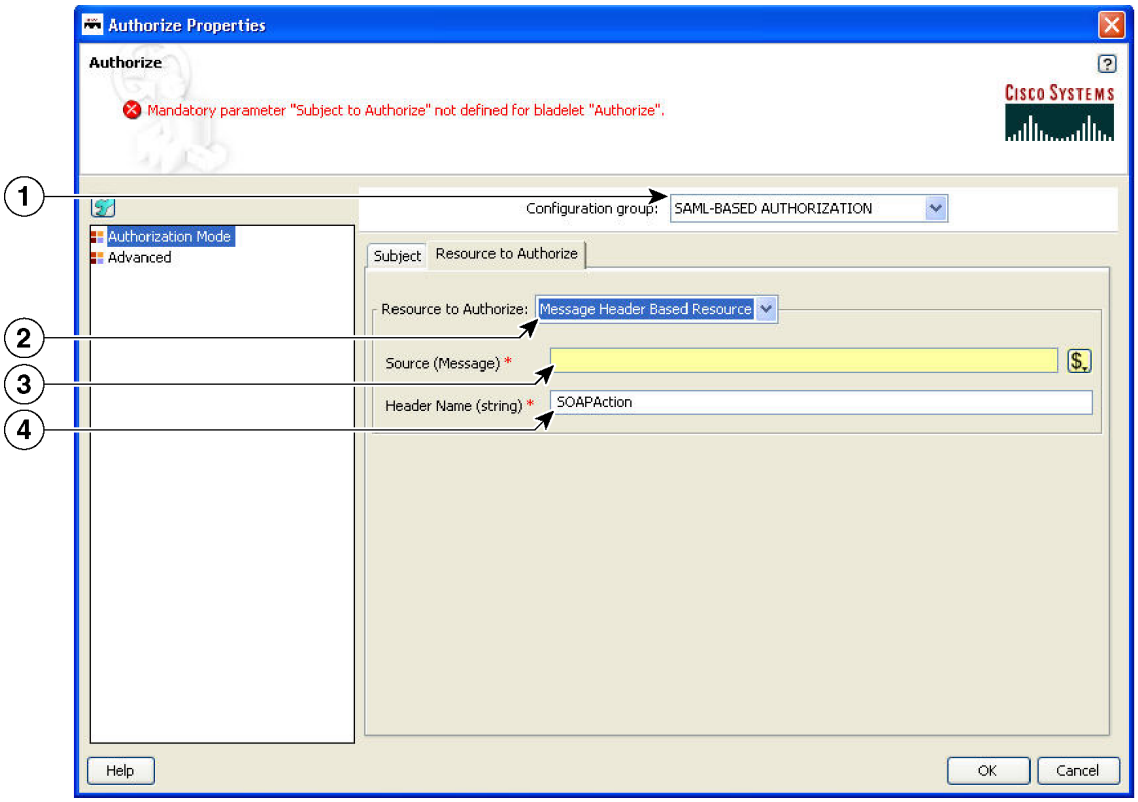
Note Each required field in the Bladelet Properties window is marked by a red asterisk. Until all required fields are completed with the correct value, an error message appears on top of the Bladelet Properties window to indicate which field remains to be completed or indicates that there is a parameter type mismatch and so on before the Bladelet is completely configured.

Figure 3-76 Authorize Properties Window—Authorization Mode, SAML-based Authorization 1



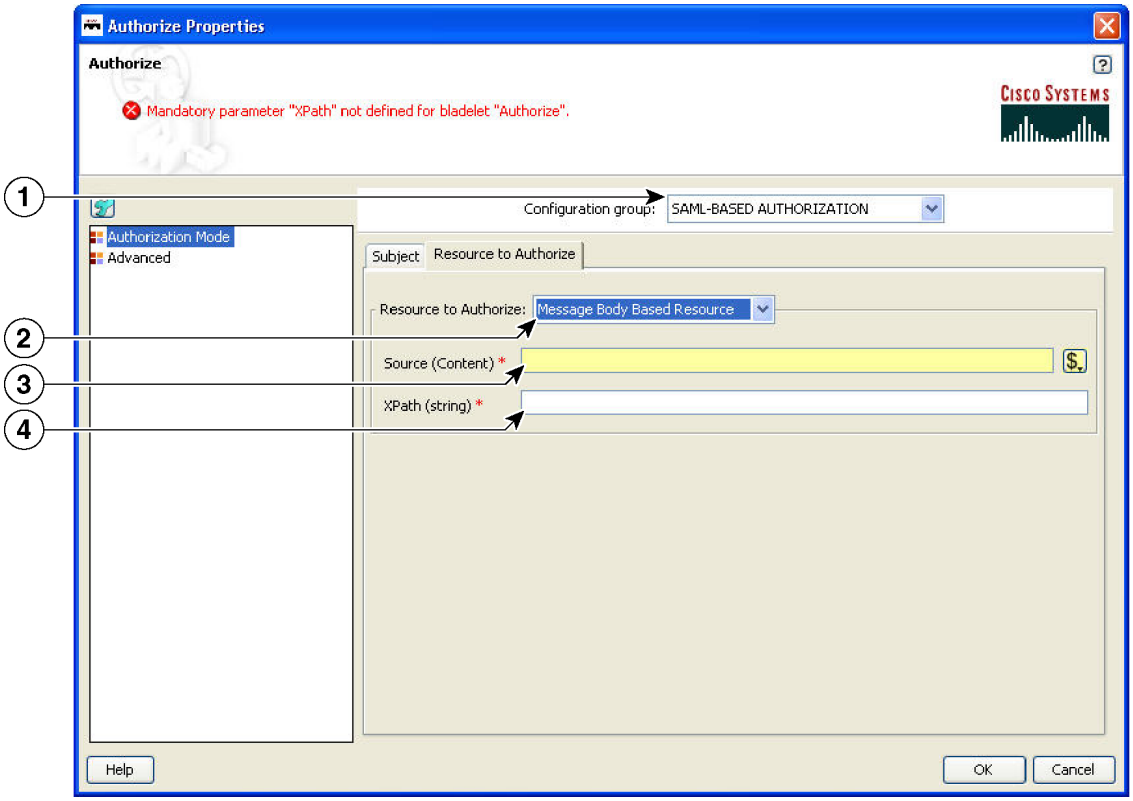
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Subject to Authorize	Subject to use for SAML authorization verification. Extract this subject before the authorization Bladelet is invoked in the PEP using Identify Bladelet.

Figure 3-77 Authorize Properties Window—Authorization Mode, SAML-based Authorization 2



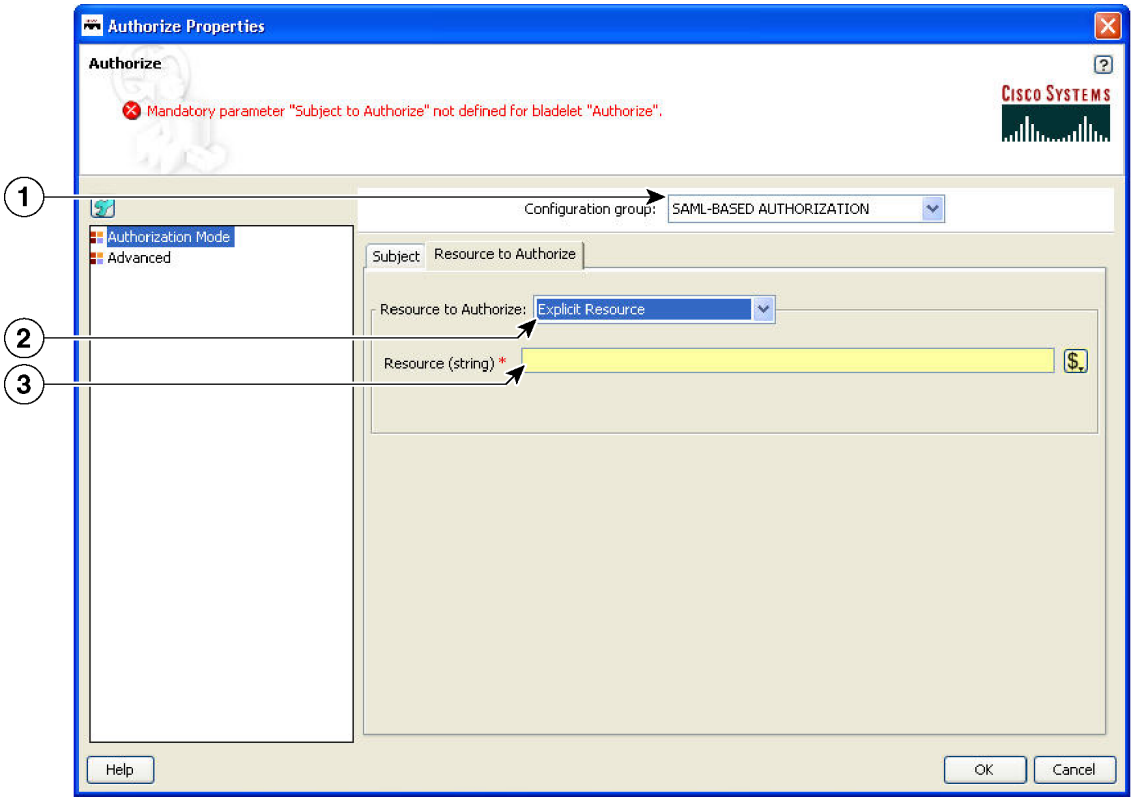
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Message Header Based Resource	Whether or not the resource to authorize is specified in the value of a message header field.
3	Source	Message that identifies the resource.
4	Header Name	Message header that contains the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

Figure 3-78 Authorize Properties Window—Authorization Mode, SAML-based Authorization 3



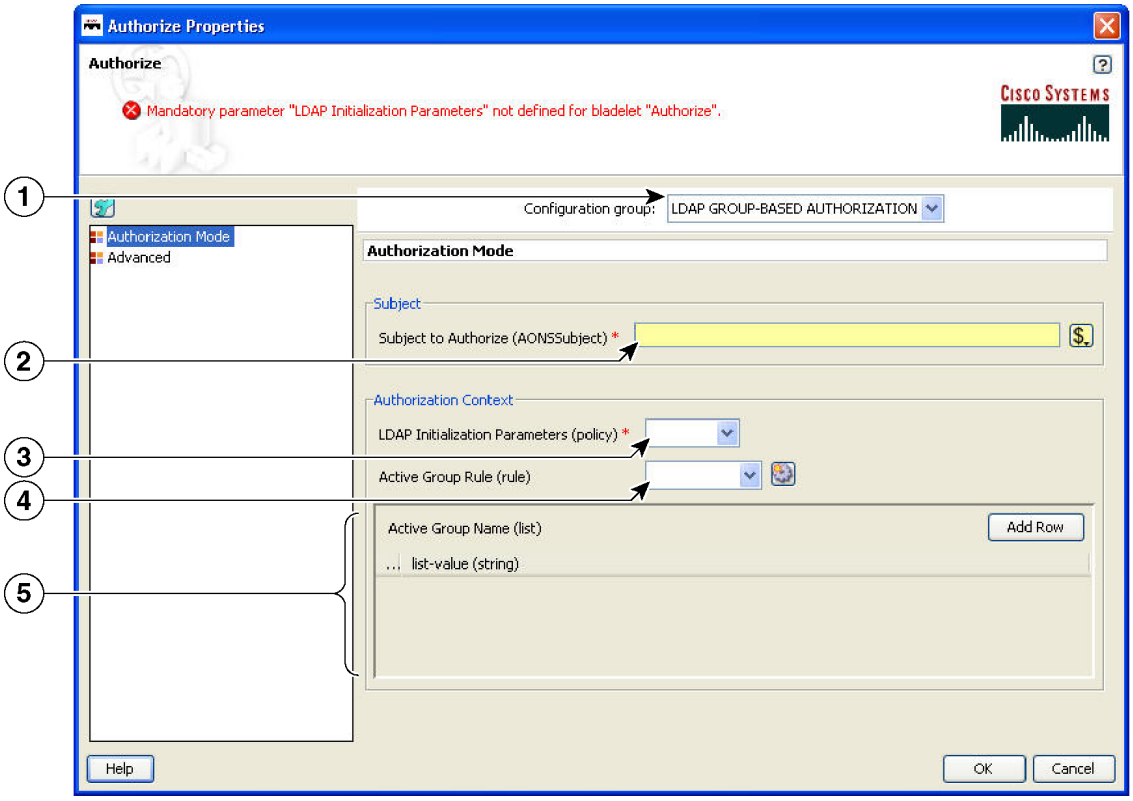
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Message Body Based Resource	Whether or not the resource to authorize is found in the message body. An XPath expression extracts the resource value from the message body.
3	Source	Message whose body contains the resource.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

Figure 3-79 Authorize Properties Window—Authorization Mode, SAML-based Authorization 4



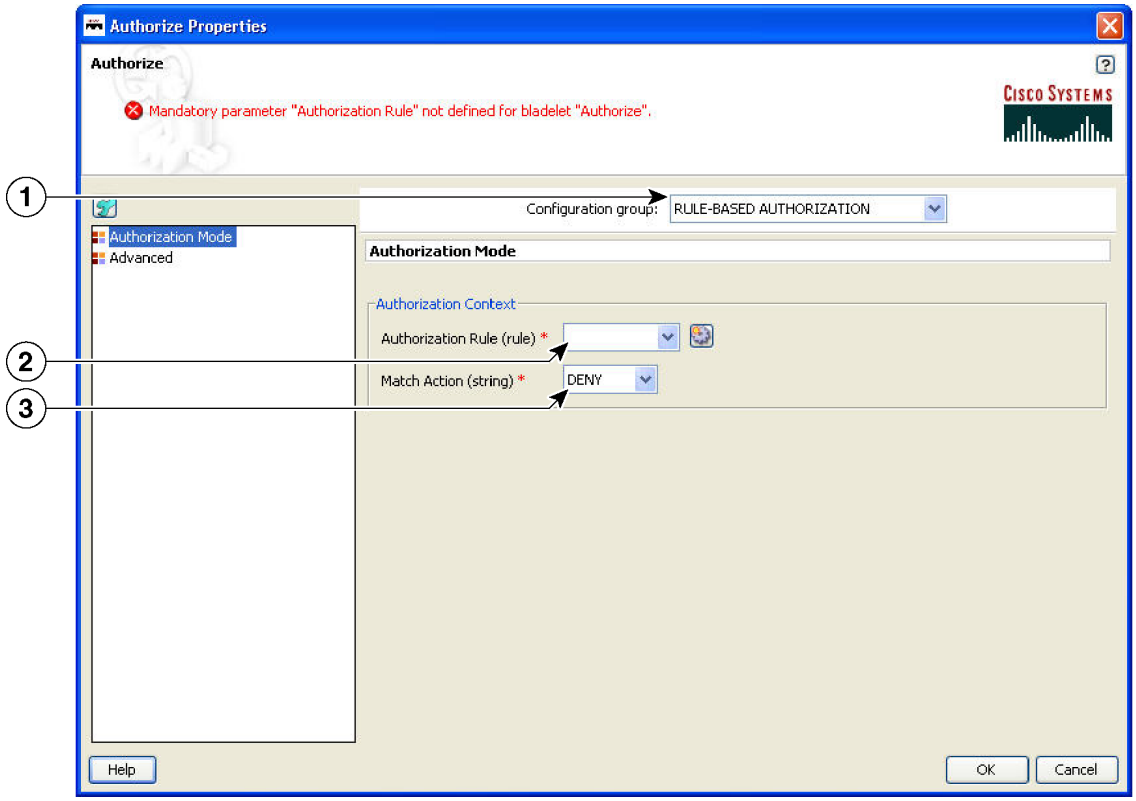
1	Configuration Group	Configuration group, set here to SAML-Based Authorization.
2	Explicit Resource	Whether or not the resource to authorize is specified explicitly.
3	Resource	Resource value. Can be explicitly specified or it be bound to a PEP variable (String) that specifies the resource.

Figure 3-80 Authorize Properties Window—Authorization Mode, LDAP Group-Based Authorization



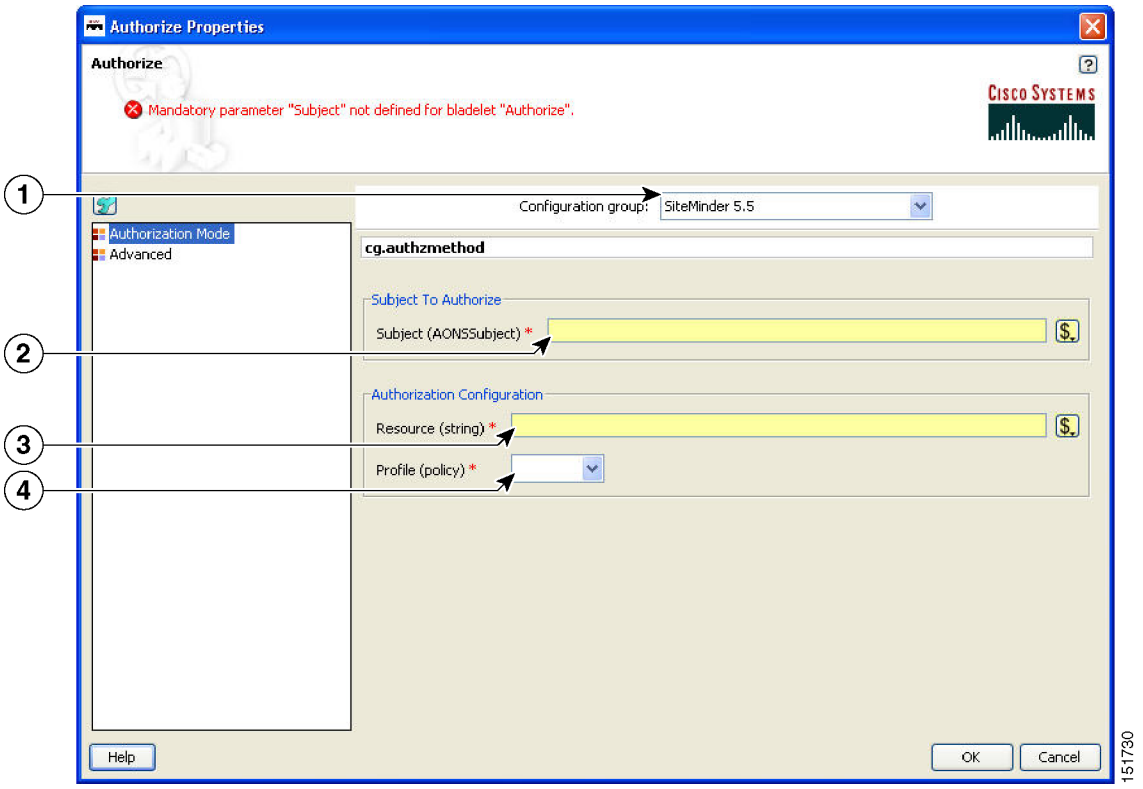
1	Configuration Group	Configuration group, set here to LDAP Group-Based Authorization.
2	Subject to Authorize	Subject to use for LDAP group-based authorization. This subject should be extracted before the authorization Bladelet is invoked in the PEP using Identify Bladelet. Further this subject should 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 3-81 Authorize Properties Window—Authorization Mode, Rule-Based Authorization



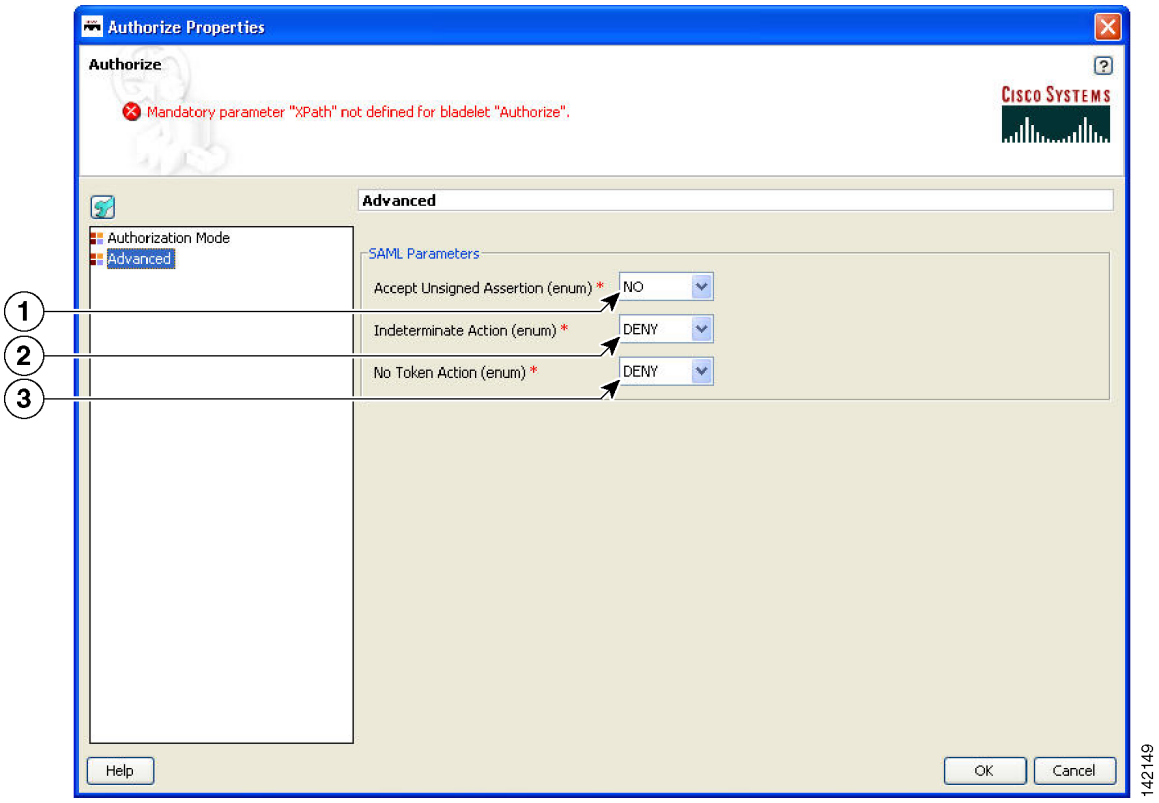
1	Configuration Group	Configuration group, set here to Rule-Based Authorization.
2	Authorization Rule	One or more conditions in a conjunctive expression. If all conditions evaluate as true, then action specified in Match Action parameter is taken.
3	Match Action	Action to be taken when all conditions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

Figure 3-82 Authorize Properties Window—Authorization Mode, SiteMinder 5.5



1	Configuration Group	Configuration group, set here to SiteMinder 5.5.
2	Subject	AONSSubject. It specifies the subject which is authorized.
3	Resource	String. Resource the subject is trying to access.
4	Profile	Policy. Property set that specifies parameters used to connect to a Policy Server. These parameters are SM Policy Server, Agent Name, and Agent Secret.

Figure 3-83 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.

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 Encrypt 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

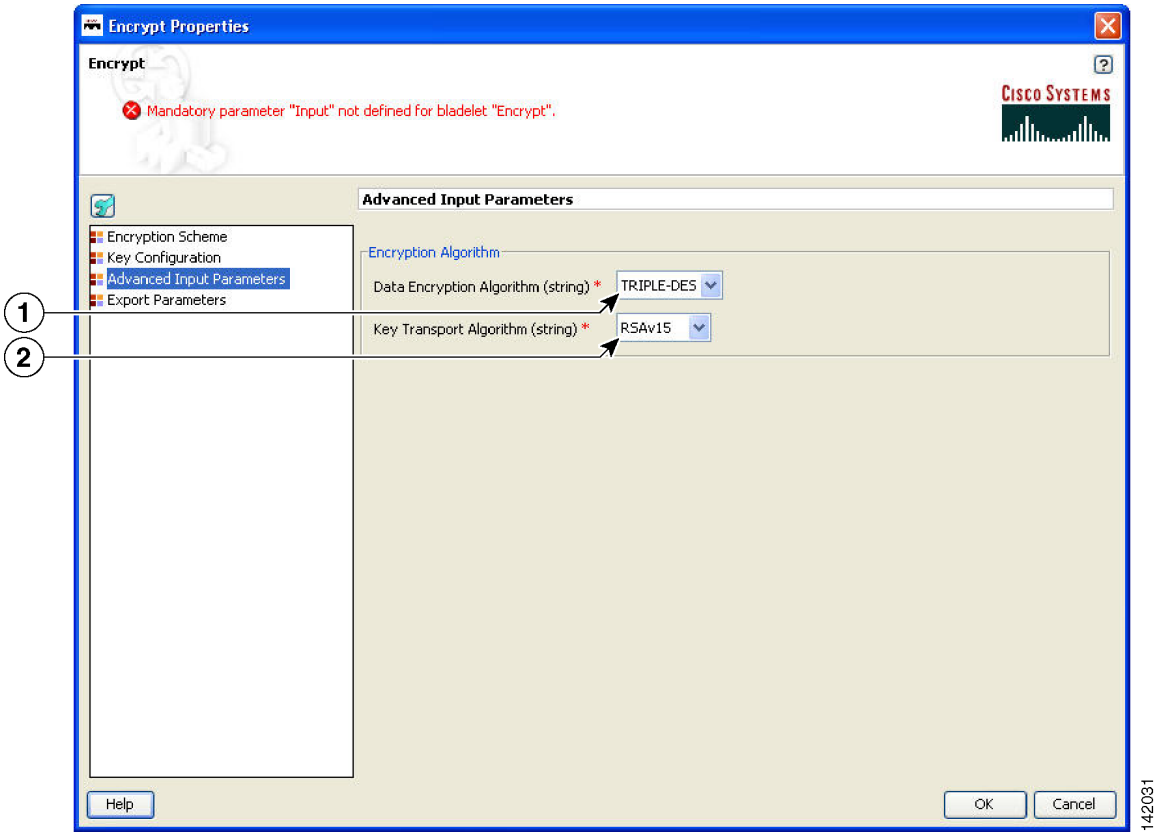
This Bladelet encrypts incoming SOAP, XML, and Non-XML messages using a dynamically generated symmetric key. The symmetric key is encrypted using the asymmetric public key of the message recipient. Given the public key of the recipient message as an input parameter, this Bladelet moves the CPU-intensive 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.

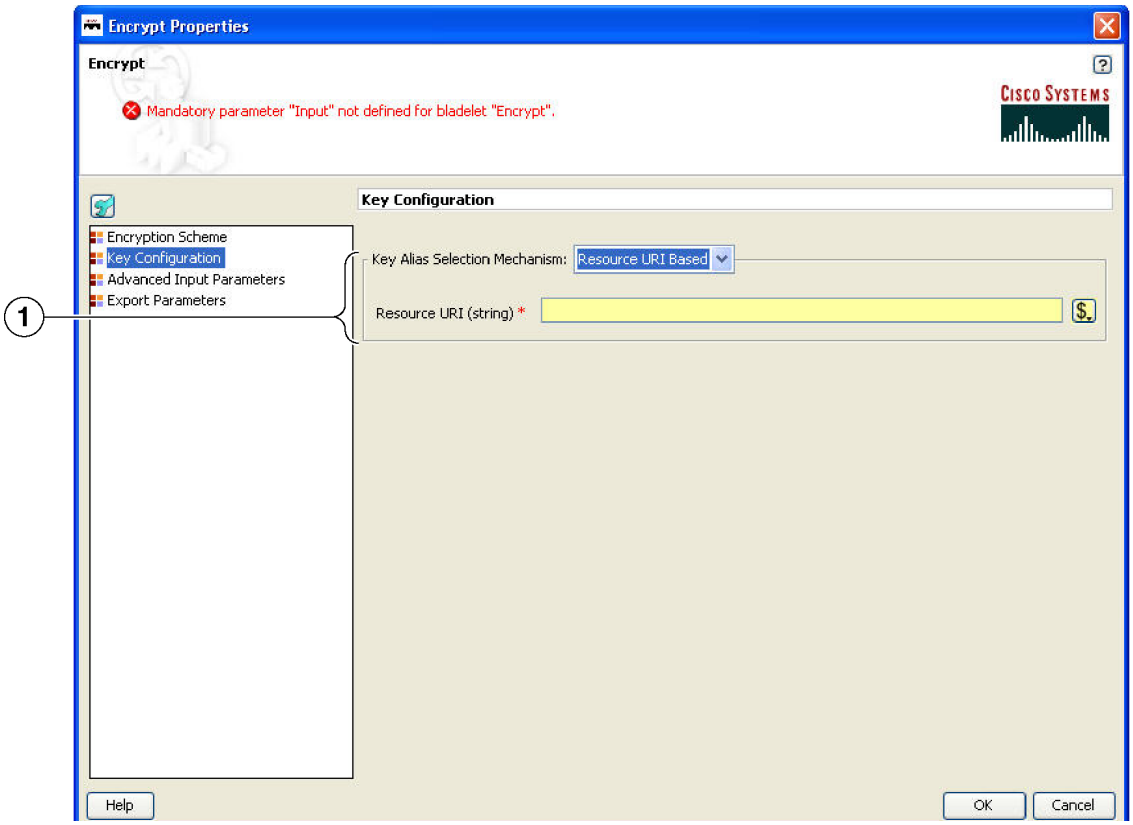
AON checks the destination URI of the message to determine the key alias for Encryption. For asymmetric key encryption, the encryption key alias is identical to the destination hostname. For example, if the destination URI is http://server1.domain.com/someservice, the encryption Bladelet expects an RSA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

Figure 3-84 Encrypt Properties Window—Advanced Input Parameters



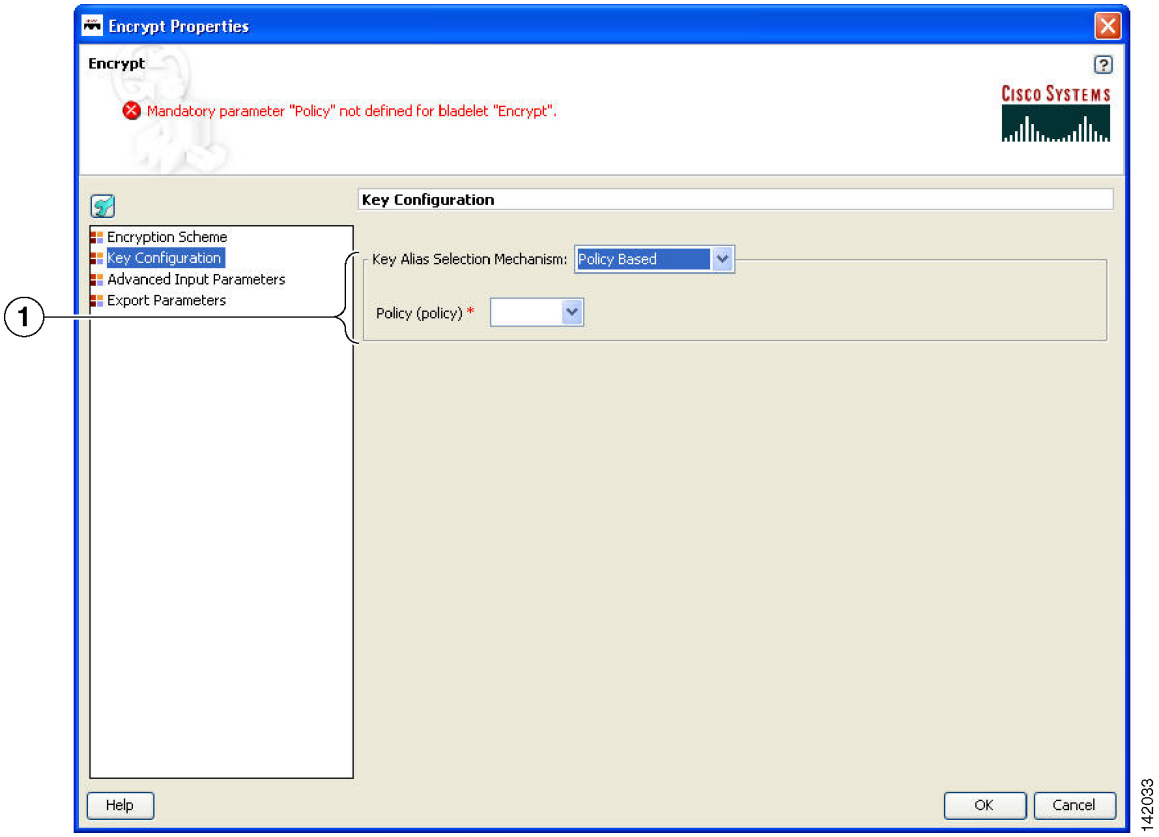
1	Data Encryption Algorithm	Algorithm used to encrypt the actual data. Choices: Triple-DES, AES128, AES192, and AES256.
2	Key Transport Algorithm	Encryption key. Currently only RSAv1.5 is supported.

Figure 3-85 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 3-86 Encrypt Properties Window—Key Configuration, Key Alias Policy Based

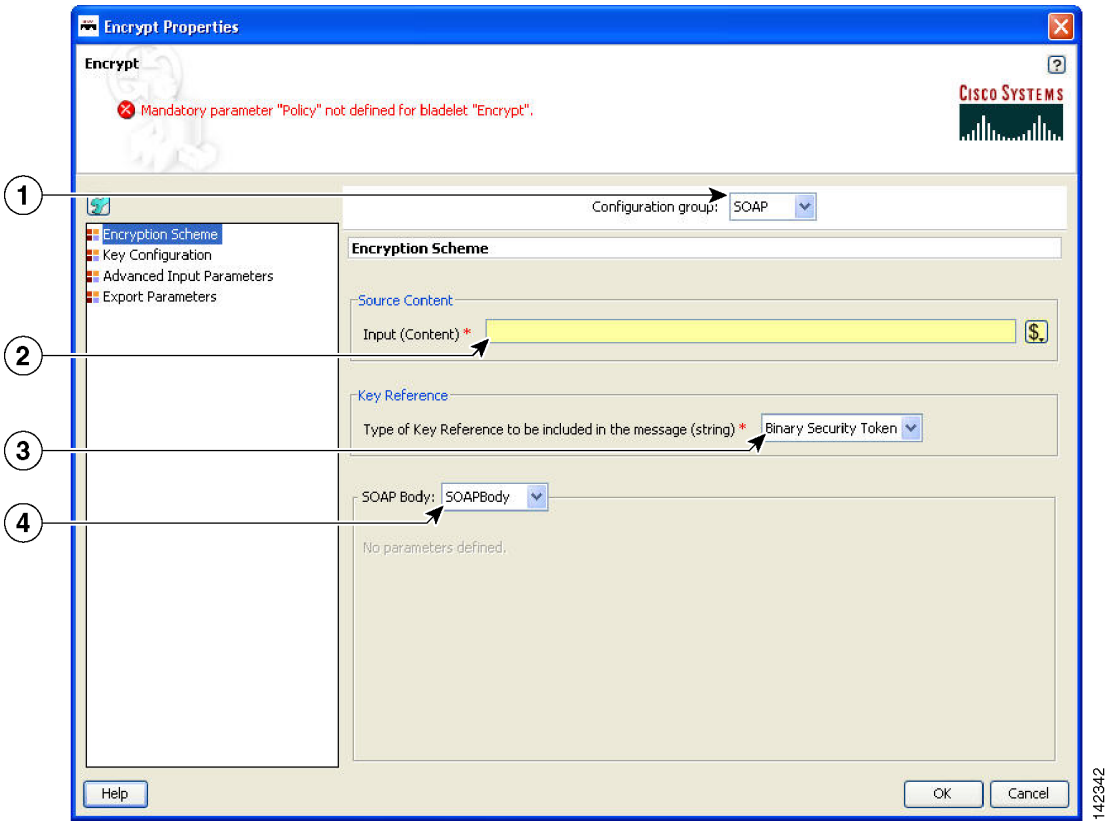


1	Policy Based	Reference of the encryption policy. The key alias in this policy encrypts the symmetric key, regardless of the resource URI that may be configured in this policy. Must already be configured on the AMC server.

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

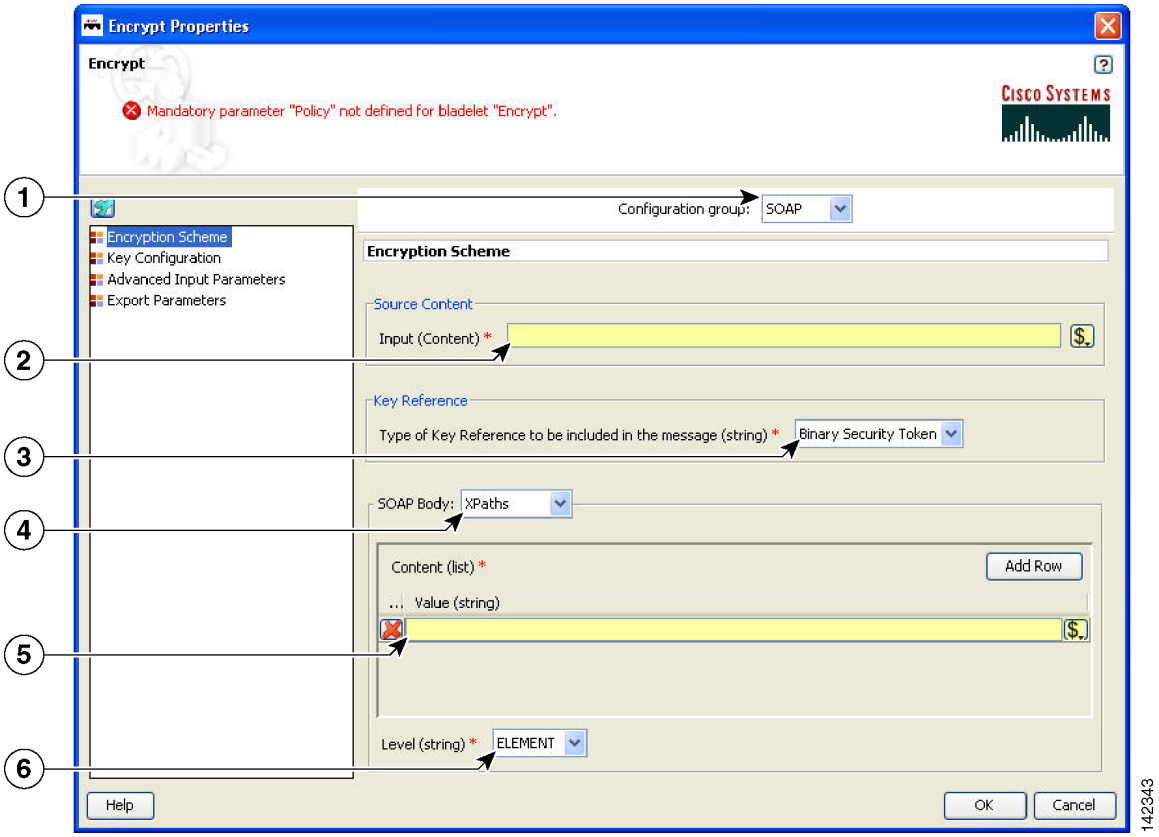
- SOAP (Figure 3-87 to Figure 3-89)
- XML (Figure 3-90 and Figure 3-91)
- Non-XML (Figure 3-92 and Figure 3-93)

Figure 3-87 Encrypt Properties Window—Encryption Scheme, SOAP, SOAPBody



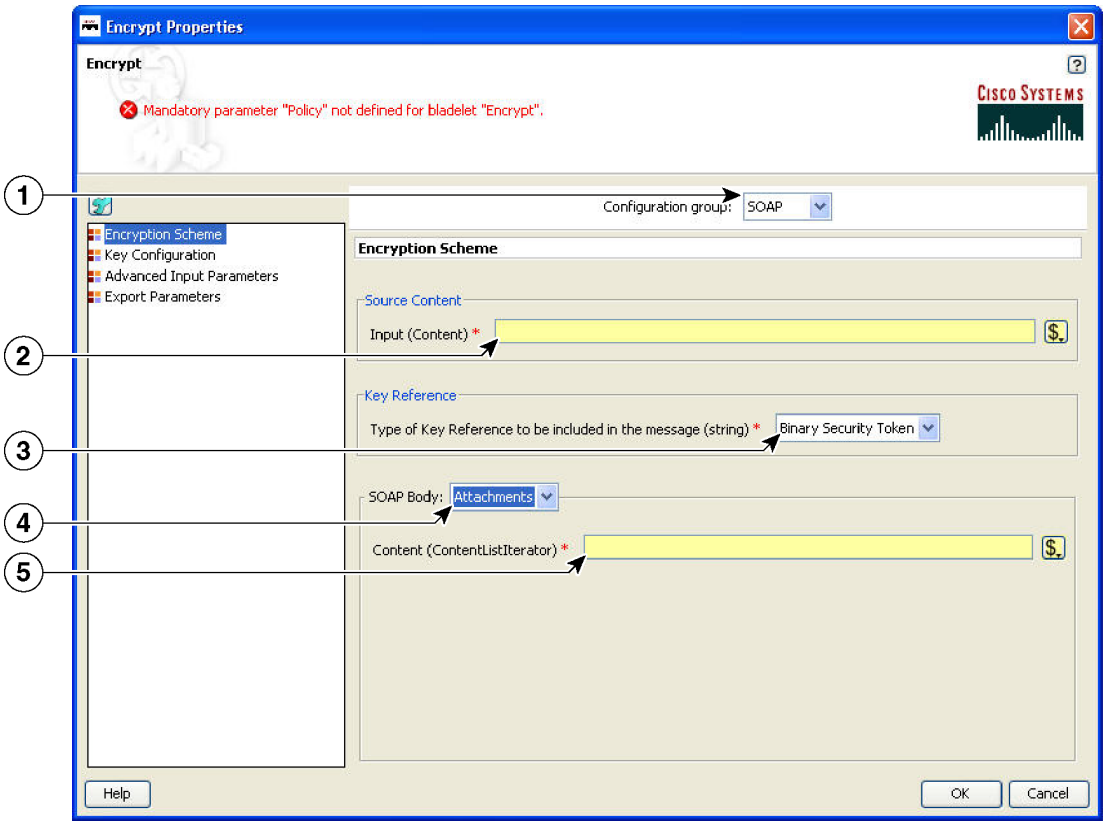
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	Content to be encrypted—XML or SOAP content containing the data that needs to be encrypted.
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary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

Figure 3-88 Encrypt Properties Window—Encryption Scheme, SOAP, XPaths



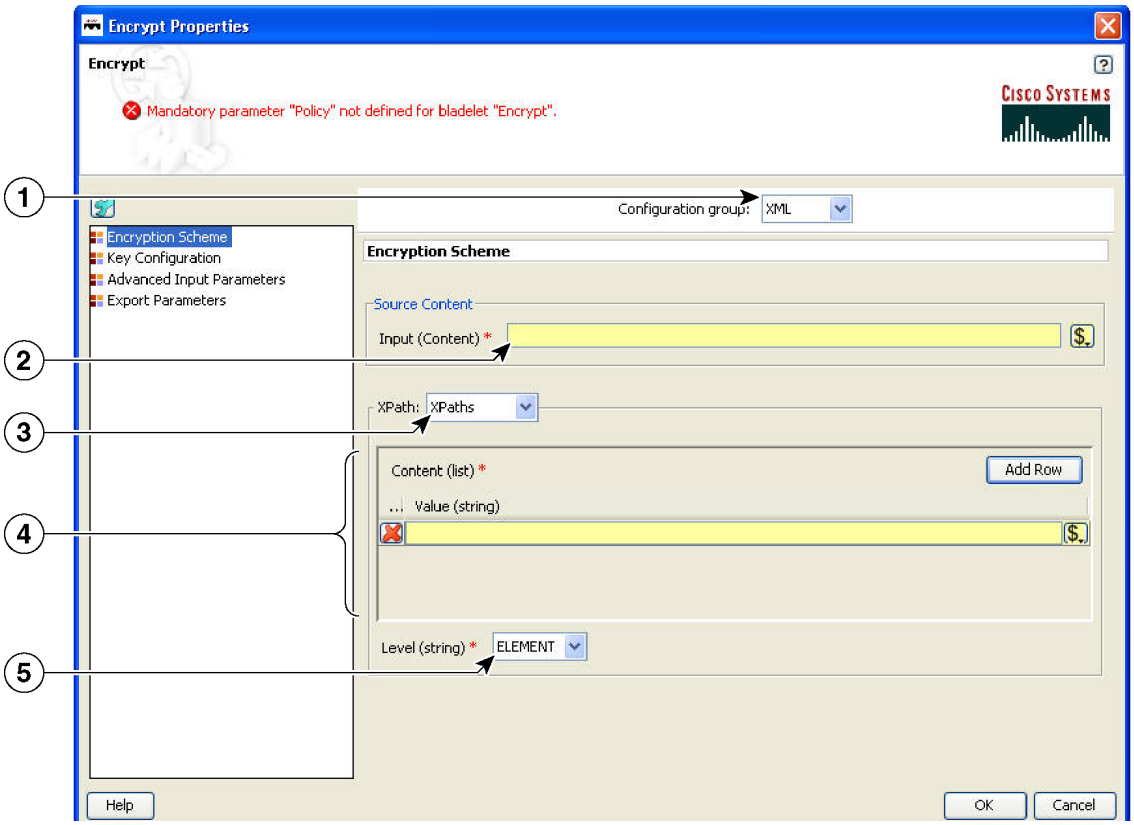
1	Configuration group	Configuration group, set here to SOAP.
2	Input	Content to be encrypted.
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • Subject Key Identifier • Issuer and Serial #
4	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 3-89 Encrypt Properties Window—Encryption Scheme, SOAP, Attachments



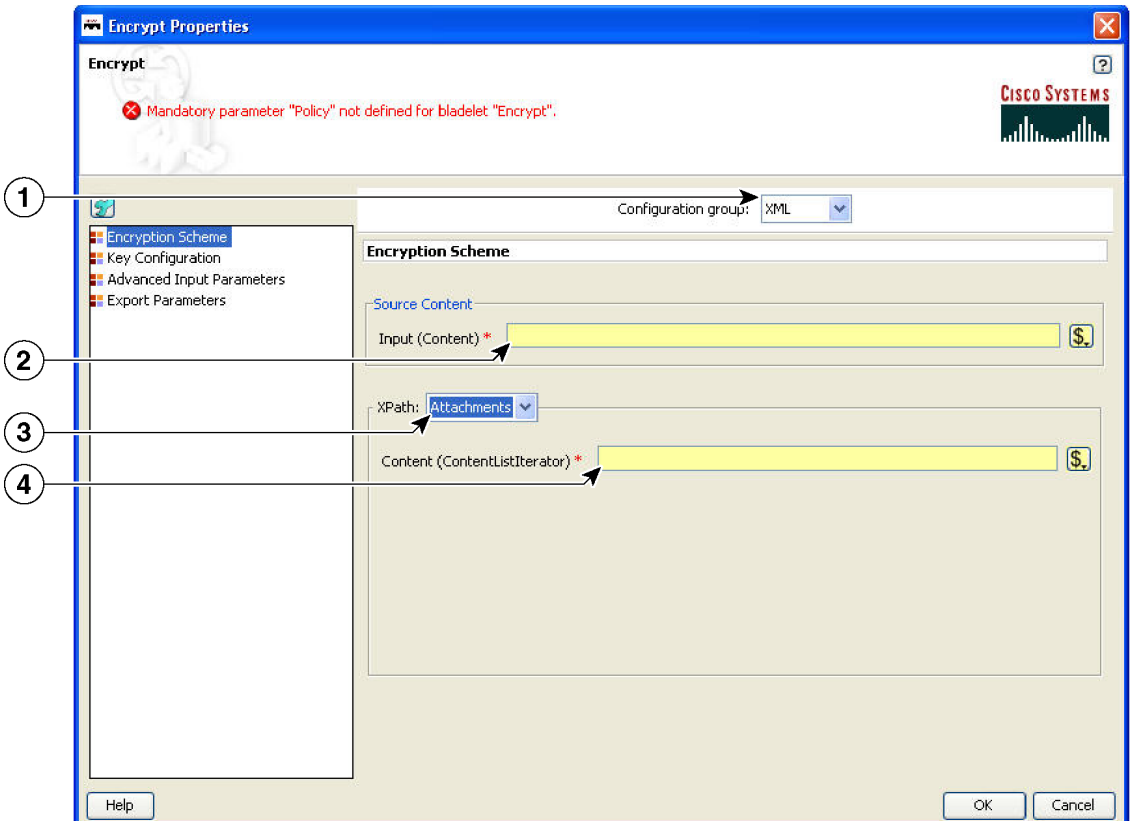
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	Content to be encrypted. Set if the encryption scheme is an attachment.
3	Key Reference	The Type of Key Reference to be included in the message here is Binary Security Token. Type of Key Reference choices: • Binary Security Token • 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 3-90 Encrypt Properties Window—Encryption Scheme, XML, XPaths



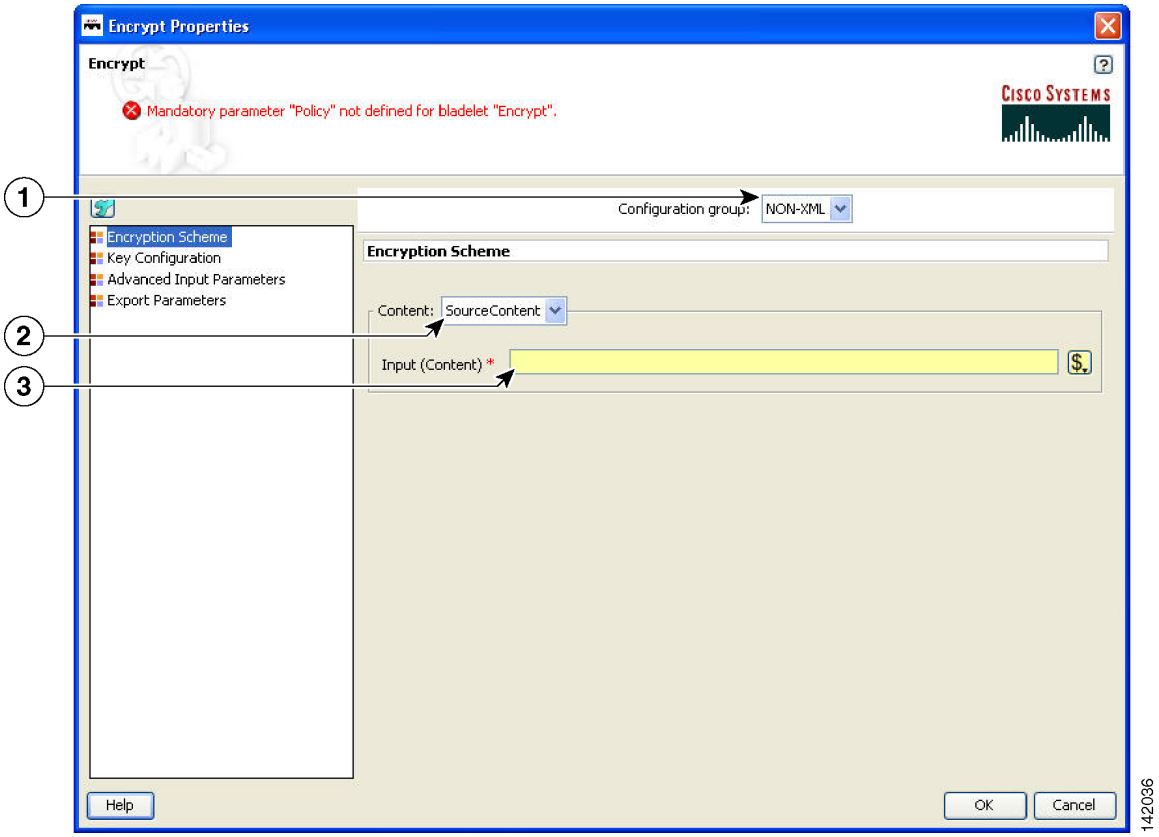
1	Configuration Group	Configuration group, set here to XML.
2	XPaths	XPath Locations of the elements to be encrypted in the XML message. You may add multiple XPath strings using the "Add Row" button.
3	Content	Content to be encrypted.
4	Level	Level: • Element—Whole XML element needs to be encrypted, 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 3-91 Encrypt Properties Window—Encryption Scheme, XML, Attachments



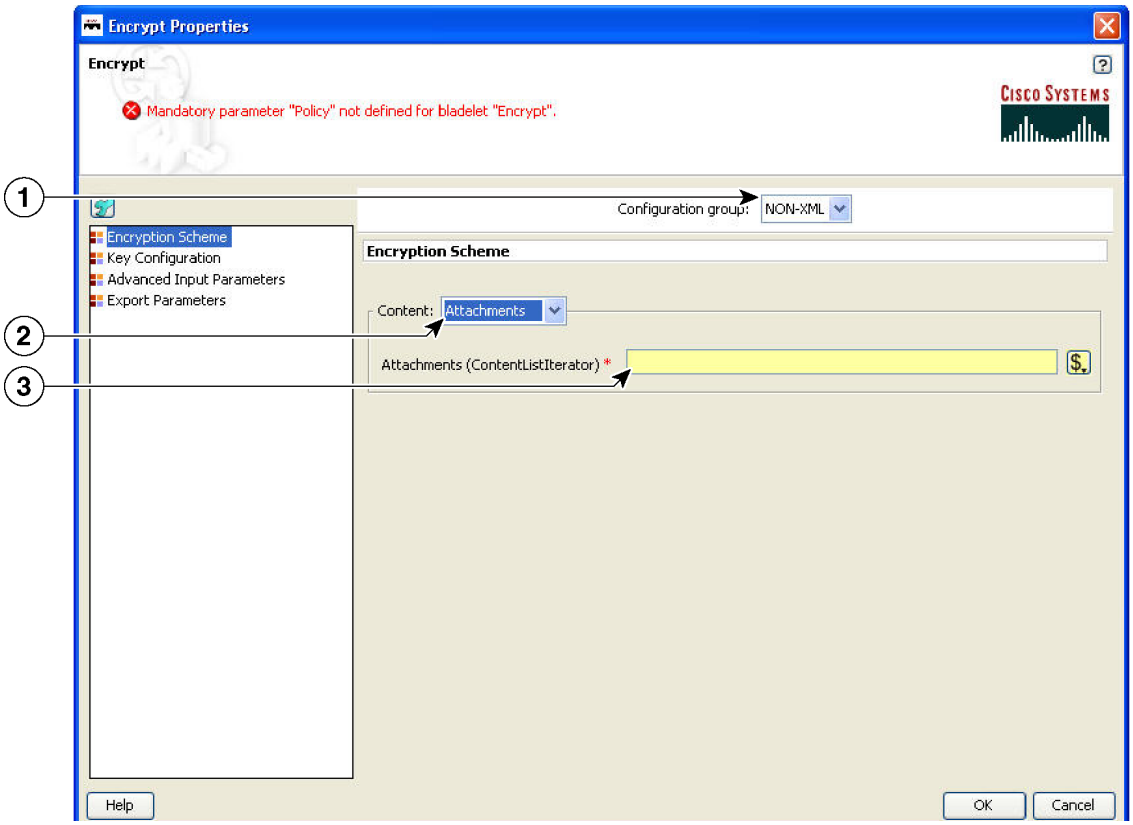
1	Configuration Group	Configuration group, set here to XML.
2	Attachments	List of attachments to be encrypted. This list is the output of a MIME-Extract Bladelet that should have preceded the Encryption Bladelet and extracted the attachments to be encrypted.
3	Content	Content to be encrypted.
4	Input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

Figure 3-92 Encrypt Properties Window—Encryption Scheme, Non-XML, Source Content



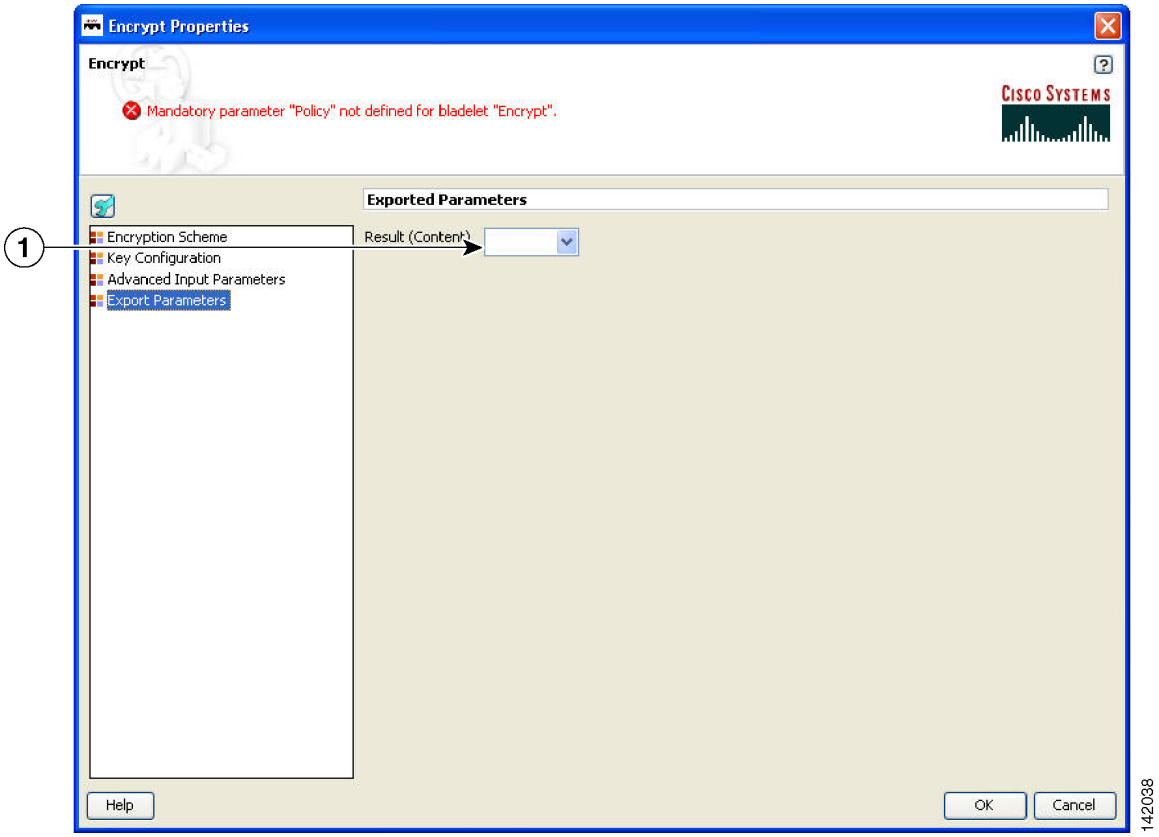
1	Configuration Group	Configuration group, set here to Non-XML.
2	Source Content	Non-XML content to be encrypted, if the content is of type non-XML.
3	Input	Content to be encrypted. This may be an XML or SOAP content containing the data that needs to be encrypted.

Figure 3-93 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 3-94 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.

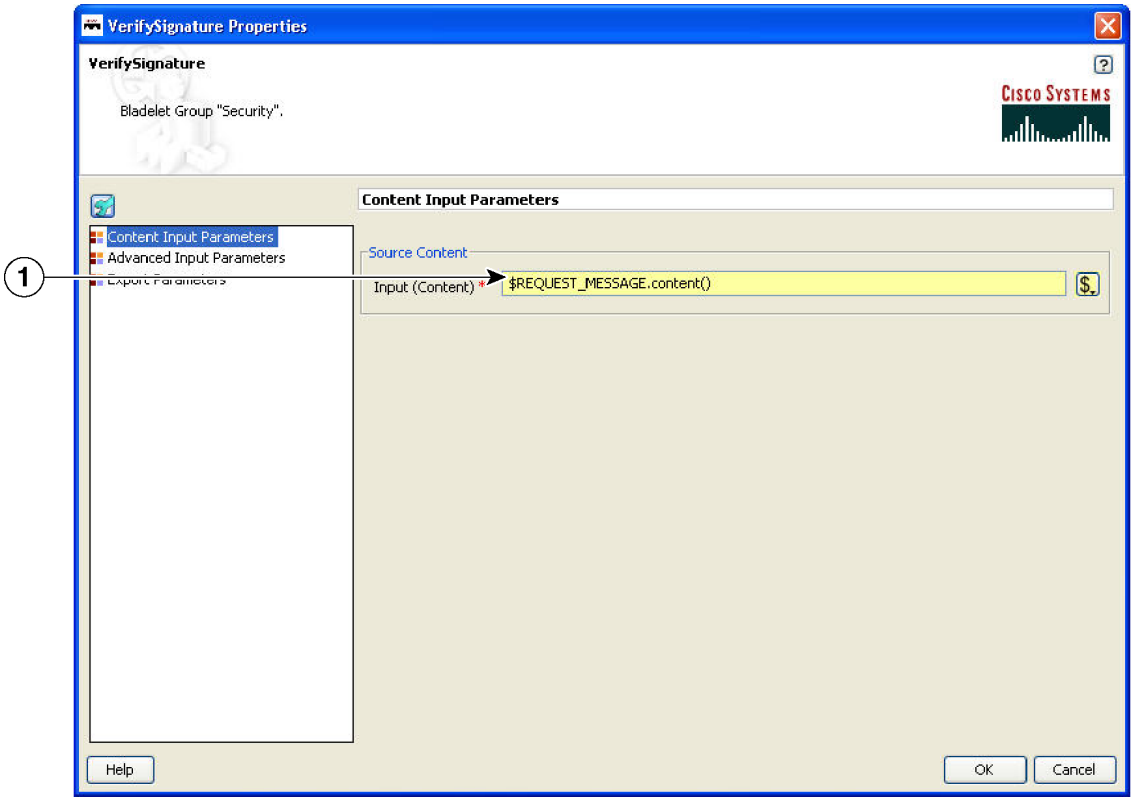
Details

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.

Note Each required field in the Bladelet Properties window is marked by a red asterisk. Until all required fields are completed with the correct value, an error message appears on top of the Bladelet Properties window to indicate which field remains to be completed or indicates that there is a parameter type mismatch and so on before the Bladelet is completely configured.

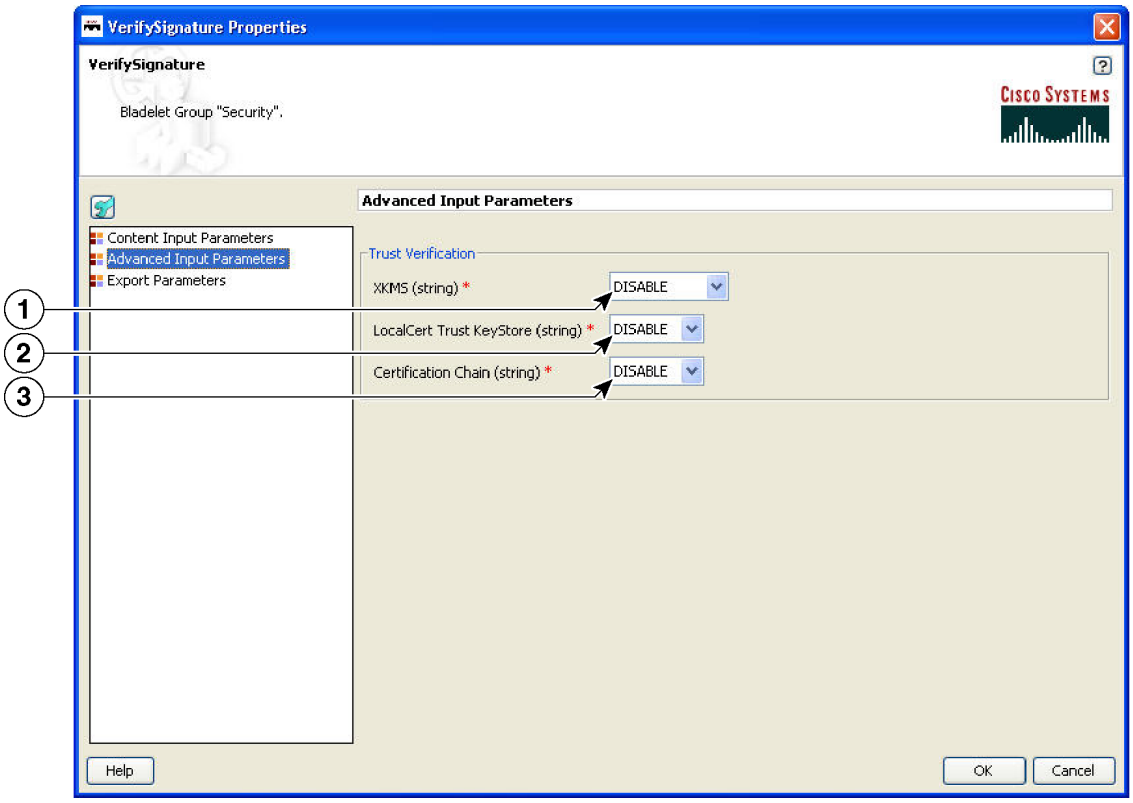
Figure 3-95 Verify Signature Properties Window—Content to Verify



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().

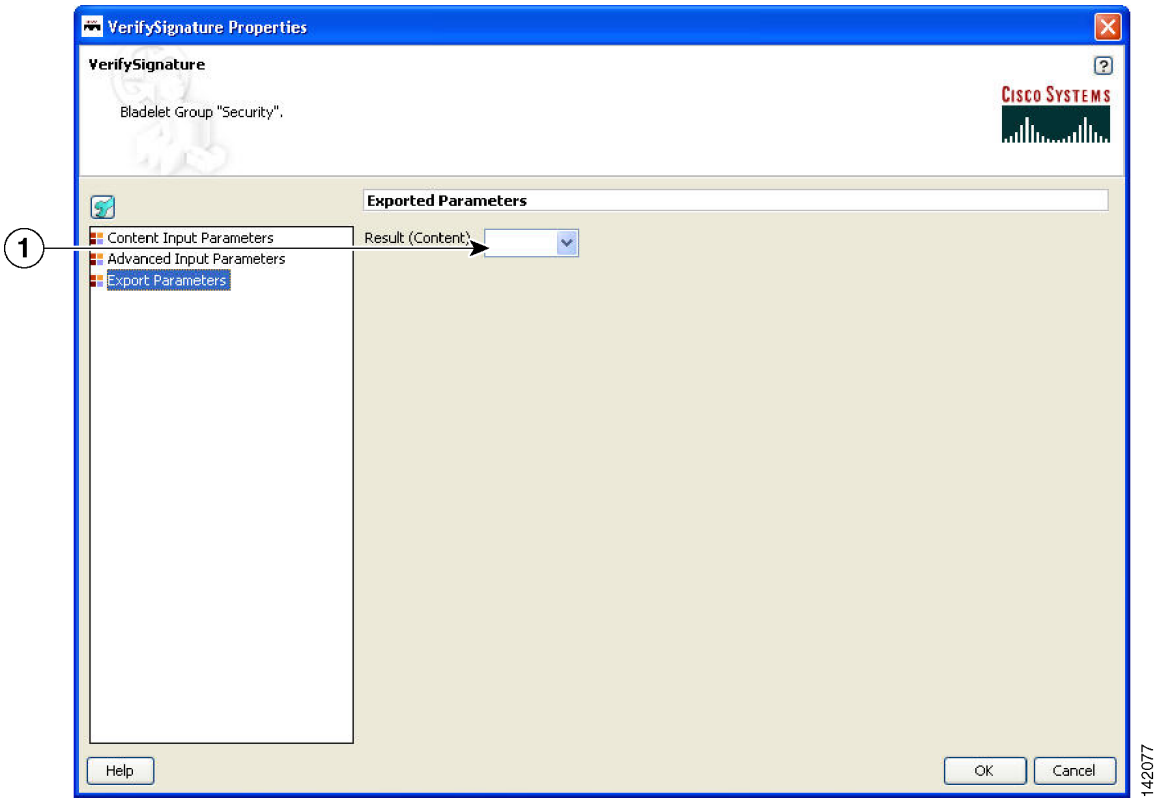
If Local Certificate Trust and/or Certificate Chain Verification is enabled, then configure the local Trust Store. The Certificate found in the Signature is expected to be found in the local Trust Store.

Figure 3-96 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 3-97 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.

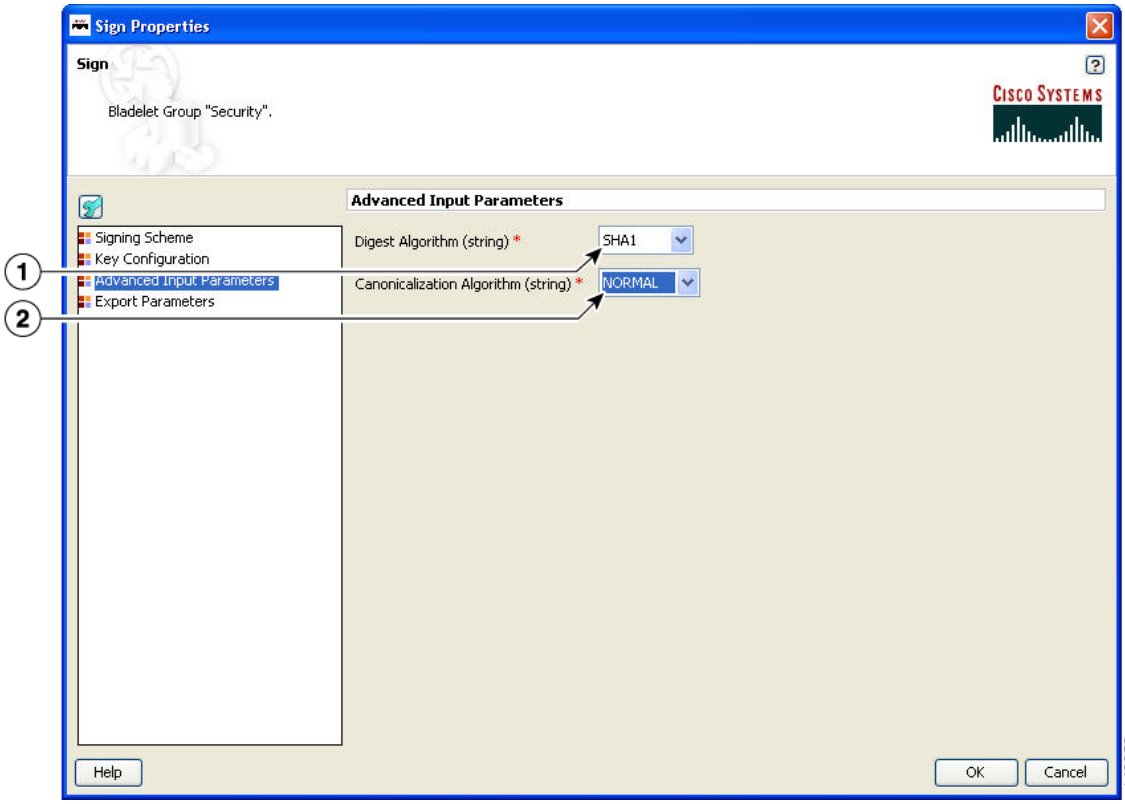
Details

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.

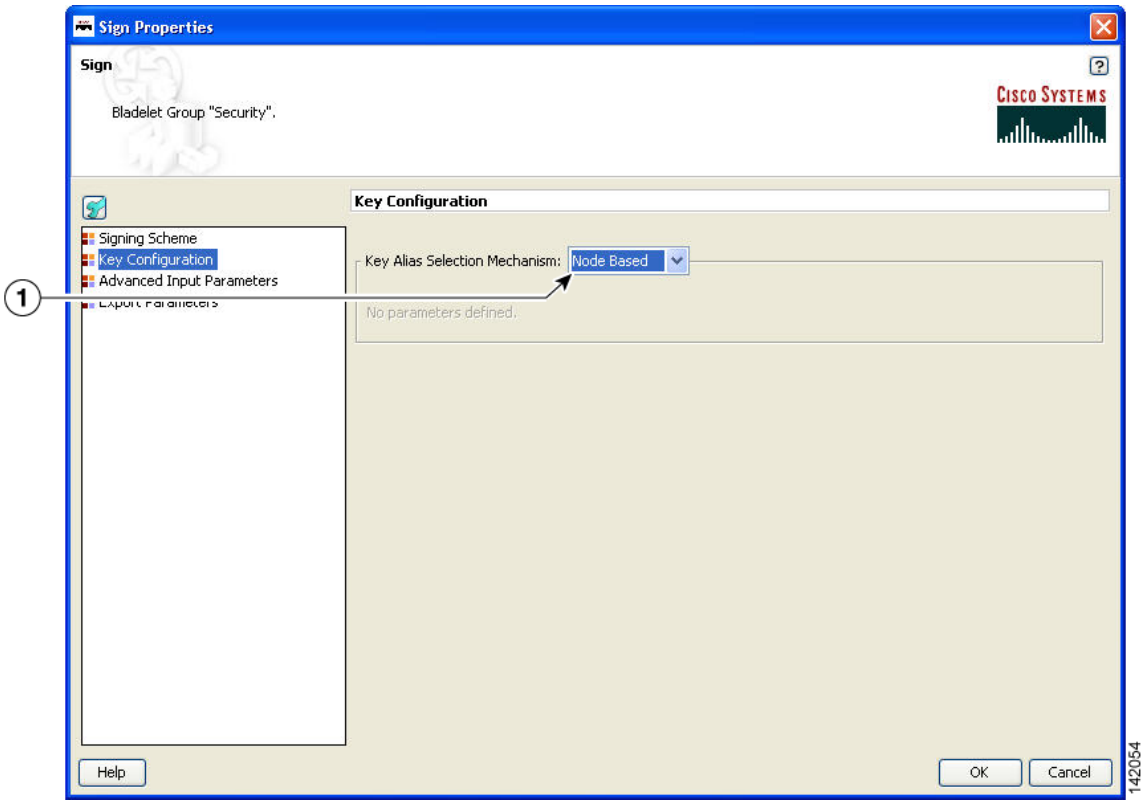
For a non-MIME message, the original message is modified inline and hence no need to configure the export parameter.

Figure 3-98 Sign Properties Window—Advanced Input Parameters



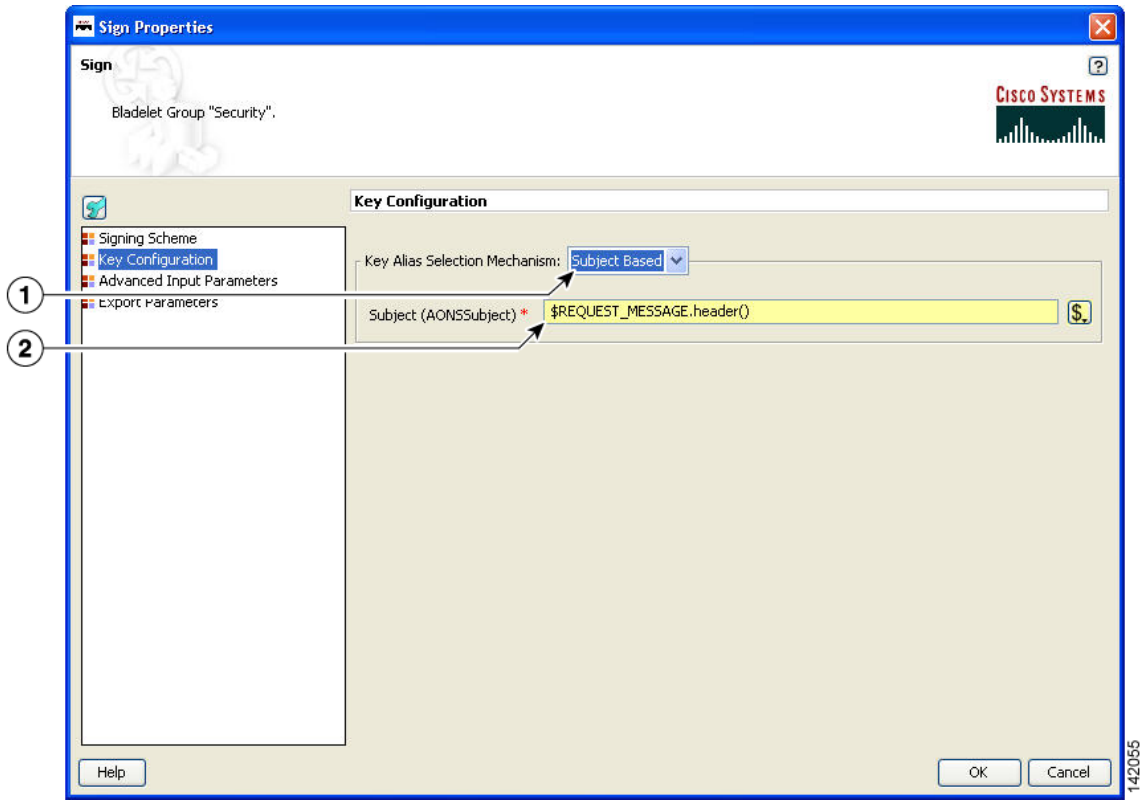
1	Digest Algorithm	Type of digest algorithm to be used to create digital signature.
2	Canonicalization Algorithm	Type of canonicalization algorithm to be used to create digital signature.

Figure 3-99 Sign Properties Window—Key Configuration, Node Based



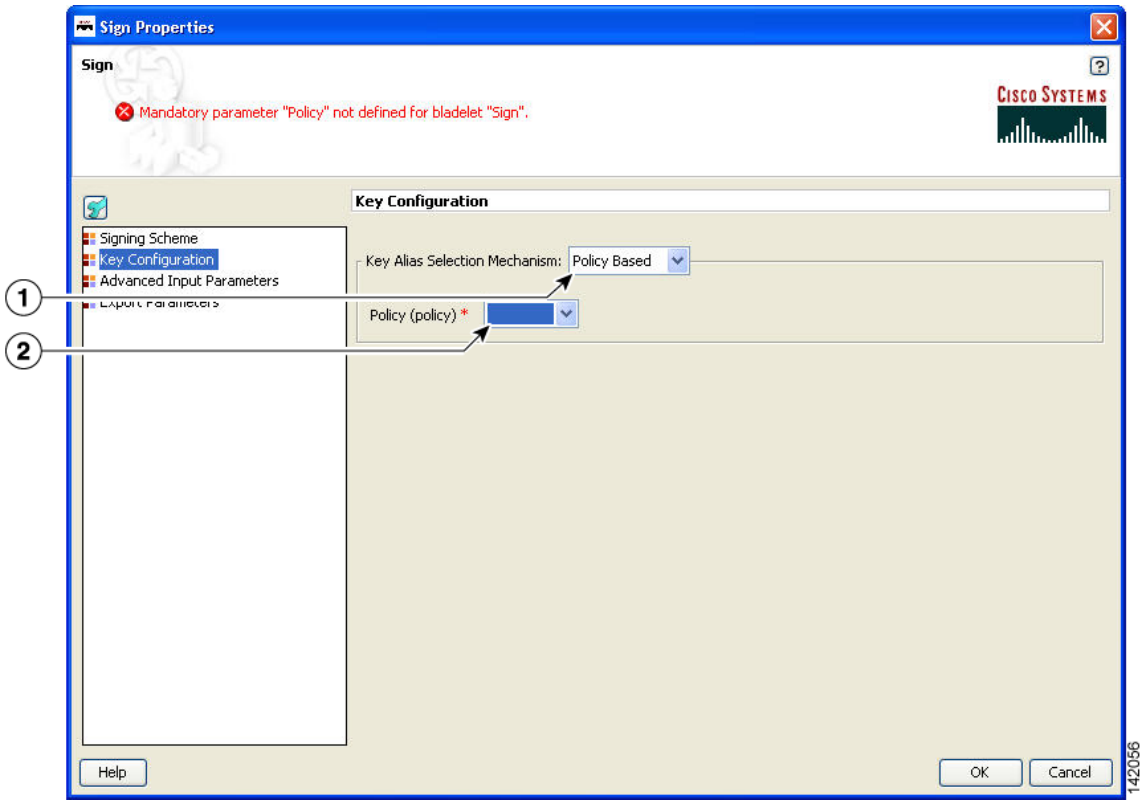
1	Node Based	Type of key configuration. Use node-based key alias instead of any signing policy. Uses the AON key for signing. Must already be configured in AMC-Keystore.

Figure 3-100 Sign Properties Window—Key Configuration, Subject Based



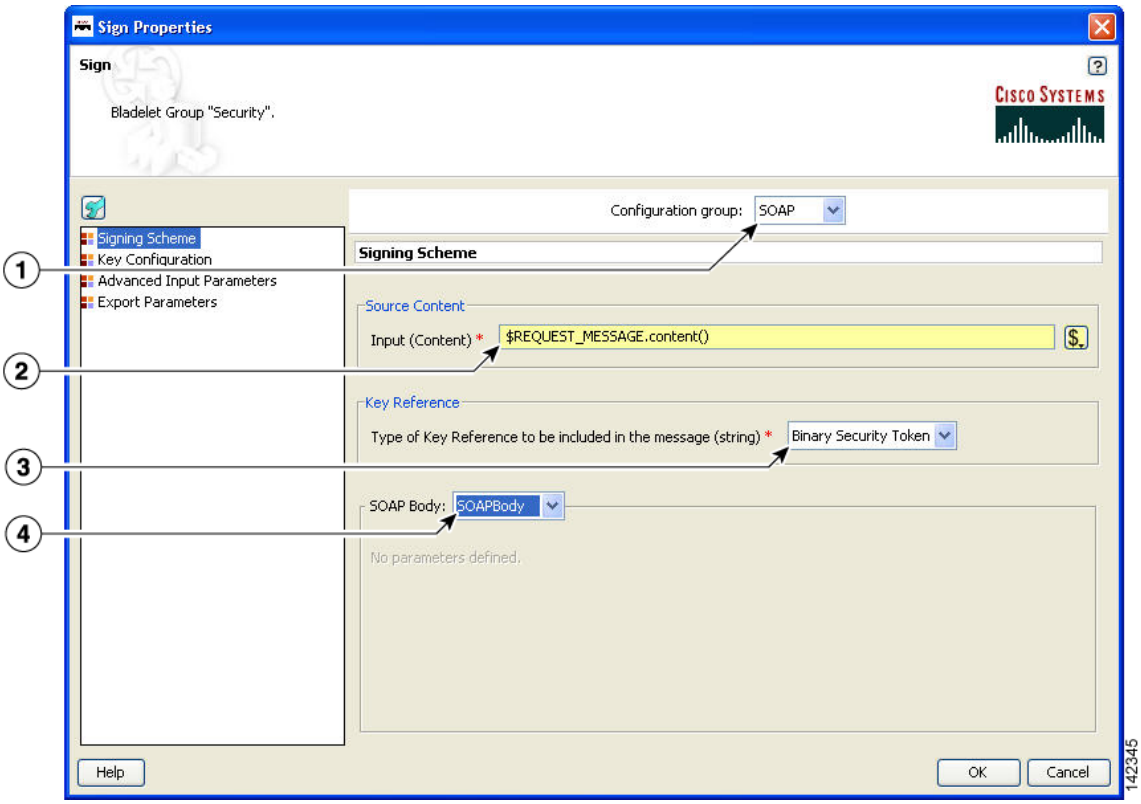
1	Subject Based	Type of key configuration. Key alias is extracted based on the value of the AONSSubject PEP variable.
2	Subject	Select subject form the auto complete text field.

Figure 3-101 Sign Properties Window—Key Configuration, Policy Based



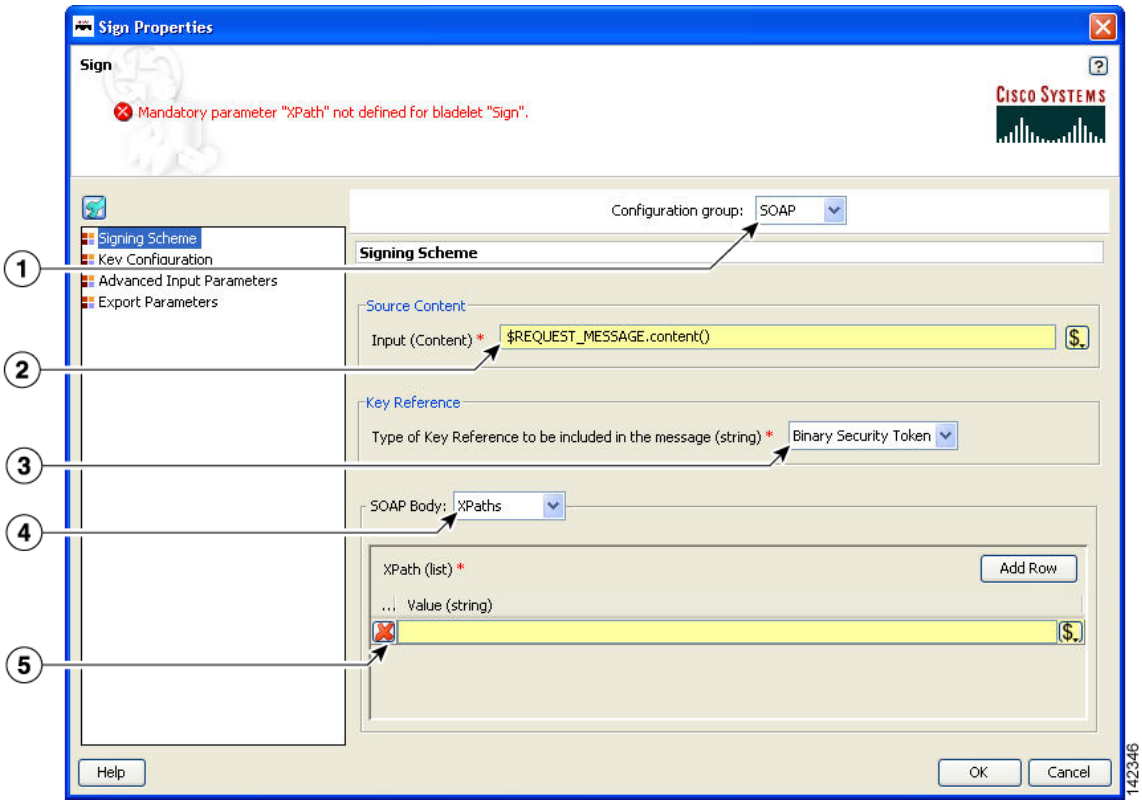
1	Policy Based	Type of key configuration. Signing policy containing configuration to the key alias.
2	Policy	Policy. Must already be configured on the AMC server.

Figure 3-102 Sign Properties Window—Signing Scheme, SOAP, SOAPBody



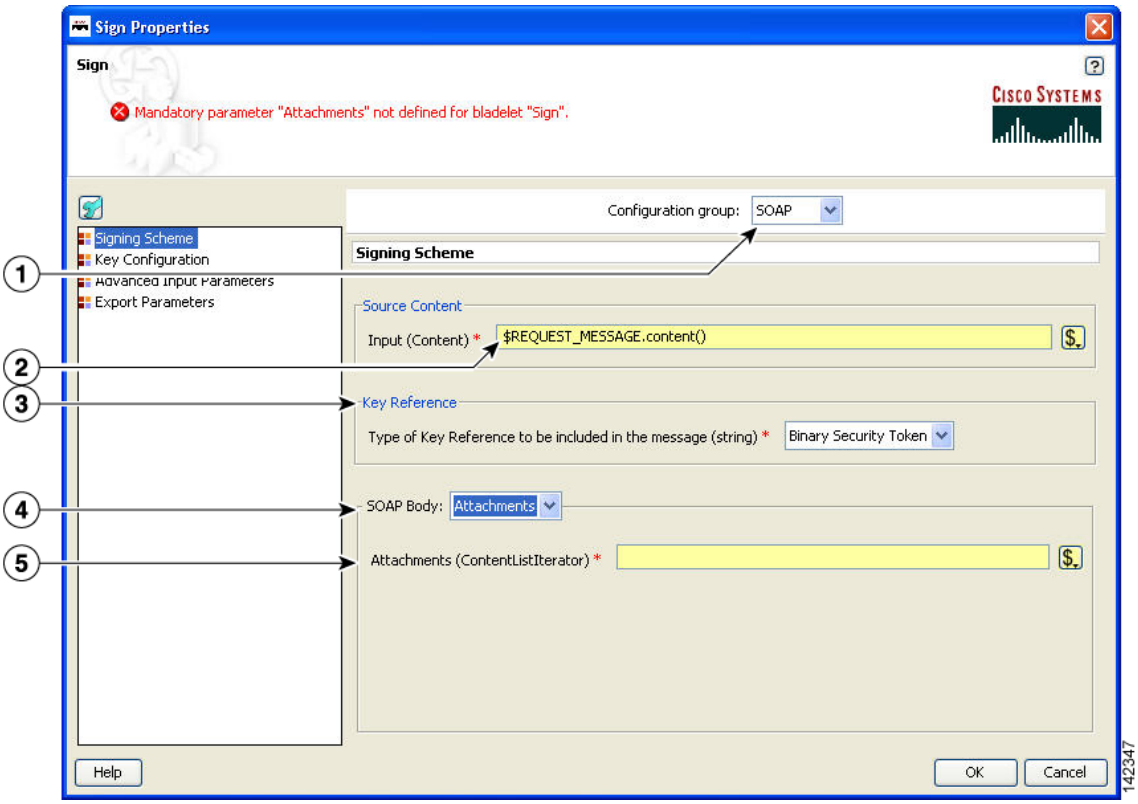
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	Source-content input. If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().
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 3-103 Sign Properties Window—Signing Scheme, SOAP, XPaths



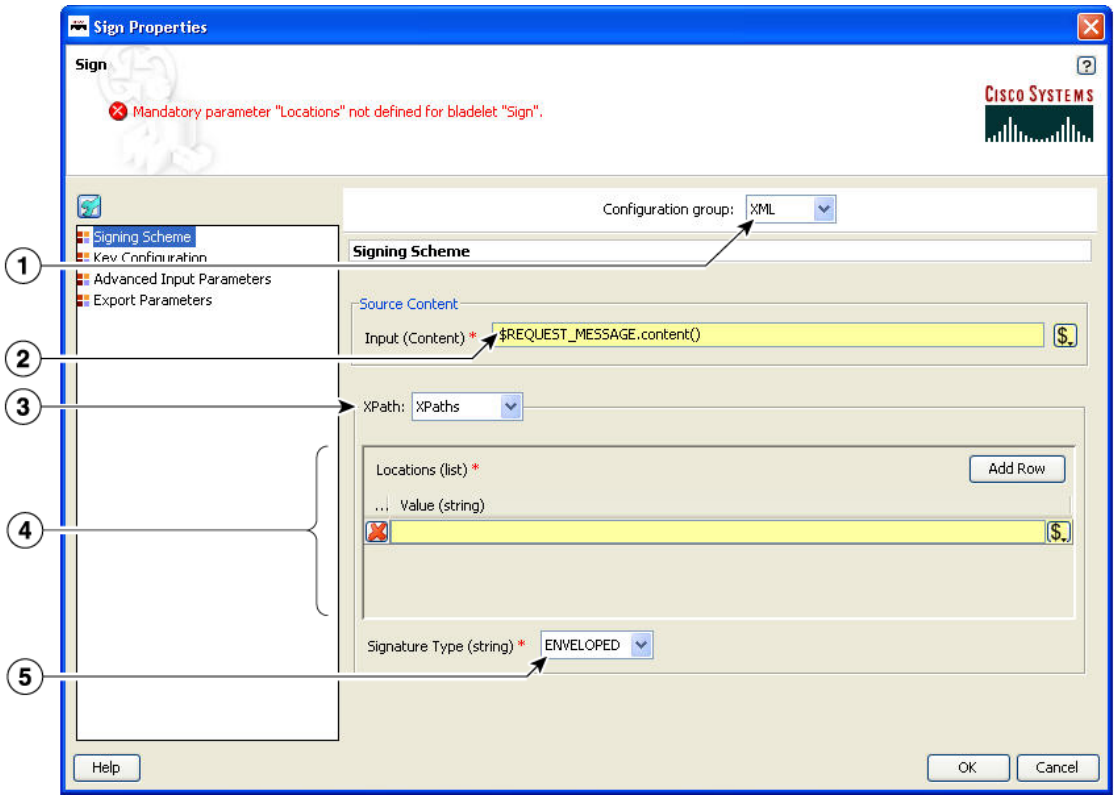
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().
3	Key Reference	The Type 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 3-104 Sign Properties Window—Signing Scheme, SOAP, Attachments



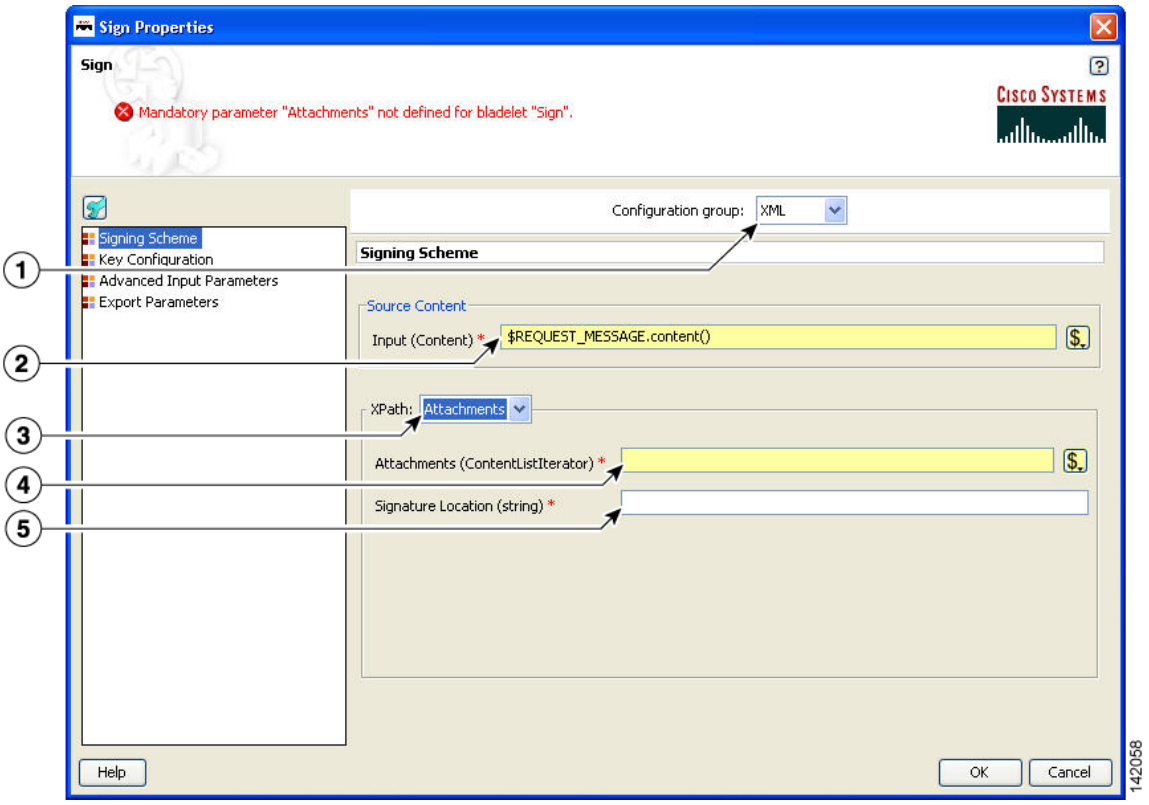
1	Configuration Group	Configuration group, set here to SOAP.
2	Input	If you are operating on request message then usually this value is bound to REQUEST_MESSAGE.content(). If you are operating on response message then the value of the PEP variable is RESPONSE_MESSAGE.content().
3	Key Reference	The Type 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 3-105 Sign Properties Window—Signing Scheme, XML, XPaths



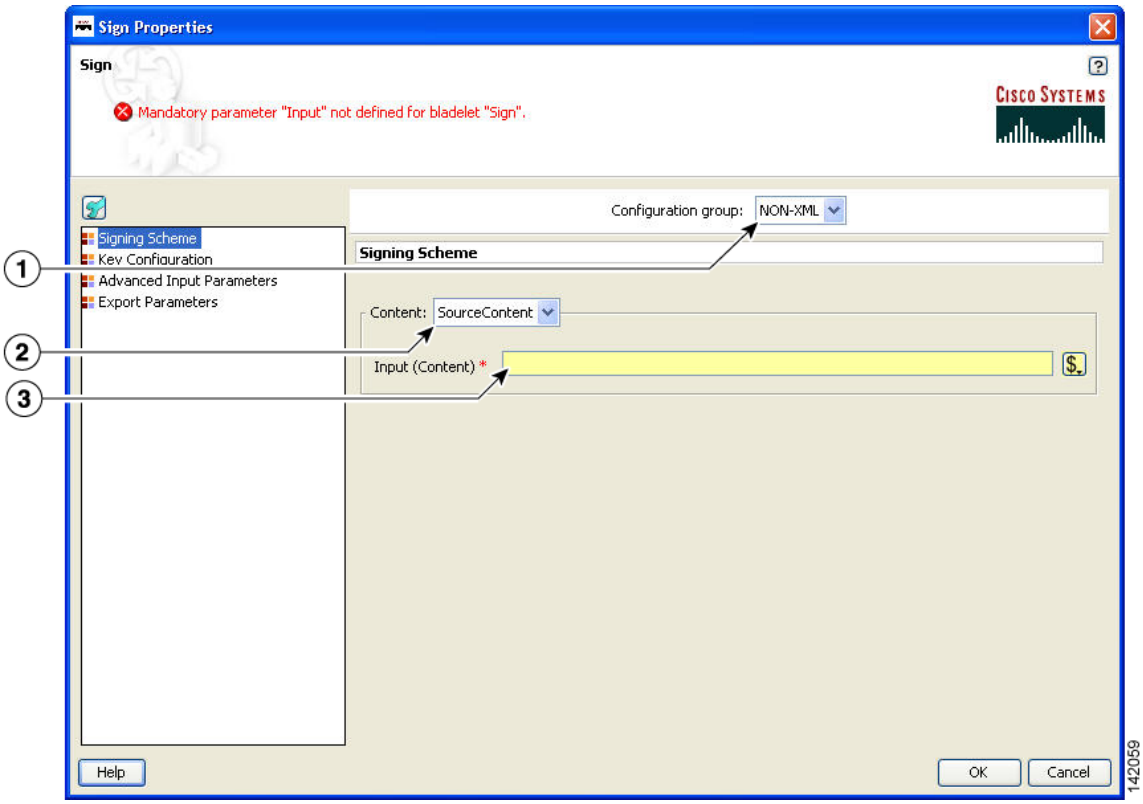
1	Configuration Group	Configuration group, set here to XML.
2	XPaths	List of Xpath expressions that are used to sign relevant portions on soap message
3	XPath Locations	One or more XPath locations (in string form).
4	Signature Type	Signature type: Enveloped or Enveloping.
5	Input	If you are operating 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 3-106 Sign Properties Window—Signing Scheme, XML, Attachments



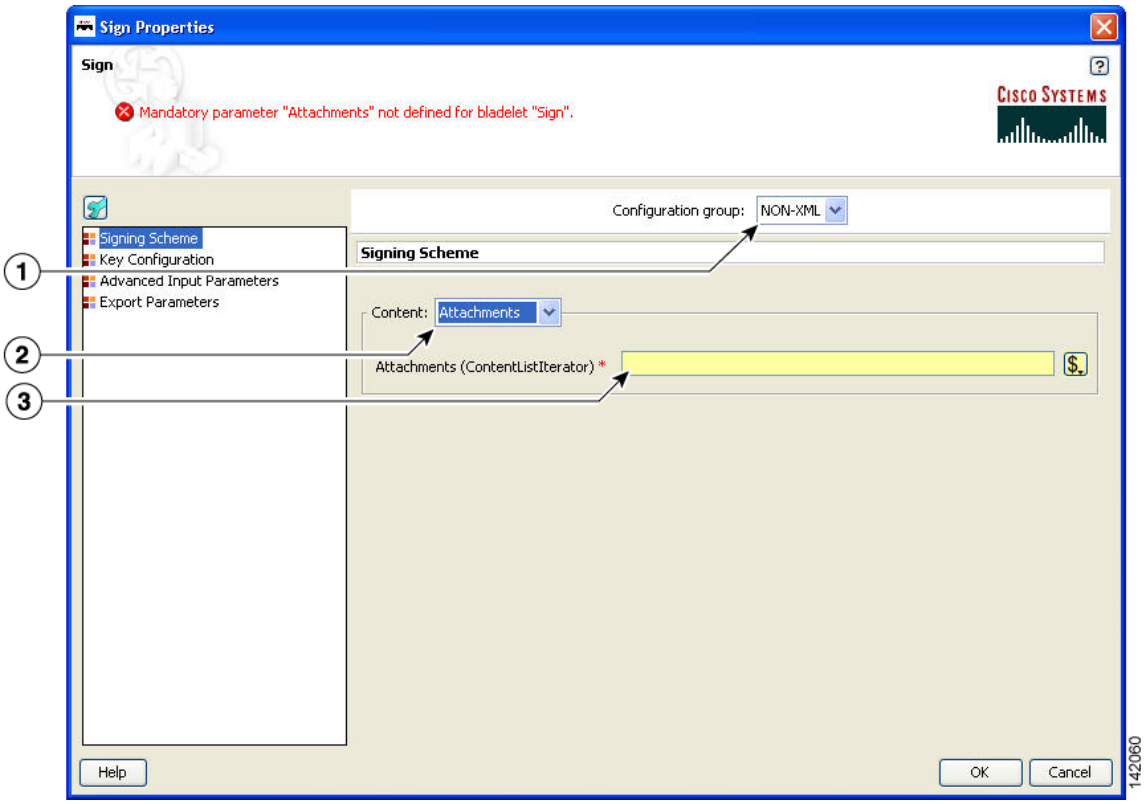
1	Configuration Group	Configuration group, set here to XML.
2	Attachments	Attachments of a multipart message, where root part is SOAP message, to be signed.
3	Attachments List	Selected attachments.
4	Signature Location	Location of the signature.
5	Input	If you are operating on request message then usually 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 3-107 Sign Properties Window—Signing Scheme, Non-XML, Source Content



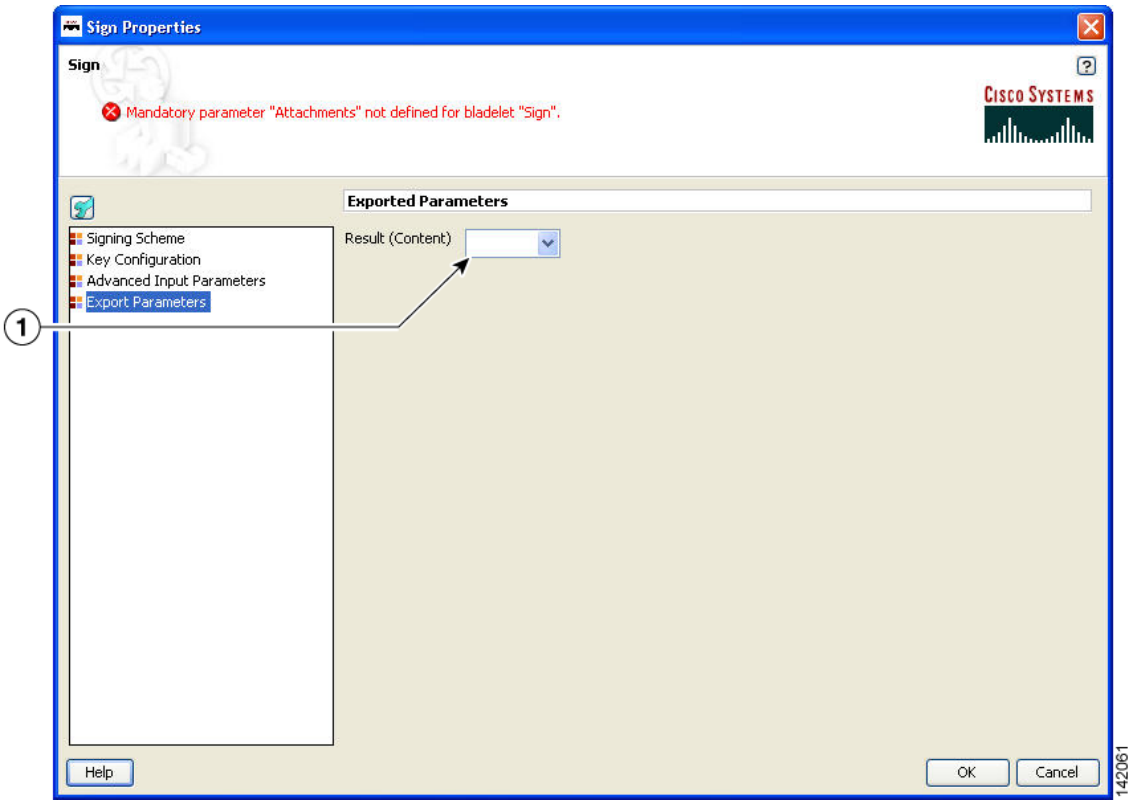
1	Configuration Group	Configuration group, set here to Non-XML.
2	Source Content	Base content of the original MIME message.
3	Input	Input content.

Figure 3-108 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 3-109 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.

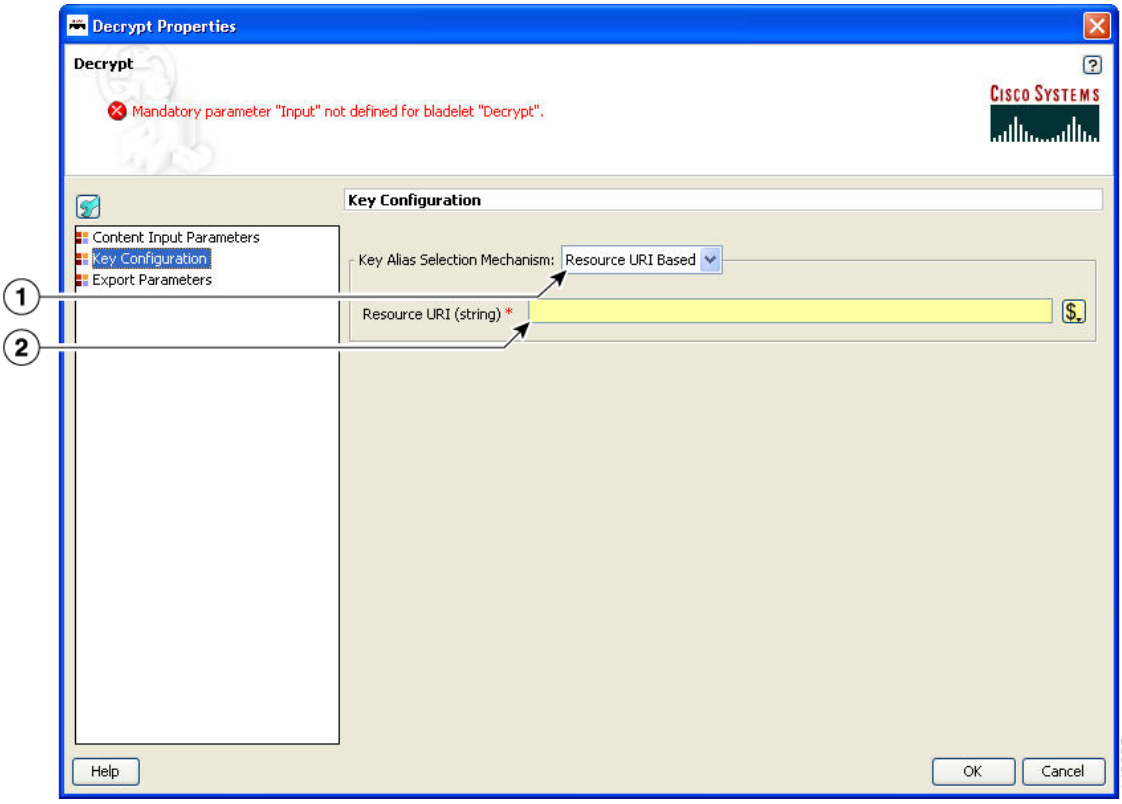
Prerequisites and Dependencies

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

Details

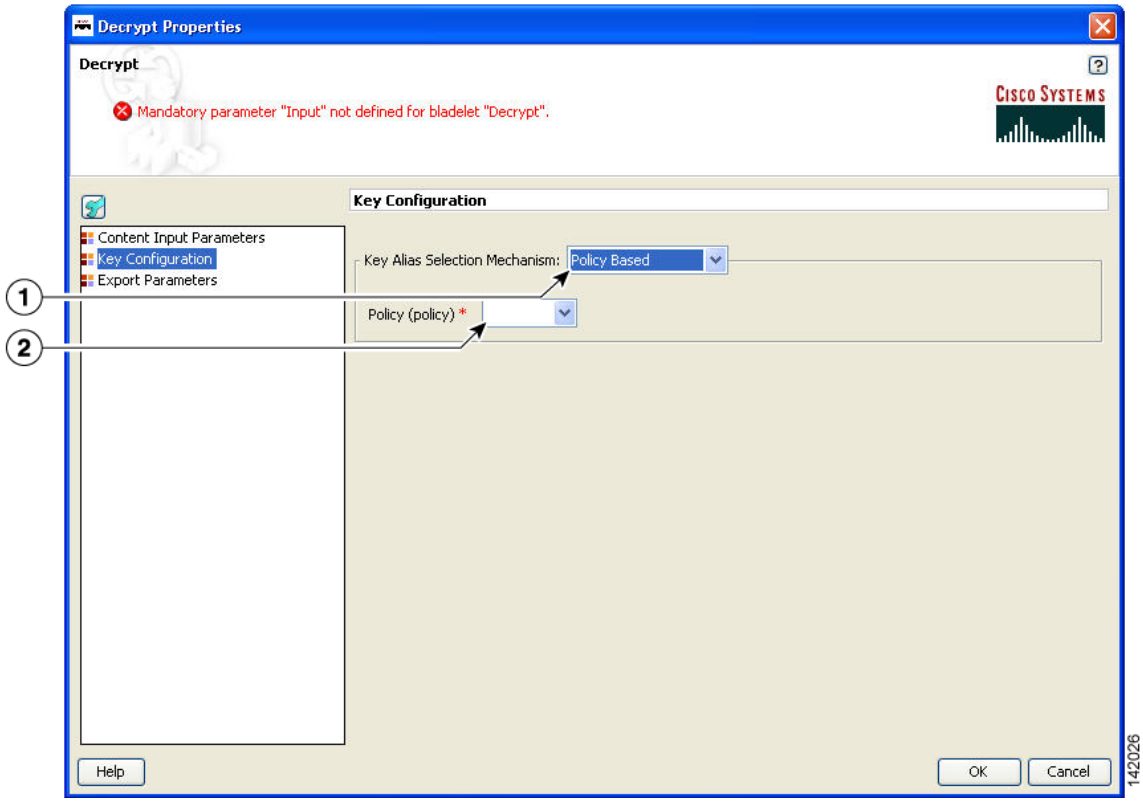
The Decrypt Bladelet decrypts SOAP messages containing data that has been encrypted with a symmetric key that has also been encrypted using an asymmetric public key. Given the private key of the recipient message as an input parameter, this Bladelet moves the CPU-intensive decryption operation to AON. 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 3-110 Decrypt Properties Window—Key Configuration, Resource URI Based



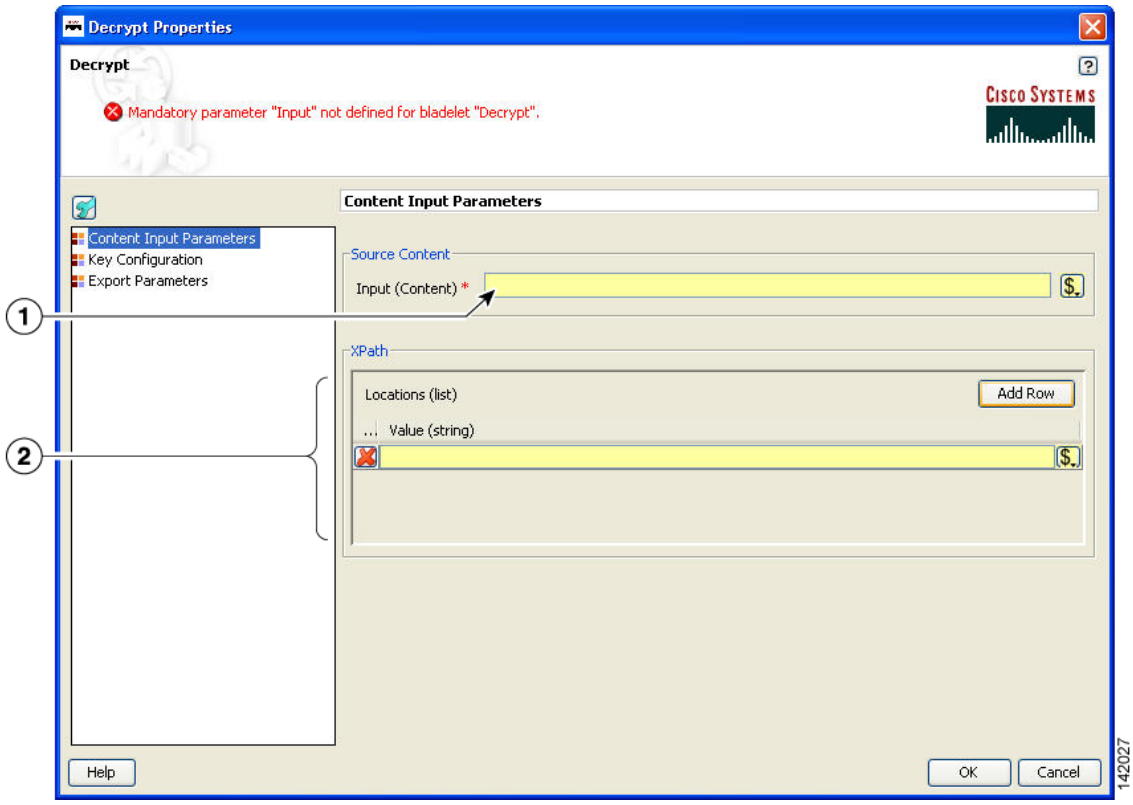
1	Resource URI Based	Resource URI Based is set as the key alias selection method.
2	Resource URI	URI of the intended recipient of this encrypted message. The key alias corresponding to this resource decrypts the symmetric key. Must already be configured on the AMC server.

Figure 3-111 Decrypt Properties Window—Key Configuration, Policy Based



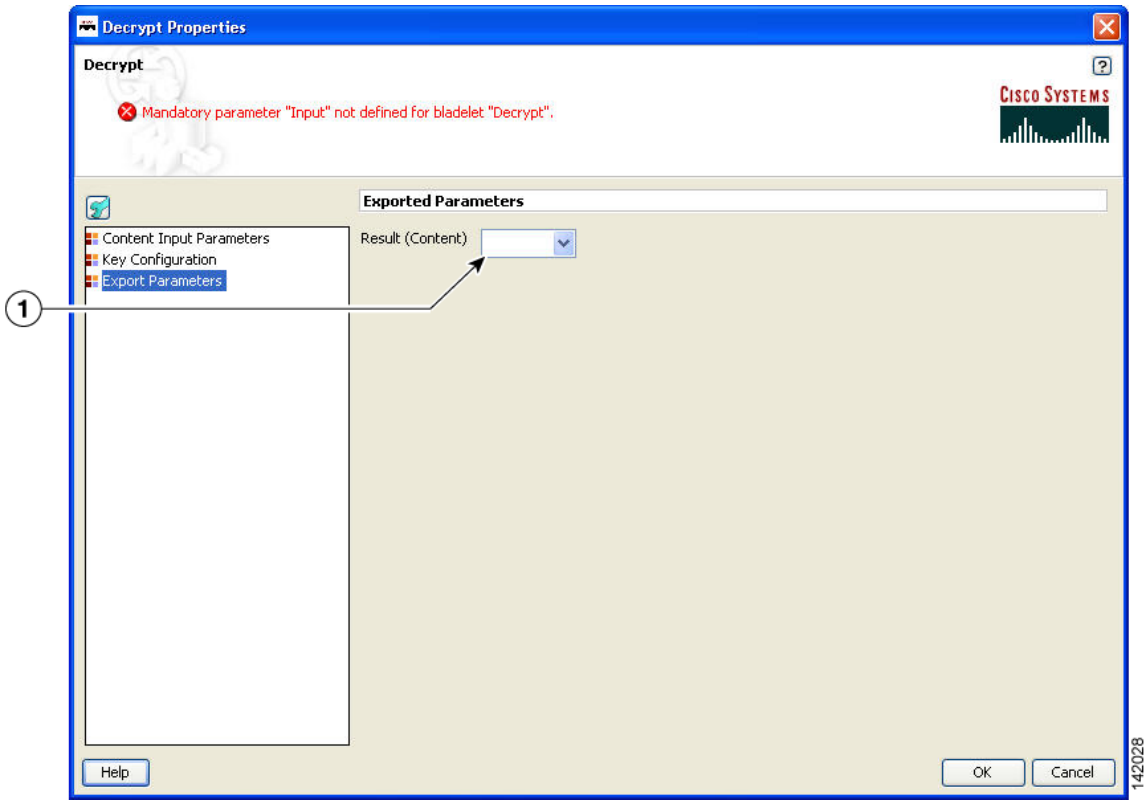
1	Policy Based	Policy Based is set as the key alias selection method.
2	Policy	Reference of the decryption policy. The key alias in this policy decrypts the symmetric key, irrespective of the resource URI that may be configured in this policy. Must already be configured on the AMC server.

Figure 3-112 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 3-113 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.”

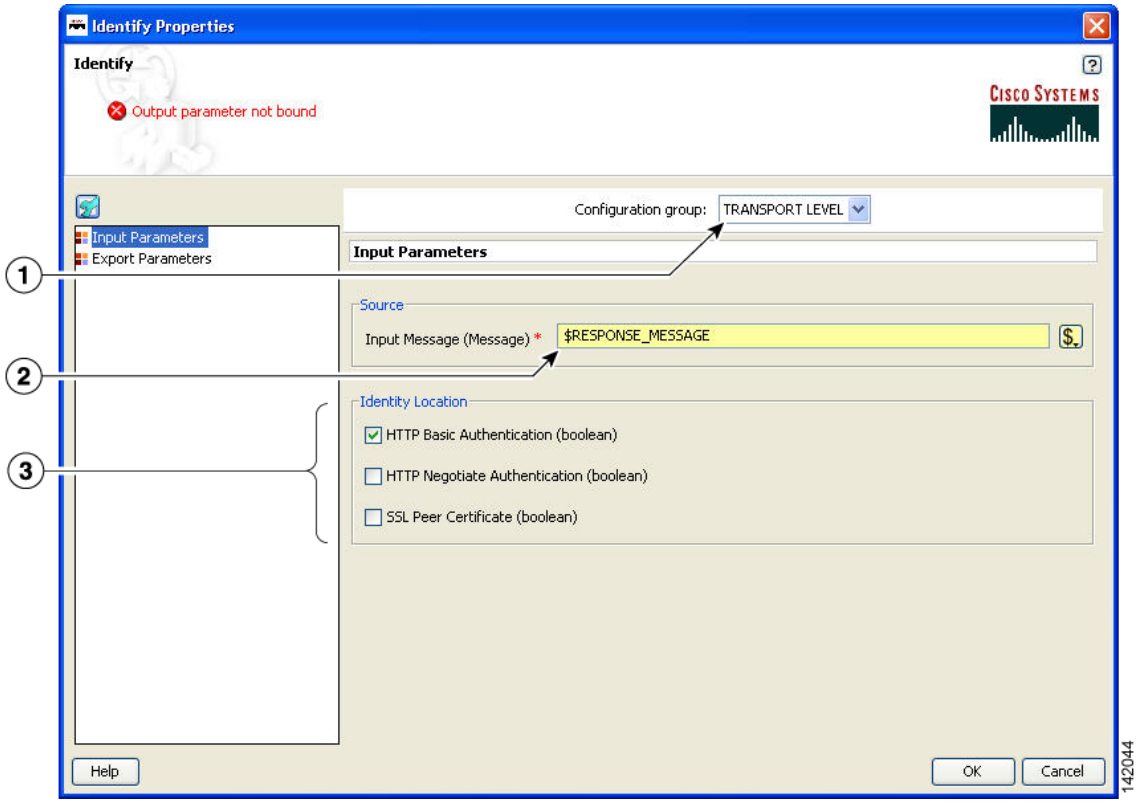
When there is no identity extracted, no HTTP-level challenge or soapfault is generated. Those message can be generated only by subsequent Bladelets that try to use the identity information for different purposes, such as authenticate and identity verify.

Prerequisites and Dependencies

None.

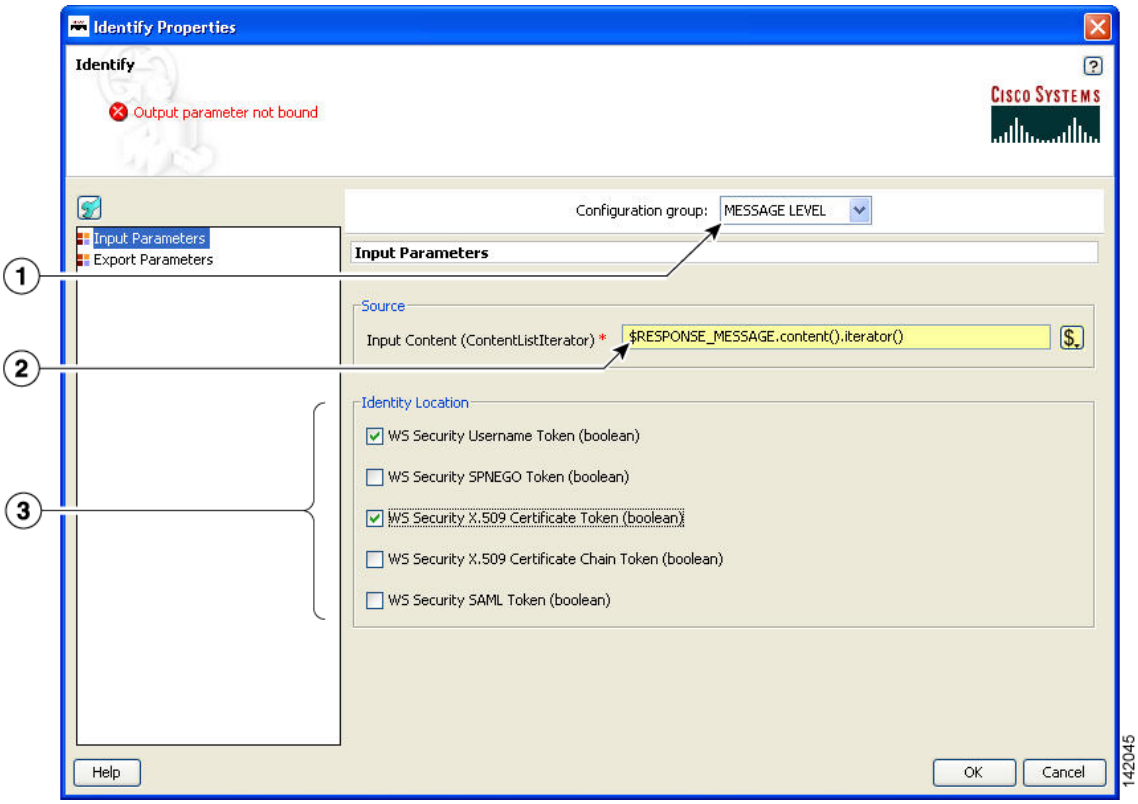
Details

Figure 3-114 Identify Properties Window—Input Parameters, Transport Level Identity



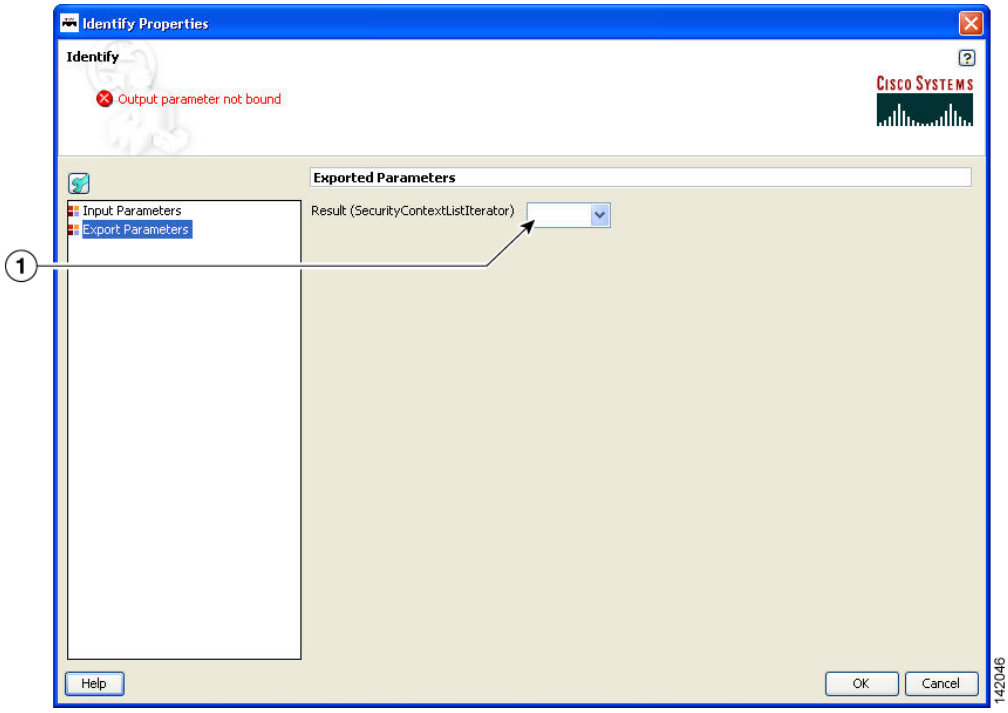
1	Configuration Group	Configuration group, set here to Transport Level.
2	Input Message	Incoming message to extract identity information from.
3	Identity Location	Location from which to extract HTTP information: • HTTP Basic Authentication—Extracts HTTP basic authentication information from incoming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

Figure 3-115 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 3-116 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

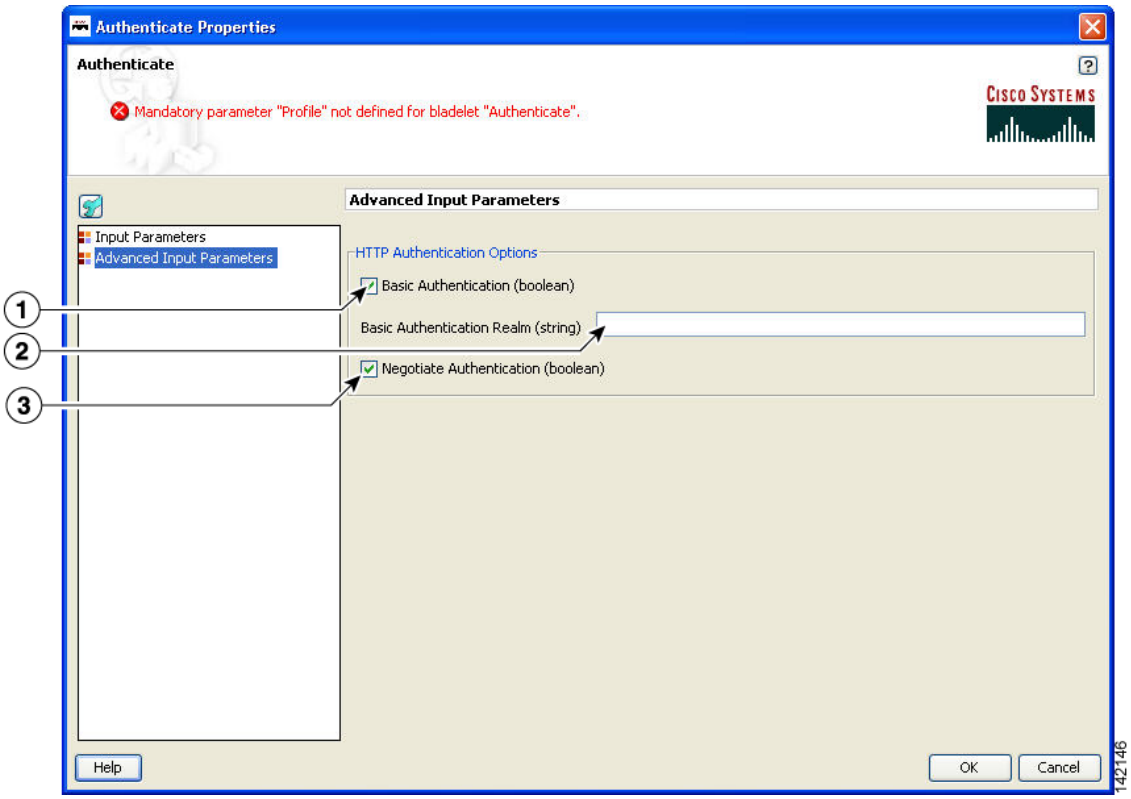
An Authenticate Bladelet authenticates only one type of identity. To authenticate multiple types of identity, you must use multiple instances of Authenticate Bladelet in the PEP.

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.

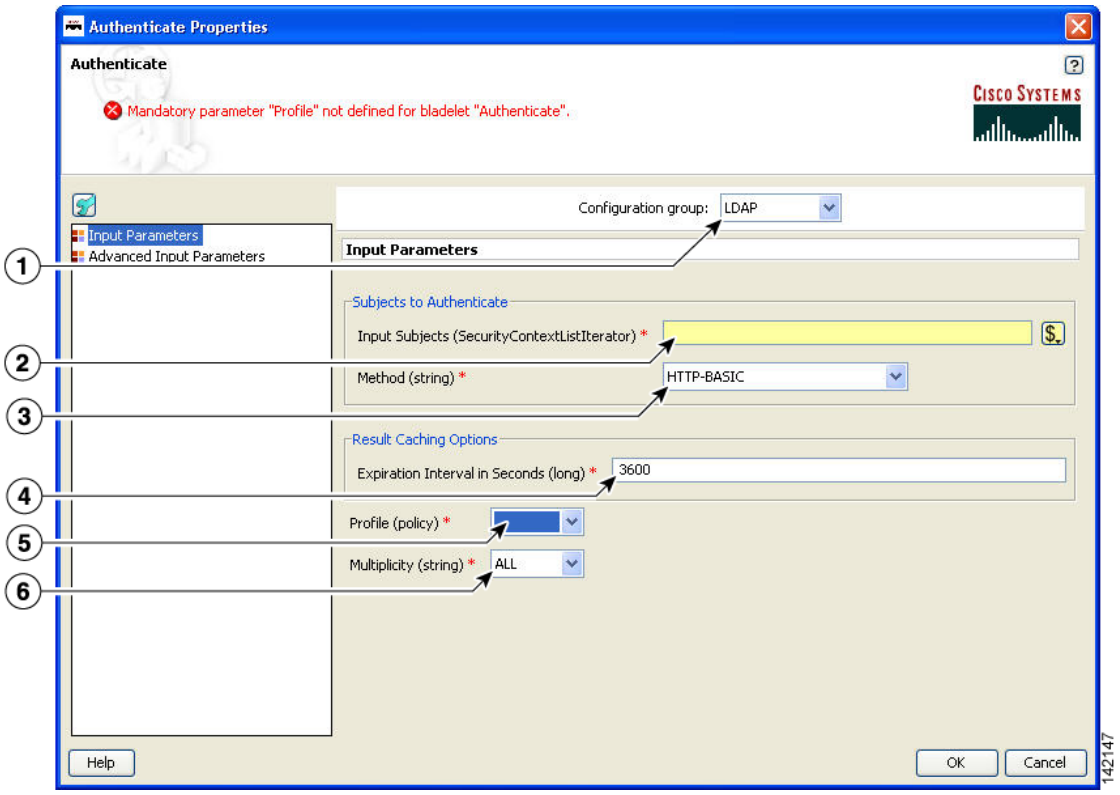
Note Each required field in the Bladelet Properties window is marked by a red asterisk. Until all required fields are completed with the correct value, an error message appears on top of the Bladelet Properties window to indicate which field remains to be completed or indicates that there is a parameter type mismatch and so on before the Bladelet is completely configured.

Figure 3-117 Authenticate Properties Window—Advanced Input Parameters



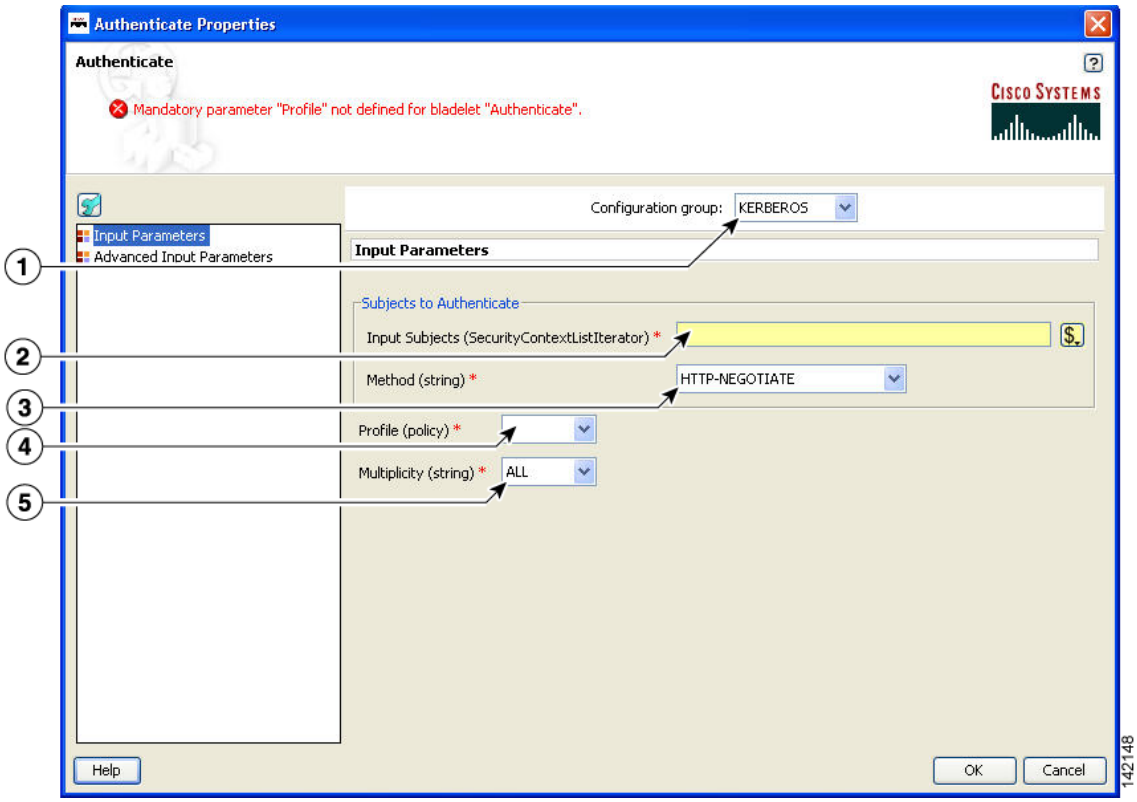
1	Basic Authentication	Whether or not this Bladelet supports HTTP basic authentication.
2	Basic Authentication Realm	Customized basic authentication realm. If nothing is defined, AON node hostname is used as default realm name.
3	Negotiate Authentication	Whether or not this Bladelet supports HTTP negotiate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

Figure 3-118 Authenticate Properties Window—Input Parameters, LDAP



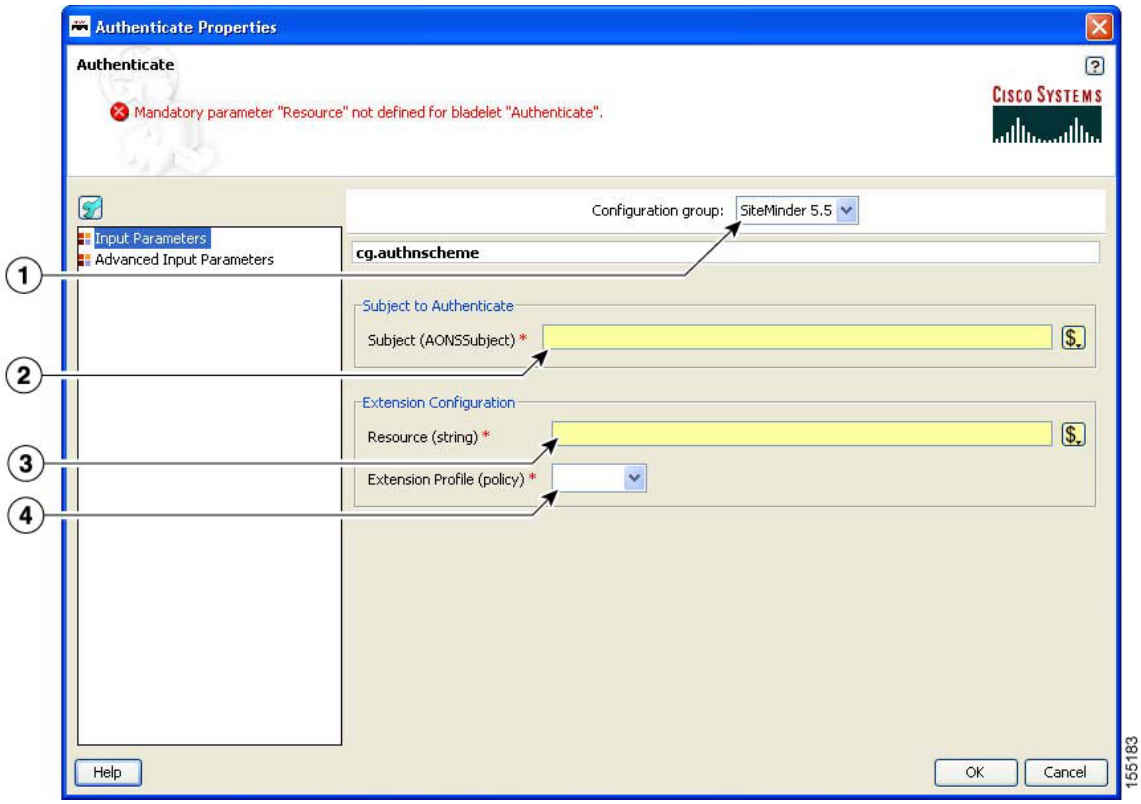
1	Configuration Group	Configuration group, set here to LDAP.
2	Input Subjects	Data structure that stores the identity information to be authenticated. It should be exported by an Identity Bladelet.
3	Method	Type of the identity to be authenticated in this Bladelet.
4	Expiration Interval in Seconds	Time-to-live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

Figure 3-119 Authenticate Properties Window—Input Parameters, Kerberos



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

Figure 3-120 Authenticate Properties Window—Input Parameters, SiteMinder 5.5



1	Configuration Group	Configuration group, set here to SiteMinder 5.5.
2	Subject	AONSSubject. It specifies the subject being authenticated.
3	Resource	String. Resource the subject is trying to access.
4	Extension Profile	Policy. Property set that specifies parameters used to connect to a Policy Server.

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

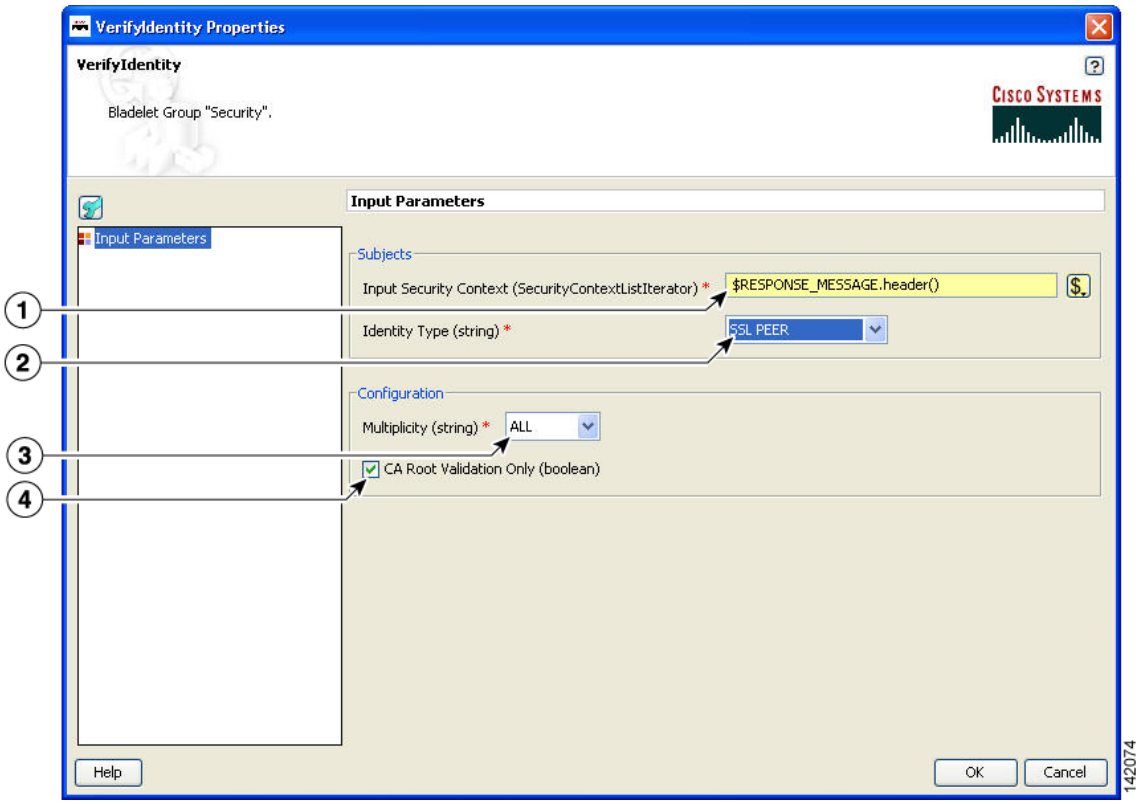
- The AONSSubjects to be verified are generated by Identify Bladelet. Ensure that Identify Bladelet precedes Verify Identity Bladelet in a valid PEP. Use the export parameter of Identify Bladelet to retrieve the AONSSubjects.
- Populate “trustedCACerts:” with trusted CA certificates. If the certificate 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 3-121 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.

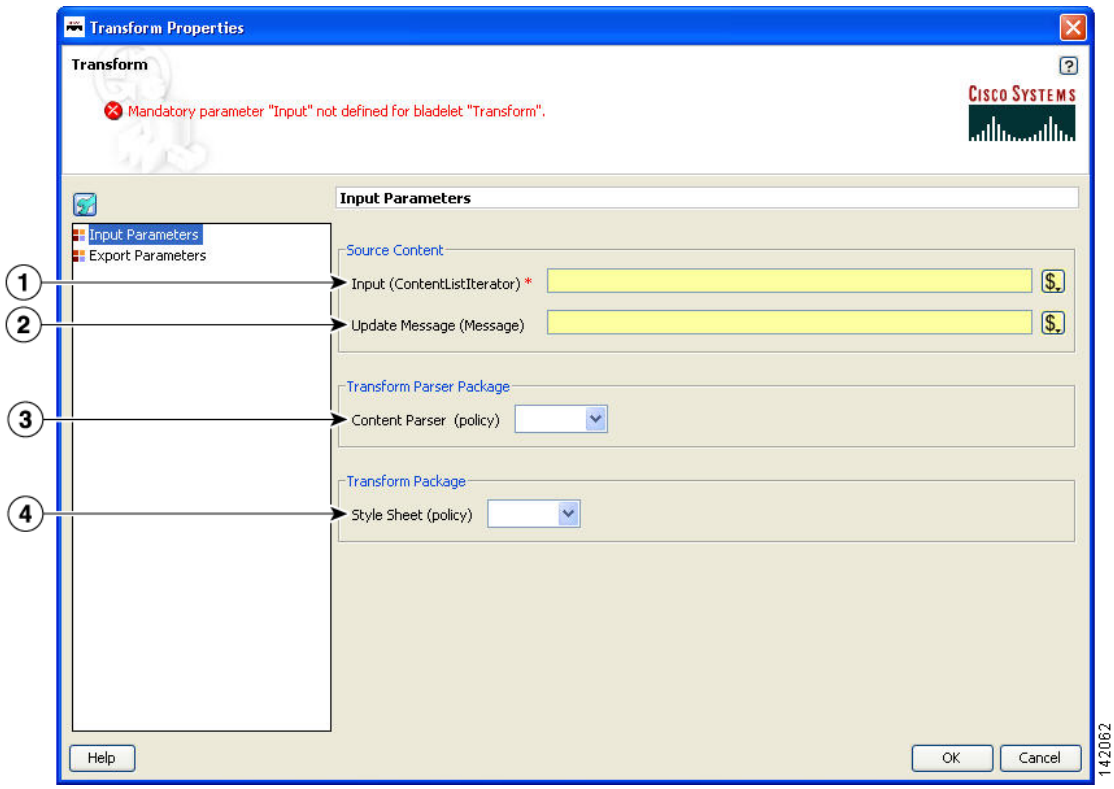
If the message is a multipart message and the list of contents are transformed in to a list of result contents, you must use BuildCompositeMessage to build a result multipart message.

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.

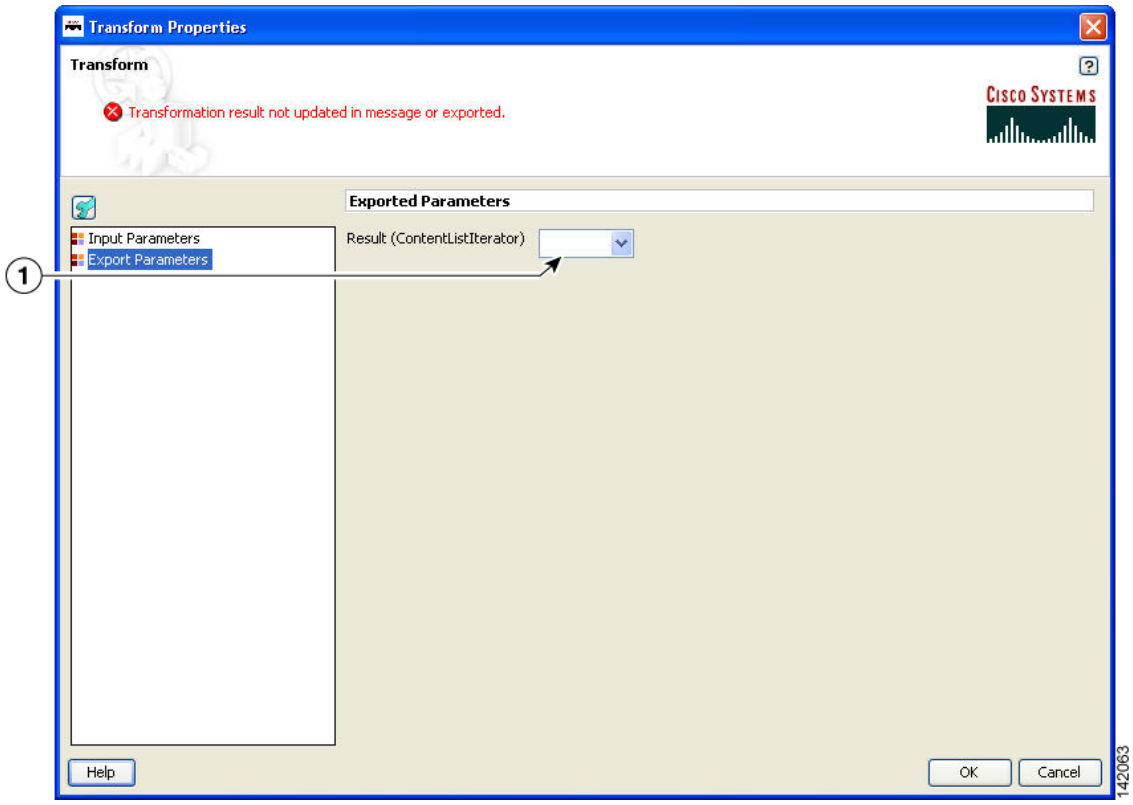
Details

Figure 3-122 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 3-123 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.