CHAPTER 5

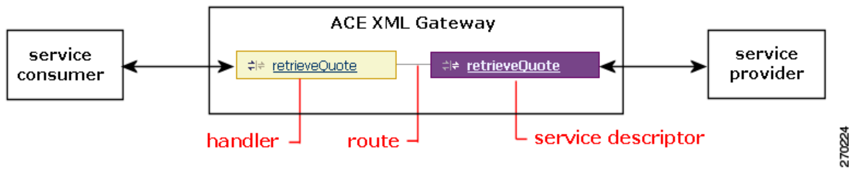
Working with Handlers, Routes, and Service Descriptors

This section describes how to work with advanced virtual services. It covers these topics:

About Advanced Virtual Services

As described in Chapter 4, "Working with Virtual Services," a basic virtual service is made up of a single policy object. It contains settings for both the consumer interface of a service at the ACE XML Gateway and for the ultimate service provider. An advanced virtual service, on the other hand, is made up of objects that define the service consumer and provider interfaces separately. The interfaces are represented by the handler and service descriptor policy objects.

Figure 5-1 Advanced virtual service



Advanced virtual services are used to implement several advanced message handling scenarios. For one, they allow you to create branched message routes (in which the path for a given message is chosen from several possible routes). They also allow for protocol mediation scenarios, in which incoming messages of one protocol are converted to another. Like any virtual service, an advanced virtual service contains message validation specifications and processing instructions for a given backend service.

This chapter discusses how to create and configure advanced virtual services.

Converting Basic Virtual Services to Advanced

A basic virtual service object can be converted into an advanced virtual service at any time. Therefore, when first creating a virtual service, it usually makes sense to start with a basic virtual service object, since you can always convert a basic virtual service object to an advanced virtual service later. In contrast, a handler, service descriptor, and route cannot be combined into a basic virtual service object. Furthermore, basic virtual service objects are usually easier to work with early in development. For instance, a basic virtual service can be duplicated or deleted with a single mouse click, while an advanced virtual service requires multiple component objects to be duplicated or deleted to achieve the same result.

When you convert a basic virtual service into an advanced virtual service, the settings of the original virtual service are propagated to the generated objects, so that they are functionally identical to the original virtual service. After the conversion, you can specify different settings for the interfaces, configure branched routing, and so on.

You cannot convert an advanced virtual service back to a basic virtual service after it has been converted. The only way to restore a virtual service to a basic virtual service is to recreate it.

To convert a basic virtual service into an advanced virtual service:

Step 1 As an Administrator user or a Privileged user with the Routing role in the console, set the active subpolicy to the one that contains the virtual service you want to convert.

Step 2 In the **Virtual Services** browser, click the name of the virtual service that you want to convert to an advanced virtual service.

The settings page for the virtual service appears.

Step 3 Click the **Convert to Advanced** button at the bottom of the page. (You may need to scroll down to make the button visible.)

Step 4 Confirm the operation when prompted.

The settings page for the new handler appears. You can now modify the settings for the handler, or return to the **Virtual Services** browser to access other components generated by the conversion.

Creating Advanced Virtual Services by WSDL Import

Advanced virtual services can be created manually (in which the handler, service descriptor, and route are created and associated with each other individually) or automatically (in which the objects are generated and configured by WSDL import). This section describes how to create an advanced virtual service automatically, by WSDL import.

The steps for automatically generating an advanced virtual service are similar to that of generating a basic virtual service (see "Virtualizing Services by WSDL Import" section on page 4-23), with the differences highlighted in the following steps:

Step 1 In the **Configure Interface & Access** page of the WSDL import wizard, expand the Advanced Options item at the bottom of the page.

Step 2 If it is selected, remove the selection from the option labelled **Condense SOAP Document operations into one virtual service per WSDL "Service" element, if possible**.

The next option, **Create a direct-mapping route between each handler and service instead of a pass-through route**, becomes available.

Step 3 Enable the option **Create a separate handler and service descriptor for each operation**.

Step 4 Choose whether the route should operate in direct mapping or passthrough fashion with the **Create a direct-mapping route between each handler and service instead of a pass-through route** option. This option determines how messages are handled at the Gateway as follows:

- With direct mapping, an incoming message is disassembled at the ACE XML Gateway and recreated according to the policy settings and usual normalization measures. Headers or other message elements that are not standard for the message protocol or specifically provided for in the policy configuration are removed by the Gateway. This option provides greatest security and imparts the benefits of message normalization, but can lead to unanticipated integration problems.
- If direct mapping is disabled, as it is by default, the more permissive passthrough mapping mode is used. Instead of completely disassembling and recreating the message, the ACE XML Gateway propagates unexpected headers and other message attributes to the outgoing interface.

Step 5 Configure other settings in the configuration wizard as described in "Virtualizing Services by WSDL Import" section on page 4-23.

Creating Service Descriptors

If a WSDL that describes the services you want to virtualize at the Gateway is not available, you can create an advanced virtual service by hand. This section describes how to create service descriptors, the component of a virtual service that defines settings for the backend service provider interface. It contains specifications for the outgoing request sent to the service provider and for the incoming response.

To create a service descriptor:

Step 1 While logged in to the ACE XML Manager web console as an Administrator user or a Privileged user with the Routing role, open the Virtual Services page by clicking the **Virtual Services** link in the navigation menu.

Step 2 From the Virtual Services menu, choose **Create a Service Descriptor** and click **Go**. (The menu appears at the top of the page in the Virtual Services browser.)

The **Step 1 of 5: Service Protocol** page appears.

Step 3 Choose the protocol of the backend service from the **Select the protocol that this service uses** menu. Messages delivered through this service descriptor will use the protocol you choose. Notice that there are options for SOAP, HTTP, ebXML messaging, TIB/RV, JMS, and MQSeries.

Note Additional protocols may appear if custom I/O extensions created with the ACE XML Gateway SDK are installed on your system. For more information about I/O extensions, contact Cisco support.

Step 4 Click the **Continue** button.

Step 5 In the **General Information** page, type a name for this service descriptor that is unique for service descriptors in the policy.

Step 6 Choose the server that hosts the backend service in the **Server** menu and click **Continue**.

If the **Server** menu does not appear or the correct server is not listed in the menu, a resource that represents the server does not yet exist in the policy. To add the server definition to the policy, click the **Add a New Server** button.

For more information, see Chapter 14, "Configuring Destination Server Settings."

SOAP WS-Addressing service descriptors do not use a preconfigured server. For more information on these types of service descriptors, see "Dynamic Service Routing with WS-Addressing" section on page 11-121.

Step 7 The settings in the next three steps vary depending on the protocol of the service descriptor. For details on the steps by protocol, refer to the documentation referenced in the following table.

Table 5-1 Protocol documentation reference

For details on...	See...
SOAP RPC or SOAP Document	"Creating a Basic Virtual Service Manually" section on page 4-30
SOAP with WS-Addressing	"Dynamic Service Routing with WS-Addressing" section on page 11-121
HTTP	"Creating a Basic Virtual Service Manually" section on page 4-30
ebXML Message Service	Chapter 18, "Working with ebXML Traffic"
TIB/RV	Chapter 17, "Working with TIB/RV and MQ Services"
MQSeries	Chapter 17, "Working with TIB/RV and MQ Services"

Step 8 After completing the configuration, click **Save Changes** to commit your changes to the active subpolicy of the working policy.

The ACE XML Manager displays the settings page for the new service descriptor. You can now create handlers for routing consumer messages to the service represented by the new service descriptor.

Creating Handlers

A handler policy object defines settings for the consumer interface for an advanced virtual service. Among other settings, a handler specifies the path for invoking the service at the ACE XML Gateway, the listening port on which it is exposed, and message specifications for the incoming request and outgoing response.

To create a handler:

Step 1 While logged into the ACE XML Manager web console as an Administrator user or a Privileged user with the Routing role, set the active subpolicy to the one that is to provide the new handler, if it is not already active.

Step 2 Click the Quick Link icon next to the **Policy** heading of the navigation menu and choose **Create a new handler**.

Step 3 In the New Handler page, choose the protocol of the messages this handler should accept from the **Select the protocol for this handler** menu and click **Continue**.

Step 4 In the Step 2 of 5: General Information page, configure these settings:

- Name—A descriptive name for this handler that is unique for handlers in the policy.
- Handler Group—The handler group to which this handler belongs. Handler groups are used to organize related handlers.
- Default Message Logging—The message-logging level for activities of this handler under normal (non-error) conditions. During initial development, it may be helpful to log message bodies.
- On Error—The message-logging level this handler uses under error conditions. For more information about logging levels, see Chapter 32, "Monitoring System Status."

  If message body logging is enabled, these options appear for processing the message before it is logged:

  – Request XSLT—An XSL transformation this handler applies to the message body of an incoming request message before logging it. To transform the message, choose an XSLT from the menu or upload it.

  – Response XSLT—The XSL transformation this handler applies to the message body of a response from the protected service before logging the message body. The default value is none. To transform the message, choose an XSL transform from the menu or upload it.

- Send SNMP Traps—Configures this SNMP trapping behavior for this handler. The default value, never, causes the handler not to send SNMP traps. To use SNMP traps to alert administrators of the status and events relating to the handler, you also need to enable SNMP traps from each Gateway appliance configuration menu, as described in the *Cisco ACE XML Gateway Administration Guide*.

  You can have a trap generated for errors only or for any event. The SNMP agent sends a trap named `softwareNotificationTrap` when a handler event occurs.

- Flex Path—Directs message processing for this service to the Flex Path, bypassing the stream-based Reactor processor. In most cases, you can allow the ACE XML Manager itself to assign services to Flex Path. Usually, if a task applied in a virtual service is not supported by the Reactor (such as protocol mediation), the service is assigned to Flex Path processing automatically, without any special configuration requirements. However, an exception exists for the GZIP response compression by port feature. If message compression is enabled on a port, the ACE XML Gateway compresses qualifying responses only for services that for any reason are already handled on the Flex Path. Reactor does not perform compression, and messages it handles that would otherwise be compressed are not passed to the Flex Path. Therefore, if it is important in your system to have responses for a service compressed when possible, you should enable this option on the service handler or virtual service object.
- Description—A text message that describes this handler to other users of the ACE XML Manager. It appears in the General section of the information page the ACE XML Manager displays when you click this handler's name in the **Virtual Services** browser.

  This description is also visible in the service directory on the ACE XML Manager. This directory is intended to advertise service availability to service consumer developers. Therefore, you should add a description that documents the service for those users.

Step 5 When finished, click **Continue**.

Step 6 The settings in the pages that follow vary depending on the protocol you selected. For protocol-specific information on handlers, see the section of this guide indicated in the following table.

Table 5-2 Protocol documentation reference

For details on...	See...
SOAP	"Creating a Basic Virtual Service Manually" section on page 4-30
HTTP	"Creating a Basic Virtual Service Manually" section on page 4-30
ebXML Message Service	"Working with ebXML Traffic" section on page 18-185
TIB/RV	"Working with TIB/RV and MQ Services" section on page 17-179
MQSeries	"Working with TIB/RV and MQ Services" section on page 17-179

Step 7 After completing the handler configuration, click **Save Changes** to commit the changes to the active subpolicy.

Step 8 You are prompted to add a route for the handler. A route associates the handler with a particular backend service (through a service descriptor). You can finish without adding a route now or add a route as described in "Working with Routes" section on page 5-44.

When finished, the settings page for the handler appears. By default, the handler is not yet provisioned to accept requests from service consumers. For information on provisioning the handler, see Chapter 6, "Controlling Access to Services."

Working with Routes

A route connects a handler and service descriptor. It defines a path that messages can take between a consumer interface at the Gateway and a backend service. A route can itself impose requirements or processing measures upon messages.

Routes are usually created in the course of creating handler and service descriptors. However, occasionally you may need to create a route directly. This is most often the case when configuring a handler or service descriptor to have multiple routes.

Multiple routes allow for branched, decision-based routing. When multiple routes exist for a message path, the Gateway uses *selectors* to determine which route should get a particular message. Selectors can specify conditions based on header values or other attributes of the message. If a message meets all selector conditions of a route, it is passed on the route.

Non-matching messages sent to the handler get rejected with an HTTP 404 error, unless you designate a default route. Messages that fail to match the selectors for any of the handler's routes get assigned to the default route.

While a handler can have multiple routes, a given message can only be accepted on one route. If a message to a handler matches the conditions in more than one non-default route, only one is picked (that is, the message is not broadcast to multiple service descriptors). If one matched route has more selectors than the other, it wins. If the matching routes are equally restrictive, however, one of the routes is picked by the ACE XML Gateway. How this ambiguity is resolved cannot be guaranteed by the ACE XML Gateway. In general, you should avoid creating multiple routes with possibly ambiguous selector conditions.

A route can connect a handler to a service descriptor of different protocols. Message mediation is performed transparently at the route. However, you can also directly specify transformations at the route to modify the content or structure of messages.

Configuring a Route

To add a route to a handler:

Step 1 While logged into the web console as an Administrator or Privileged user with the Routing role, set the active subpolicy to the one in which you want to configure a route, if it is not already active.

Step 2 Click the **Virtual Services** link in the navigation menu.

Step 3 In the **Virtual Services** browser, click on the name of the handler for which you want to create a route.

Step 4 Scroll to the bottom of the service settings page, if necessary, until the Routes section of the page is visible, and click the **Add New Route** button.

Step 5 Choose the destination service descriptor for this route from the menu labeled **Please select the service to which you wish to route requests** menu.

Note Only service descriptors available to the currently active subpolicy appear in this menu.

Step 6 Choose whether the route should be the default route or specify message conditions using selectors. With multiple routes, a default route provides a path for messages that do not otherwise meet the conditions of other routes. (While using a default route is optional, there cannot be more than one default route.) To add selectors to this route, follow the instructions in "Adding Selectors" section on page 5-45.

Step 7 Click **Save Changes** to create the new route.

The new route is created and attached to the handler. Optionally, you can now configure custom exceptions and additional request/response processing for this route.

Adding Selectors

To implement branched routing from a particular handler, you use selectors to set conditions for the messages that pass on the routes. A selector can impose conditions based on message content, header values, or other attributes.

To add a selector to an existing route:

Step 1 In the Routes section of the handler settings, click the **Edit** link next to the Selectors header for the route to which you want to add a selector.

The **Edit Route to <*ServiceDescriptor*>** page appears.

Step 2 To configure this route to test incoming messages against one or more selectors, enable the **Only requests matching these selectors** button.

The **Add a New Selector Row** button becomes enabled.

Step 3 Click the **Add a New Selector Row** button.

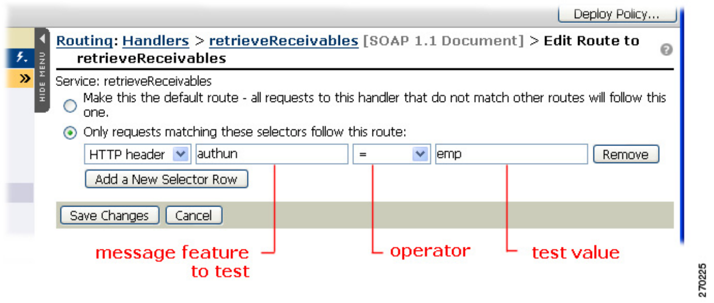
Controls for configuring a selector appear on the page in the form of three fields. The menu on the left indicates the message feature to which the condition should apply. Note that the features vary according to the protocol of the handler.

For example, for an HTTP POST Arguments handler, you can configure criteria against a given header or argument value (if you have configured arguments in the message specification).

Step 4 Choose the message feature the selector is to match from the menu on the left side of the page.

When you choose a message feature to be tested, additional controls appear for specifying the criteria for that type of feature. Figure 5-2 shows the form for a message argument.

Figure 5-2 Selector configuration settings



Routing: Handlers > retrieveReceivables [SOAP 1.1 Document] > Edit Route to retrieveReceivables

Service: retrieveReceivables

( ) Make this the default route - all requests to this handler that do not match other routes will follow this one.

(●) Only requests matching these selectors follow this route:

HTTP header | authun | = | emp

Step 5 Use the controls to specify the condition a message must meet to be passed on the route.

For example, to screen by a custom HTTP header, choose HTTP header as the condition attribute and indicate the name of the header. In the test value, specify a value that, if matched in the incoming message, causes the message to be accepted.

Step 6 Add as many conditions as desired. A message must meet all of the selector conditions you specify to be passed on the route.

Step 7 Click **Save Changes** to commit the changes to the working policy.

The new selector information appears in the **Selectors** pane of the **Routes** section of the handler information page.

Designating the Default Route

For handlers that have more than one route, you can specify a default route. The default route is used for messages that do not meet the conditions of any other route.

Note Notice that having a default route is not a requirement. If a message does not match conditions of any of the routes of a handler, an HTTP 404 error response is returned to the client.

By default, new routes do not contain selector conditions; they are created as default routes. After configuring a route condition, you may need to restore a route as the default route. To do so, follow the instructions in "Adding Selectors" section on page 5-45 to access the selector settings. In the Edit Route page for the selector, click the option labelled **Make this the default route**.

You cannot deploy a policy that contains a handler with more than one default route. If it does, this error appears in the console: "ERROR: This Handler defines multiple default routes." You need to remove or add a condition to the additional default routes to correct this error.

Modifying Message Content at the Route

A route can modify message content in several ways. For instance, you configure an XSLT on the route that is applied to any message passed by the route. The capability for modifying message at the route is similar to that offered by the handler or service descriptor. However, specifying content processing at the route enables you to specialize this behavior for a specific route among possibly others on the handler, without having to assign the behavior to the service descriptor. The best place to apply the processing (whether in the route, service descriptor or handler) depends on your specific use case.

By default, a route passes message body and argument content between handlers and service descriptors in passthrough mode. HTTP headers in incoming messages are removed by the ACE XML Gateway, with new ones created for the outgoing message.

A route can treat message body and argument content in *passthrough* fashion or as a *direct mapping*.

- In direct mapping, the incoming message is disassembled at the Gateway and recreated according to the service settings in the policy and the Gateway's usual normalization measures. Accordingly, if not specifically provided for in the policy configuration, headers or other elements are removed by the Gateway. This option provides greatest security and imparts the benefits of message normalization, but can lead to unanticipated integration problems.
- Passthrough mapping, the default, is more permissive. Instead of completely disassembling and recreating the message, the Gateway propagates unexpected headers and other message attributes to the outgoing interface.

Alternatively, you can configure cross-mapping of message arguments. In this case, the values of one argument are used to populate another argument.

Creating Content Mappings

To map fields from the input message to the output message in a route, you first need to configure the arguments in the handler and corresponding service descriptor specifications. You can then configure the mapping, as described in the following steps.

These procedures apply to argument mapping in the response leg of the message transaction, however, the steps are similar for other contexts as well.

Step 1 In the **Virtual Services** browser, click the name of the service descriptor for which you would like to configure argument mapping.

Step 2 In the **Incoming Response** settings, click the **edit** link next to **Response Message Specification**.

Note Argument configuration settings are unavailable in contexts in which arguments are not relevant. For example, it is not available in HTTP response processing. Generally, argument processing in response message is most likely to be relevant for messaging-based sources, such as JMS.

Step 3 In the **Request Message Specification** page, click **Add A New Row**.

Step 4 Configure the following settings for the argument and click **Save Changes** when finished.

- **Name**—The name of the argument.
- **Req**—Whether the field is required.
- **Type**—The type of the argument.
- **Validation**—Whether the ACE XML Gateway should check the argument for correctness. The type and content are subject to validation. If you choose type & content in the console, fields appear that let you configure the manner of validation appropriate for the type you've selected (for example, an XML schema selector for XML arguments).

In a *service descriptor* response specification, these settings refer to the source arguments, that is, the mapping source arguments of the incoming response. In a *handler* response specification, these settings control how arguments are created in the outgoing response.

Step 5 Configure the mapping target argument in the handler connected to the service descriptor. Keep in mind that this is the argument that the client expects to find in response messages that will be populated with content from the argument you configured in the service descriptor.

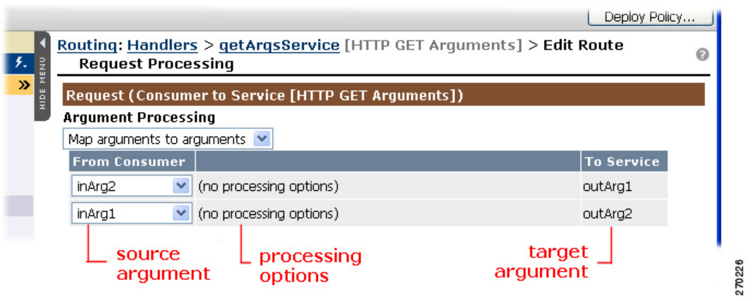
Step 6 When finished, in the **Routes** settings of the handler information page, click the **edit** link next to either of the following headings:

- **Req. Processing**—To map fields from the incoming request to the outgoing request
- **Resp. Processing**—To map fields from the incoming response to the outgoing response

Step 7 From the **Argument Processing** menu, choose an option that specifies argument mapping, such as **Map body and properties to body and properties**.

Step 8 In the **From Service**, choose the source argument from the menu that corresponds to the **To Consumer** argument (or **From Consumer** to **To Service**, if mapping a request). Alternatively, choose no value or a constant value to be used to populate the argument.

Figure 5-3 Argument mapping


Routing: Handlers > getArgsService [HTTP GET Arguments] > Edit Route
Request Processing

Request (Consumer to Service [HTTP GET Arguments])

Argument Processing: Map arguments to arguments

From Consumer		To Service
inArg2	(no processing options)	outArg1
inArg1	(no processing options)	outArg2

Step 9 After mapping the arguments, click **Save Changes** to commit your changes to the working subpolicy.

For more information on argument mapping, see "Working with Request Arguments" section on page 5-50. For more information on message mediation, see "Configuring JMS Mediation" section on page 16-175.

Processing HTTP Headers

By default, HTTP headers in incoming messages are not propagated to outgoing messages. The ACE XML Gateway consumes the headers of incoming messages and creates new ones as specified in the policy for the outgoing message.

You can change this behavior so that headers are passed through, assigned a constant value, or modified by regular expression match and replacement. To do so, define a HTTP header processing specification at the route. The specification needs to identify the incoming header by name.

As an alternative to specifying incoming headers by name, you can use a wildcard operator (*) to match any header. This enables you to pass through all headers received in the message by the ACE XML Gateway. In particular, you may need to specify this configuration when backend applications or network elements rely upon headers such as `Set-Cookie` or `User-Agent` headers in requests.

The values of certain types of headers cannot be set in the policy. In general, these headers are populated with runtime values appropriate for the message, such as the `Date` and `Content-Length` header. In addition, header passthrough does not apply to what are considered "hop-by-hop" headers. Such headers are significant only in the context of a single transport-level connection, and include:

- Accept-Encoding
- Connection
- Keep-Alive
- Proxy-Authenticate
- Proxy-Authorization
- TE
- Trailers
- Transfer-Encoding
- Upgrade

Note The `Server` HTTP header value set by the ACE XML Gateway is configurable. To access the setting, click the **I/O process settings** link in the System Management page. The value can be set separately for the HTTP Server and Reactor HTTP processor in the **Server Name** fields.

For other types of HTTP headers, including custom headers, you can configure HTTP Header processing as follows. Note that the steps vary between advanced and basic virtual services.

For an advanced virtual service, follow these steps:

Step 1 In the setting page of an HTTP based handler, click **edit** next to either:

- **Req. Processing** to configure the headers of outgoing requests
- **Resp. Processing** to configure the headers of outgoing responses

Step 2 Under the **HTTP Header Processing** section of the page, click the **Add a New HTTP Header** button. Note that his option only appears in HTTP based routes, such as SOAP or HTTP GET.

Step 3 In the **Header Name** field, type the exact name of the header as it would appear in the message.

You can have the ACE XML Gateway pass through all headers received in the incoming message by entering the wildcard operator (*) in this field. Note that wildcard matching should only be used with the passthrough header option, not with options that perform content replacement. Also, partial wildcard matching is not supported, such as `custom-*-header`.

Step 4 For the **Action**, choose from either:

- **pass through**, to have the header and its value propagated from the incoming to the outgoing interface without change (as mentioned, this does not apply to the `Server` or `Date` headers).
- **send constant value**, to have a header's value rewritten to the value you specify. If not present in the incoming request, the header is added.
- **use regular expression match**, to use a variable part of the incoming header value along with text you specify in the output pattern. Use backslash variables (such as \1 and \2) to propagate matched values of the incoming header value to the outgoing header.

Step 5 Click **Save Changes** when finished.

For basic virtual service objects, you can only configure passthrough of supported headers. (For features such as header rewriting, you must convert the object to an advanced virtual service.)

To enable HTTP header passthrough for basic virtual service objects:

Step 1 In either the request or response processing sections of the service settings, click the **enable** link next to the HTTP Header Processing label.

To passthrough headers from the incoming request to the backend service, click the **enable** link in the Request Processing section. To passthrough headers returned from the backend service to the client, click the **enable** link in the Response Processing section.

Step 2 Enable the option to pass through all incoming HTTP headers.

Deploy the policy to have your changes take effect at the ACE XML Gateway.

Working with Request Arguments

In general, you can configure routing between handlers and service descriptors of different protocols without concern for the details of how the message is transformed—the ACE XML Gateway automatically converts messages as appropriate for the mediation. This default transformation may need to be modified to achieve particular results in some cases, but often the default results are sufficient. However, an exception exists for mediation of messages that contain URL request arguments. In this case, you will need to perform some specific configuration steps to ensure a useful transformation.

For HTTP Get Argument handlers, request arguments appear in the request URL after the question mark ("?") delimiter with multiple arguments separated by an ampersand ("&"). For example:

http://sample.com/quote/agent?sid=2345&loc=ca

For HTTP Post Arguments handlers, the arguments are conveyed in the body of the request, but take the same name-value form as HTTP Get arguments.

You can have the ACE XML Gateway process the argument data in several ways, including by:

- Argument passthrough, in which URL arguments are propagated to the request URL without modification. Argument passthrough occurs automatically if both the consumer interface and backend interface are HTTP Get Arguments protocol. For HTTP Post handlers, you'll need to configure this behavior directly.

- Argument mapping, in which the content of a specific argument is propagated to an element (either another argument or body) of the outgoing request.
- Argument conversion, in which the ACE XML Gateway renders arguments of the incoming request as an XML document. You can apply an XSLT to the document to further process the content. This option lets you process multiple, arbitrary arguments in the request.

The options available vary depending on the protocol of the handler and service descriptor you are using, as described in the following sections.

For information on argument mapping, see “Modifying Message Content at the Route” section on page 5-47. The other options are described next.

Using URL Argument Passthrough

When routed to HTTP Post Argument service descriptors, arguments received in incoming requests are not automatically propagated to the URL of the outgoing request as they are for outgoing HTTP Get Argument requests.

By default, arguments received in the incoming request, whether to a HTTP Get Argument or HTTP Post Argument handler, are mapped to arguments in the body of the outgoing HTTP Post Argument request. (Arguments addressed to HTTP Get Argument handlers that route to HTTP Get Argument service descriptors are automatically passed through to the outgoing request URL.)

You can have URL arguments of the incoming request propagated to the outgoing request URL instead of the body using the following steps. Use this configuration when the backend service expects both an HTTP Post body and URL arguments in the request.

First, set up URL argument passthrough as follows:

Step 1 Open the settings page for the handler for which you want to configure argument passthrough.

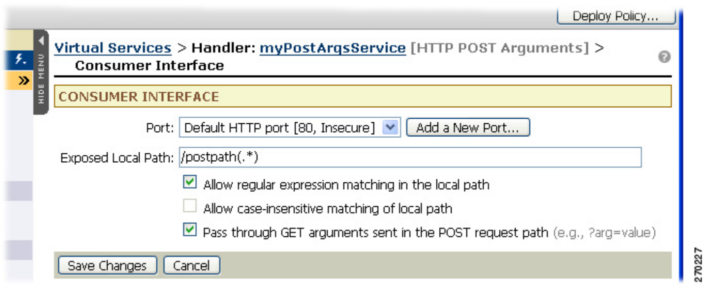
Step 2 Click the **Edit** link next to the **Consumer Interface** heading.

Step 3 If it is not already selected, click the checkbox labelled **Allow regular expression matching in the local path**. This feature must be enabled for URL argument passthrough.

Step 4 In the **Exposed Local Path** field, add a regular expression that will match the query parameters in an incoming request. To match any number of arbitrary queries, add a wildcard expression, such as: (.*)

Step 5 Check the option labeled **Pass through GET arguments sent in the POST request path**. The page should appear similar to the following:

Figure 5-4 Consumer interface argument passthrough



Virtual Services > Handler: myPostArgsService [HTTP POST Arguments] > Consumer Interface

CONSUMER INTERFACE

Port: Default HTTP port [80, Insecure]

Exposed Local Path: /postpath(.*)

[✓] Allow regular expression matching in the local path
[ ] Allow case-insensitive matching of local path
[✓] Pass through GET arguments sent in the POST request path (e.g., ?arg=value)

Step 6 Click **Save Changes** and configure service descriptors, as described next.

For each service descriptor to which this handler routes message:

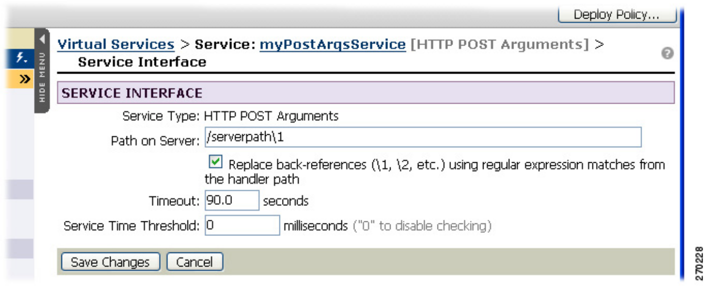
Step 1 Open the service descriptor settings by clicking on the name of the service descriptor in the **Virtual Services** browser.

Step 2 Click the **Edit** link next to the **Service Interface** heading.

Step 3 In the **Path on Server** field, add a back reference to the existing path. If you have a single regular expression in the consumer path field, you only need to add a single back-reference: \1

Step 4 Check the checkbox labeled **Replace back-references (\1, \2, etc.) using regular expression matches from the handler path** and save your changes.

Figure 5-5 Service interface argument passthrough



Service Type: HTTP POST Arguments
Path on Server: /serverpath\1
☑ Replace back-references (\1, \2, etc.) using regular expression matches from the handler path
Timeout: 90.0 seconds
Service Time Threshold: 0 milliseconds ("0" to disable checking)

The configuration is now complete. After you deploy the policy, arguments in the requests addressed to the handler are passed through to the outgoing request URL.

Processing Arguments

The steps for creating a simple mapping of incoming message arguments to outgoing messages are described in "Modifying Message Content at the Route" section on page 5-47. For the argument mappings described in that section, the argument name must be configured in the policy. Arguments not specified in the policy are dropped.

Some applications require greater flexibility. In particular, dynamic argument mappings are required for an increasingly common use case—mediating from traditional or REST-based web applications on the client side to SOAP services on the backend.

The data passed in URL arguments in client requests is usually important to the backend application and needs to be propagated to the outgoing request.

When you choose the **Convert incoming arguments to XML in outgoing request body** option in the request processing configuration for the route, the ACE XML Gateway generates an XML document from the request arguments. With this option, you can further specify an XSLT that processes the argument source document from the ACE XML Gateway's default format to the format expected by the destination service.

What you need to know to process the XML argument document differ slightly depending on whether you are mapping to a SOAP RPC or SOAP document service, as described in the following sections.

Processing Request Arguments for SOAP Document Services

In converting arguments from an HTTP Get or a Post Body request to a SOAP Document service descriptor, the ACE XML Gateway creates a source document in which each request argument is mapped to a `param` element:

```
<param name="arg_name" value="arg_value" />
```

In the element, *`arg_name`* is the name of the argument passed in the request and *`arg_value`* is its value.

There can be multiple arguments with the same name in the document. The arguments appear in the order in which they appear in the request. Also, the `param` elements are contained in a single `params` parent element.

For example, for the following request URL, the ACE XML Gateway would generate the intermediary SOAP message shown in Example 5-1:

http://example.com/quote?custid=9239&loc=ca&loc=us

Example 5-1 Sample Argument Source XML Document

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

 <soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     <params>
       <param name="custid" value="9239"/>
       <param name="loc" value="ca"/>
       <param name="loc" value="us"/>
     </params>
   </soap:Body>
 </soap:Envelope
```

A typical XSLT configured for a route would, therefore, match `param` elements and process the arguments as desired.

The following listing shows an example of an XSLT capable of processing this message. It preserves the `Envelope` and `Body` elements. Within a `purchaseOrder` element, it creates an element for each `param`. The element is named with the `name` attribute and contains the `value` attribute content.

Example 5-2 Sample Argument Processing XSLT

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >

 <xsl:output method="xml"/>
 <xsl:template match="*">
   <xsl:element name="{name(.)}" namespace="{namespace-uri(.)}">
     <xsl:for-each select="@*">
       <xsl:attribute name="{name(.)}"
              namespace="{namespace-uri(.)}">
         <xsl:value-of select="."/>
       </xsl:attribute>
     </xsl:for-each>
    <xsl:apply-templates/>
  </xsl:element>
 </xsl:template>

 <xsl:template match="params">
   <xsl:element name="purchaseOrder">
     <xsl:for-each select="param">
       <xsl:element name="{@name}">
         <xsl:value-of select="@value"/>
       </xsl:element>
     </xsl:for-each>
   </xsl:element>
 </xsl:template>

</xsl:stylesheet>
```

To implement this type of argument processing, follow these steps:

Step 1 Open the Req Processing settings for the HTTP Get Arguments or HTTP Post Arguments handler that routes to the target SOAP Document service descriptor.

Note For more information, see "Modifying Message Content at the Route" section on page 5-47.

Step 2 In the argument processing option, choose **Convert incoming arguments to XML in outgoing request body**. This options directs the ACE XML Gateway to render arguments from the request into an intermediary XML document, to which you can apply an XSLT for final processing.

Step 3 Specify the XSLT that you want applied to the XML document in the **Transform with XSLT** menu. If the XSLT is not yet loaded in the policy, it will not appear in the options menu. Use the **Upload** button to add the XSLT to the policy.

Step 4 Click **Save Changes** to commit your changes to the working policy.

For more information on transformations, see Chapter 13, "Transforming Messages with XSLT."

Processing Request Arguments for SOAP RPC Services

On routes between HTTP Get Argument and SOAP RPC service descriptors, the ACE XML Gateway processes arguments similarly as for SOAP Document target services. However, the argument source document is the ACE XML Gateway generates is slightly different.

When converting URL arguments for SOAP RPC service descriptors:

- The SOAP Method value configured for the service descriptor is used as the first child element of the SOAP Body element.
- Each GET argument in the incoming request appears as an element in the outgoing request, with the argument value as the contents of the element.

Notice that the RPC encoding performed by the ACE XML Gateway matches the SOAP RPC convention as defined in the SOAP 1.1 and 1.2 specifications. In particular, it matches the subset that is compliant with the WSI (Web Service Interoperability) Basic Profile 1.0.

For more information, see:

- SOAP 1.1 Encoding description (http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383512)

- SOAP 1.2 (http://www.w3.org/TR/2003/REC-soap12-part2-20030624/#soapforrpc)
- WSI BP 1.0: (http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html)

For a request to a HTTP Get handler that routes to a SOAP RPC service descriptor that has a SOAP Method value of submitPO, therefore, the following request results in the argument document shown in Example 5-3:

http://example.com/get2?item=1231&loc=CA

Example 5-3 Sample URL Arguments Rendered as SOAP RPC Requests

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body
 soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <submitPO>
     <item>123251</item>
     <location>CA</location>
   </submitPO>
  </soap:Body>
</soap:Envelope>
```

For SOAP RPC target services, this message may be sufficient in its default form. However, as for SOAP Document services, you can apply an XSLT to the message to process it further.

For more information on argument processing, see "Processing Request Arguments for SOAP Document Services" section on page 5-53.

Next Steps

After deploying the policy, you can test your initial configuration using the test browser built into the ACE XML Manager web console. In the context of a handler, the test browser sends a request to the consumer interface (using the **Test Handler** button). In a service descriptor, the test browser sends requests directly to the backend service (**Test This Service** button). For more information, see "Sending a Test Message" section on page 4-29.

After creating handlers, routes and service descriptors to define the external service and consumer interfaces, as described in this chapter, you can add rules and processing tasks for the services.