CentraSite

Virtualized Services in CentraSite Control

Version 9.5 SP1

November 2013

This document applies to CentraSite Version 9.5 SP1.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: IINM-UG-VIRTUAL-95SP1-20140410

Table of Contents

Preface

This document describes how to:

- Create, configure and deploy virtualized services to the webMethods Mediator policy-enforcement point.
- Create run-time policies and apply them to virtualized services in order to govern the virtualized services' run-time execution.
- Create consumer applications, which specify the consumers that are allowed to consume services and other assets at run time.

The content is organized under the following sections:

Introduction to Virtualized Services	Describes the virtualized service, which is one of the predefined asset types that CentraSite supports out-of-the-box.
Who Can Create and Manage Virtualized Services?	Describes the users who can create and manage virtualized services.
Creating Virtualized Services	Describes how to create a virtualized service.
Configuring SOAP-based Virtual Services	Describes how to configure the processing steps for a SOAP-based virtual service.
Configuring REST or XML Virtual Services	Describes how to configure the processing steps for a REST or XML virtual service.
Invoking webMethods IS Services in Virtualized Services	Describes how to create a webMethods Integration Server (IS) flow service that you can invoke in virtualized services.
Using Context Variables in Virtualized Services	Describes how to use predefined and custom context variables in virtualized services.
Viewing or Editing the Profiles of Virtualized Services	Describes how to view or modify the profiles of a virtualized service.
Important Considerations when Configuring SOAP-based Virtual Services	Discusses some things you should consider when you configure SOAP-based virtual services.
Important Considerations when Configuring REST or XML Virtual Services	Discusses some things you should consider when you configure REST or XML virtual services.
Creating Run-Time Policies	Describes how to create run-time policies and apply them to virtualized services in order to govern the virtualized services' run-time execution.
Consumer Applications	Describes how to create consumer applications, which specify the consumers that are allowed to consume services and other assets at run time.
Deploying Virtualized Services and Consumer Applications	Describes how to deploy virtualized services and consumer applications to webMethods Mediator.

1 Introduction to Virtualized Services

A virtualized service is a service that runs on webMethods Mediator and acts as the consumer-facing proxy for a native service that runs elsewhere on the network. You can create a virtualized service for a SOAP-based Web service, a REST service or an XML service. A virtualized service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing location, protocol and format independence between the consuming application and the provider service.

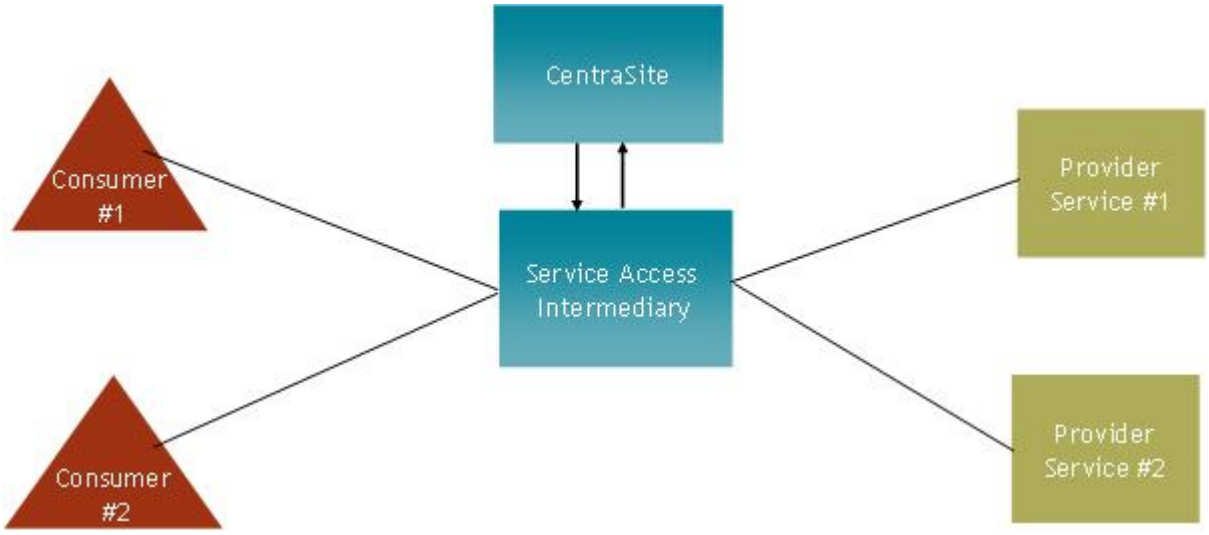
CentraSite supports three types of virtualized services, all of which are predefined asset types that CentraSite supports out-of-the-box:

Virtualized Service Type	Description
Virtual Service	These are virtualized SOAP-based Web services.
Virtual REST Service	These are virtualized REST services.
Virtual XML Service	These are virtualized XML services.

This document uses the term "virtualized service" when referring to the three types of virtualized services in general.

You deploy virtualized services to the webMethods Mediator policy-enforcement point. When a virtualized service is deployed, consumers access this proxy service instead of the actual native service. The virtualized service forwards the request to the appropriate back-end service and, if it has been configured to do so, performs additional mediation functions such as message transformation, protocol bridging, load balancing and failover handling.

The figure below shows webMethods Mediator (the "service access intermediary") mediating traffic between consumers and virtualized services. Mediator provides loose coupling between consumers and providers, and is able to identify the consumers who are accessing the provider services by examining the traffic flowing through it.



A virtualized service is an asset that you create and store in the Asset Catalog. You should virtualize each service that you want to deploy to webMethods Mediator. You can virtualize a service from scratch or make a virtual copy of an existing service, and then configure the following information in the virtualized service definition (VSD):

You configure a virtualized service by configuring its processing steps, as follows:

- The Entry Protocol Step.
  - For a SOAP-based virtual service, you specify the protocol (HTTP , HTTPS or JMS) and SOAP format (1.1 or 1.2) of the requests that the service will accept.
  - For a virtual REST or XML service, you specify the protocol (HTTP or HTTPS) of the requests that the service will accept. You also specify the HTTP methods (GET, POST, PUT, DELETE) that the virtual service should be allowed to perform on a REST resource.
- The Request Processing Step, in which you specify optional custom processing for requests responses (such as message transformation or pre-processing).
- The Response Processing Step, in which you specify optional custom processing for responses (such as message transformation, post-processing or error messaging instructions).
- The Routing Rules Step, in which you specify the manner in which to route the requests (e.g., directly to the native service, or routed according to your routing rules, or routed to a pool of servers for the purpose of load balancing or failover handling). You also specify the protocol used for authenticating the requests (for example, the credentials specified in the request's HTTP header).

You can create one or more virtualized services for a single SOAP-based Web service, a REST service or an XML service. For example, if you have different authentication or routing requirements for two different sets of consumers, you would create a different virtualized service for each set of consumers.

2 Who Can Create and Manage Virtualized Services?

To create and manage virtualized services (of any type), you must belong to a role with the following permissions:

- Create Assets —OR— Manage Assets
- Manage Runtime Policies (required to configure virtual services)
- Manage Runtime Targets (required to deploy virtual services)

Note: If a user has View permission on a service and "Create Assets" permission within their own organization, he or she can virtualize the particular service. However, the user will not be permitted to configure the processing steps for the service that is virtualized unless he or she also has the "Manage Runtime Policies" permission for their organization. Only users with "Manage Runtime Policies" permission can configure these steps. Consider identifying a small group of users who will be responsible for configuring the processing steps for a service, and give this group a role that includes the "Manage Runtime Policies" permission. Because these users might configure virtualized services that other users have created, they will also need Modify permission on the actual virtualized services. To ensure that these users have access to the virtualized services that they need to configure, consider creating a design/change-time policy that automatically gives this group of users Modify permission on a service when it is virtualized.

For more information about roles and permissions, see the section *Users, Groups, Roles, and Permissions*.

3 Creating Virtualized Services

Generally speaking, you should ensure that the following conditions are satisfied before you create a virtualized service:

- Ensure that the interface for the native service is completely implemented and that interface is reflected in the WSDL or schema file that is registered for the service in the CentraSite registry.
- An instance of the service is deployed and running at a known point in network.
- The metadata for the native service is valid and up-to-date. If the metadata for the native service has not been completely specified or is out-of-date, you should update it before you generate the virtual service so that you do not carry inaccurate/incomplete data into the virtualized service.

There are two ways to virtualize a service:

Creating a Virtualized Service from Scratch

You can create a virtualized Web Service, REST Service or XML Service *from scratch*, meaning you manually create the virtualized service and set its attributes.

To create a virtualized service from scratch

1 In CentraSite Control, go to **Asset Catalog > Browse**.

2 Click the **Add Asset** button.

3 In the Add Asset dialog, specify the following attributes:

In this field...	Do the following...
Type	The type of virtualized service that you want to create (i.e., Virtual Service, Virtual REST Service or Virtual XML Service).
Name	A name for the new virtualized service. Unlike native services, the names of virtualized services cannot contain spaces or special characters (except _ and -). Consequently, if you adopt a convention that involves using the name of the service as part of the virtualized service name, then the names of the services themselves must not contain characters that are invalid in virtualized service names. If you want to change the name of a virtual service later, make sure the service in undeployed, and then change the name in the service's detail page.
Description	*Optional*. A description for the virtualized service. This description appears when a user displays a list of virtualized services in the user interface.
Organization	Specify the organization to which this virtualized service will be added. **Note:** The **Organization** list contains the names of all organizations for which you have "Manage Assets" permission.

In this field...	Do the following...
	If you select an organization other than your own organization, you will nevertheless be the owner of the virtualized service. **Note:** Choose the organization with care. You cannot change the organization assignment after the virtualized service is added to the catalog. You can, however, export a virtualized service from one organization and import it to another.

4 Click **OK**. The virtualized service is added to the Asset Catalog. The asset's details page is displayed.

> **Important:** During virtualization of an asset, CentraSite will not allow you to add the virtualized asset to the asset catalog unless you have specified all "required" attributes in the asset's type definition and all referenced objects to which the asset has an association. The value for the "required" attribute and referenced object must be specified in the virtualized service's profile in order to add the asset to the asset catalog.

5 View or edit the virtualized service's profiles as described in *Viewing or Editing the Profiles of Virtualized Services*.

6 Configure the virtualized service's processing steps as described in either **Configuring SOAP-based Virtual Services** or **Configuring REST or XML Virtual Services**.

Virtualizing an Existing Service

You use the **Virtualize** option to virtualize an existing Web Service, REST Service or XML Service in CentraSite Control.

To virtualize an existing service

1 In CentraSite Control, display the service for that you want to create a virtual copy. For procedures, see the section *Using the Asset Catalog*.

2 On the asset detail page, click **Actions** and then select **Virtualize**.

> **Important:** Be aware that CentraSite will not allow you to virtualize a service if it does not contain an associated WSDL or schema file in the registry.

3 In the **Virtualize <name of the type> Service** wizard, specify the following fields:

In this field...	Specify...
Name	A name for the new virtualized service. Unlike services, the names of virtualized services cannot contain spaces or special characters (except _ and -). Consequently, if you adopt a convention that involves using the name of the service as part of the virtualized service name, then the names of the services themselves must not contain characters that are invalid in virtualized service names. If you want to change the name of a virtual service later, make sure the service in undeployed, and then change the name in the service's detail page.
Description	*Optional*. A description for the virtualized service. This description appears when a user displays a list of virtualized services in the user interface.
Organization	Specify the organization to which this virtualized service will be added. **Note:** The **Organization** list contains the names of all organizations for which you have "Manage Assets" permission. If you select an organization other than your own organization, you will nevertheless be the owner of the virtualized service. **Note:** Choose the organization with care. You cannot change the organization assignment after the virtualized service is added to the catalog. You can, however, export a virtualized service from one organization and import it to another.

Version	*Optional*. A version identifier for the virtualized service. The version identifier is an optional attribute that you can increment when a service is modified to indicate that the service has been updated. You can use any versioning scheme you choose. The version identifier does not need to be numeric. This is the "public" version identifier that CentraSite Control shows to users when it displays the list of services.
	
	Examples:
	
	`0.0a`
	`1.0.0 (beta)`
	`Pre-release 001`
	`V1-2007.04.30`
	
	You can later create new versions of the virtual service (see the section *Using the Asset Catalog*).
	
	In addition, CentraSite automatically generates a system version number, which is visible on the virtualized service detail page. The system version number is independent from the version number you specify here. For more information, see the section *Using the Asset Catalog*.

Create a Run-Time Policy	Use this checkbox to specify the behavior for the run-time policy (or policies) associated with the virtualized service. If you select the checkbox, the run-time policy or policies associated with the virtualized service will be created automatically when the service is virtualized. If you do not select the checkbox, the run-time policy or policies will not be created when the service is virtualized.

In this field...	Specify...
	You can only select the checkbox if you have the "Manage Runtime Policies" organization-level permission; without this permission, the checkbox is deactivated.

4 If you have attributes specified in the service for virtualizing, click **Next** to go to the attribute mapping dialog.

5 In the **Attributes** dialog, choose the attributes of the service that you want to copy to the virtualized service.

Notes:

1. If you choose a parent attribute, by default *all* its child attributes are copied as well. You can de-select any child attribute.
2. Similarly, if you de-select a parent attribute, all its child attributes are de-selected as well.
3. Attributes defined as **required** in the asset's type definition and all referenced objects to which the asset has an association will be internally copied from the asset to the virtualized service. These attributes will remain selected and disabled by default in the attribute mapping dialog.

6 Click **Finish**.

The virtualized service is created in the Asset Catalog and its detail page is displayed. The WSDL or XML interface and the entry/exit protocol (e.g., HTTP, HTTPS or JMS) will be identical to the service.

7 View or edit the virtualized service's profiles as described in *Viewing or Editing the Profiles of Virtualized Services*.

8 Configure the virtualized service's processing steps as described in either **Configuring SOAP-based Virtual Services** or **Configuring REST or XML Virtual Services**.

Restriction

Even if a Service asset type contains one or more custom profiles, a virtualized service of the asset type Service will never include any of these custom profiles in the catalog.

4 Configuring SOAP-based Virtual Services

The CentraSite Control user interface enables you to configure the following processing steps for a SOAP-based virtual service:

The Entry Protocol Step (SOAP)

The Entry Protocol step specifies the protocol (HTTP, HTTPS or JMS) and SOAP format (1.1 or 1.2) of the requests that the virtual service will accept.

This step allows you to bridge protocols between the consuming application and the native service. For example, suppose you have a native service that is exposed over JMS and a consuming application that submits SOAP requests over HTTP. In this situation, you can configure the virtual service's Entry Protocol step to accept HTTP requests and configure its Routing Protocols step to route the request to the Web service using JMS.

Besides using the Entry Protocol step to resolve protocol differences between the consumer and the native service, you might use this step to intentionally expose a virtual service over a particular protocol. For example, if you have a native service that is exposed over HTTP, you might expose the virtual service over JMS simply to gain the asynchronous-messaging and guaranteed-delivery benefits that one gains by using JMS as the message transport.

Use the following procedure to configure the Entry Protocol step of a virtual service.

To configure the Entry Protocol step

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Entry Protocol** tab and specify the protocol (HTTP, HTTPS or JMS) for the virtual service to accept requests.

Note: CentraSite supports HTTP version 1.1 only.

In this field...	Specify...
Protocol	The protocol (HTTP, HTTPS or JMS) over which the virtual service will accept requests. Note that you can select *both* **HTTP** and **HTTPS** if needed. **Important:** Before you deploy a service over HTTPS, ensure that the Integration Server on which the Mediator is running has been configured for SSL. In addition, make sure you specify an HTTPS port in the Mediator's Ports Configuration page. (In the Integration Server Administrator, go to **Solutions > Mediator > Administration > General** and specify the port in the **HTTPS Ports Configuration**

In this field...	Specify...
	field.) For details on the Port Configuration page, see the section *Configuring Mediator* in the document *Administering webMethods Mediator*.)
JMS Provider Alias	If you selected **JMS**, specify the Integration Server's JMS Trigger name. The alias should include the JNDI destination name and the JMS connection factory.
Format	The SOAP format (**SOAP 1.1** or **SOAP 1.2**) of the requests that the virtual service will accept.

The Request Processing Step (SOAP)

The Request Processing step specifies how the SOAP request message is to be transformed or pre-processed before it is submitted to the native service.

As long as a consumer sends a SOAP request to the correct virtual service endpoint, and the request includes a soapAction header, then Mediator can detect the correct service and operation; in this case, no message transformation is required. However, in some cases a virtual service might need to transform SOAP messages.

For example, you might need to accommodate differences between the message content that a consuming application is capable of submitting and the message content that a native service expects. For example, if the consuming application submits an order record using a slightly different structure than the structure expected by the native service, you can use the Request Processing step to transform the record submitted by the consuming application to the structure required by the Web service.

Specifically, you would need to configure the virtual service to:

- Transform or pre-process the request messages into the format required by the native service, before Mediator sends the requests to the native services. Additionally in this case, the transformation is required if the virtual service has a schema validation policy, which validates the requests.
- Transform or pre-process the native service's response messages into the format required by the consumer applications, before Mediator returns the responses to the consumer applications.

There are two ways to transform or pre-process a message:

- By passing the message to an XSLT transformation file.
- By passing the message to a webMethods Integration Server service.

Use the following procedure to configure the Request Processing step of a virtual service.

To configure the Request Processing step

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Request Processing** tab. On this tab you can specify one or more **Transform** steps and one or more **webMethods IS Service** steps as follows.

1. Click **Add Step**, select one of the following kinds of request processing steps and click **OK**.

Request Processing Step	Description
Transform	Configure this step if you want to perform an XSLT message transformation on the request message before it is submitted to the native service. **Important:** The XSL file uploaded by the user should *not* contain the XML declaration in it (e.g., `xml version="1.0" encoding="UTF-8"`). This is because when the virtual service is deployed to Mediator, Mediator embeds the XSL file in the virtual service definition (VSD), and since the VSD itself is in XML format, there cannot be an XML declaration line in the middle of it. This can lead to unexpected deployment issues which can be avoided by making sure the XSL file does not contain the declaration line.
webMethods IS Service	Configure this step if you want to invoke a webMethods IS service to preprocess the request before it is submitted to the native service. For more information, see *Invoking webMethods IS Services in Virtual Services*.

2. In the **Step** list, click the step (**Transform** or **webMethods IS Service**) and complete the appropriate dialog box as follows.

For this step...	Do the following...
Transform	Click **Browse**, select the XSLT transformation file from your file system and click **OK**. **Note:** If you make changes to the XSLT file in the future, you must re-deploy the virtual service.
webMethods IS Service	Type the fully qualified service name or, to display a list of webMethods IS services that are published to CentraSite, type a keyword phrase in the **Service** field. The wildcard character * is supported. For example, to return all IS services that start with `Test`, type `Test*`. (The list that appears also identifies the application server instance on which each service is located.) Then select one or more services to be used to manipulate the request (the axis2 MessageContext instance) and click **OK**. Mediator will pass to the invoked IS service the request message context (the axis2 MessageContext instance), which contains the request-specific information. Thus, you can use the public IS services that accept MessageContext as input to

For this step...	Do the following...
	manipulate the response contents. For more information, see *Invoking webMethods IS Services in Virtual Services*. **Note:** The webMethods IS service must be running on the same Integration Server as Mediator.

3. Configure additional request processing steps if desired, and click **Save**.

Note: Specify the steps in the order in which they should be invoked. Use the arrow buttons to rearrange the sequence of steps. To delete a step, select the check box next to the step and click **Delete**.

The Response Processing Step (SOAP)

The Response Processing step is similar to the Request Processing step. This step specifies how the response message from the native service provider is to be transformed or processed before it is returned to the consuming application.

You can configure this step to:

- Return a custom error message (and/or the native provider's service fault content) to the consuming application when the native provider returns a service fault. In addition, you can invoke one or more webMethods IS services to process the error message before and/or after the custom error message is added to the response.
- Perform message transformations using a specified XSLT file.
- Process the response message by passing it to a **webMethods IS service**.

Use the following procedure to configure the Response Processing step of a virtual service.

To configure the Response Processing step

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Response Processing** tab. On this tab you can specify one **Error Messaging** step, one or more **Transform** steps and one or more **webMethods IS Service** steps as follows.

Note: You may configure and test a virtual service without specifying response processing. Then you may go back later and specify response processing, if desired.

1. Click **Add Step**, select one of the following steps and click **OK**.

Step	Description
Error Messaging	You use this step to configure error responses for this particular virtual service. Alternatively, you can configure global error responses for *all* virtual services, using Mediator's Service Fault Configuration page, as described in the document *Administering webMethods Mediator*. The precedence of the Error Messaging instructions is as follows: ■ If you configure an Error Messaging step for a virtual service, the error messaging step takes precedence over any settings on the global Service Fault Configuration page. ■ If you do not create an Error Messaging step for a virtual service, the settings on the Service Fault Configuration page take precedence.
Transform	Configure this step if you want to invoke an XSLT transformation file to transform the SOAP response payloads from XML format to the format required by the consumer.
webMethods IS Service	Configure this step if you want to invoke a webMethods IS service to process the response message from the native service before it is returned to the consuming application. For more information, see ***Invoking webMethods IS Services in Virtual Services***.

2. In the **Step** list, click **Error Messaging** and configure it as follows.

   You use this step to configure error responses for this particular virtual service. Select one or both of the following options:

   Send Native Provider Fault: When you select this option, Mediator returns the native service provider's fault content (if available) to the consuming application. Mediator will send whatever content it received from the native service provider. If you select this option, the **Response** option is ignored when a fault is returned by the native service provider. (Faults returned by internal Mediator exceptions will still be handled by the **Response** option.)

   Response: When you select this option, Mediator returns the following fault response to the consuming application:

```
Mediator encountered an error:$ERROR_MESSAGE while executing operation:$OPERATION ↩
service:$SERVICE at time:$TIME on date:$DATE. The client ip was:$CLIENT_IP. The ↩
current user:$USER. The consumer application:$CONSUMER_APPLICATION".
```

   This fault response is returned in both of the following cases:

   ■ When a fault is returned by the native service provider.

     In this case, the `$ERROR_MESSAGE` variable in the fault response will contain the message produced by the provider's exception that caused the error. This is equivalent to the `getMessage` call on the Java Exception. This maps to the **faultString** element for SOAP 1.1 or the **Reason**

element for SOAP 1.2 catch. Mediator discards the native service provider's fault and does not return this content to the web service caller since it could be considered a security issue, especially if the native provider is returning a stack trace with its response.

- When a fault is returned by internal Mediator exceptions (such as policy violation errors, timeouts, etc.).

  In this case, `$ERROR_MESSAGE` will contain the error message generated by Mediator.

The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.) which are described in the table below.

You can customize the default fault response using the following substitution variables, where Mediator replaces the variable reference with the real content at run time:

- The predefined context variables listed in ***The Predefined Context Variables***.
- Custom context variables that you declare using Mediator's API (see ***The API for Context Variables***).

Note: If you want to reference a custom context variable that you have already defined in a **context-based routing rule** (as opposed to one you have **declared using Mediator's API**), then you must add the prefix `$mx` to the variable name in order to reference the variable. For example, if you defined the variable TAXID, you would reference it as `$mx:TAXID`.

The fault handler variables are described below.

Note: If no value is defined for a fault handler variable, then the returned value will be the literal string "null". For example, `$CONSUMER_APPLICATION` will always be "null" if the service's policy does not contain the Identify Consumer action.

Fault Handler Variable	Description
`$ERROR_MESSAGE`	The error message produced by the exception that is causing the error. This is equivalent to the getMessage call on the Java Exception. This maps to the **faultString** element for SOAP 1.1 or the **Reason** element for SOAP 1.2 catch.
`$OPERATION`	The operation that was invoked when this error occurred.
`$SERVICE`	The service that was invoked when this error occurred.
`$TIME`	The date (as determined on the Container side) at which the error occurred.
`$DATE`	The date (as determined on the Container side) at which the error occurred.
`$CLIENT_IP`	The IP address of the client invoking the service. This might be available for only certain invoking protocols, such as HTTP(S).
`$USER`	The currently authenticated user. The user will be present only if the Transport/SOAP Message have user credentials.
`$CONSUMER_APPLICATION`	The currently identified consumer application.

Pre-Processing: Optional. Configure this step if you want to invoke one or more webMethods IS services to manipulate the response message before the **Error Messaging** step is invoked. The IS service will have access to the response message context (the axis2 MessageContext instance) before it is updated with the custom error message. For example, you might want to send emails or perform custom alerts based on the response payload. For more information, see *Invoking webMethods IS Services in Virtual Services*.

Post-Processing: Optional. Configure this step if you want to invoke one or more webMethods IS services to manipulate the service fault after the **Error Messaging** step is invoked. The IS service will have access to the entire service fault and the custom error message. You can make further changes to the fault message structure, if needed. For more information, see *Invoking webMethods IS Services in Virtual Services*.

Note: You can specify the Error Messaging step only once.

3. In the **Step** list, click **Transform**, click **Browse** and select the XSLT transformation file from your file system and click **OK**. If you make changes to the XSLT file in the future, you must redeploy the virtual service.
4. In the **Step** list, click **webMethods IS Service** and configure it as follows.

   Type the fully qualified service name or, to display a list of webMethods IS services that are published to CentraSite, type a keyword phrase in the **Service** field. The wildcard character * is supported. For example, to return all IS services that start with `Test`, type `Test*`. (The list that appears also identifies the application server instance on which each service is located.) Then select one or more services to be used to manipulate the response (the axis2 MessageContext instance) and click **OK**.

   Mediator will pass to the invoked IS service the response message context (the axis2 MessageContext instance), which contains the response-specific information. Thus, you can use the public IS services that accept MessageContext as input to manipulate the response contents. For more information, see *Invoking webMethods IS Services in Virtual Services*.

   Note: The webMethods IS service must be running on the same Integration Server as Mediator.

5. Configure additional response processing steps if desired, and click **Save**.

   Note: Specify the steps in the order in which they should be invoked. Use the arrow buttons to rearrange the sequence of steps. To delete a step, select the check box next to the step and click **Delete**.

The Routing Protocols Step (SOAP)

You can choose to configure one of the following routing protocols for a SOAP-based virtual service:

- "Straight Through" Routing (SOAP)
- "Content-based" Routing (SOAP)
- "Context-based" Routing (SOAP)
- "Load Balancing" Routing (SOAP)
- The Routing Protocol For Services Exposed over JMS (SOAP)

"Straight Through" Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Straight Through" routing protocol. (Alternatively, you can choose "Content-based", "Context-based" or "Load Balancing" routing.)

When you select the "Straight Through" routing protocol, the virtual service will route the requests directly to the native service endpoint you specify. You may specify how to authenticate requests (as with all routing protocols).

Alternatively, you can choose the "Content-Based", "Context-Based" or "Load Balancing" routing protocol.

To configure "Straight Through" routing

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

Field	Description
HTTP or **JMS**	Select **HTTP**.
Routing Type	Select **Straight Through**
Default To	Click the **Endpoint** button and select the URL of the native service to route the request to. For example: `http://mycontainer/creditCheckService` Alternatively, Mediator offers “Local Optimization” capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:

Field	Description

	`local://<Service-full-path>`
	
	OR
	
	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

Configure Endpoint Properties icon	This icon displays a dialog box that enables you to configure a set of properties for the endpoint as follows:
	
	■ **SOAP Optimization Method**: Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests:
	  ■ **MTOM**: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service.
	  ■ **SwA**: Indicates that Mediator expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service.
	  ■ **None** (the default).
	
	Note:
	
	1. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received.
	2. When sending SOAP requests that do *not* contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an **IS service callout** that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly.
	
	■ **WSS Header Customization**: Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service.
	  ■ **Pass all security headers**: Passes the security header, even if it is processed by Mediator (i.e., even if Mediator processes the header according to the virtual service's security run-time policy).
	
	Note: If the virtual service does not contain a security run-time policy, and the `mustUnderstand` attribute of the security header is 0/false, then Mediator will *always* forward the security header to the native service.

Field	Description
	■ **Remove processed security header from request before routing**: Removes the security header if it is processed by Mediator (i.e., if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will *not* remove the security header if *both* of the following conditions are true: 1) Mediator did not process the security header, and 2) the `mustUnderstand` attribute of the security header is 0/false). ■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *`Integration Server_directory`*`\packages\WmMediator\config\resources\pg-config.properties` . The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *`Integration Server_directory`*`\packages\WmMediator\config\resources\pg-config.properties` . The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.
HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Field	Description
	NTLM (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options: ■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *Integration Server_directory*`\config` directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests.

Field	Description
	■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

"Content-based" Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Content-based" routing protocol. (Alternatively, you can choose "Straight Through", "Context-based" or "Load Balancing" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the "Content-Based" routing protocol to route specific types of messages to specific endpoints.

You can route messages to different endpoints based on specific values that appear in the request message. You might use this capability, for example, to determine which operation the consuming application has requested, and route requests for complex operations to an endpoint on a fast machine.

The requests are routed according to the content-based routing rules you create. That is, they are routed based on the successful evaluation of one or more XPath expressions that are constructed utilizing the content of the request payload. For example, a routing rule might allow requests for half of the methods of a particular service to be routed to Service A, and the remaining methods to be routed to Service B.

You may also specify how to authenticate requests (as with all routing protocols).

If a SOAP request contains a WS-Security header, Mediator passes it to the native service.

To configure "Content-Based" routing

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab, specify the following fields and click **Save**.

Field	Description
HTTP or **JMS**	Select **HTTP**.
Routing Type	Select **Content Based**.
Routing Rules	To create a routing rule, do the following: 1. Click the **Endpoint** button (next to the **Route To** column).

Field	Description
	2. In the **Search for Endpoint** dialog that appears, click the **Search** button to search for the Web service endpoint to route the requests to. 3. Then select a service and click **OK**. 4. Click the **Configure Endpoint Properties** icon (next to the **Endpoint** button) if you want to configure a set of properties for each endpoint individually. In the dialog box, click the endpoint you want to configure and specify the following fields: ■ **SOAP Optimization Method**: Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests: ■ **MTOM**: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service. ■ **SwA**: Indicates that Mediator expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service. ■ **None** (the default). **Note:** a. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received. b. When sending SOAP requests that do *not* contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an **IS service callout** that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly. ■ **WSS Header Customization**: Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service. ■ **Pass all security headers**: Passes the security header, even if it is processed by Mediator (i.e., even if Mediator processes the header according to the virtual service's security run-time policy). **Note:** If the virtual service does not contain a security run-time policy, and the `mustUnderstand` attribute of the security header is 0/false, then Mediator will *always* forward the security header to the native service. ■ **Remove processed security header from request before routing**: Removes the security header if it is processed by Mediator (i.e., if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will *not* remove the security header if *both* of the following conditions are true: 1) Mediator did not process the security header, and 2) the `mustUnderstand` attribute of the security header is 0/false).

Field	Description
	■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *Integration Server_directory*\packages\WmMediator\config\resources\pg-config.properties . The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *Integration Server_directory*\packages\WmMediator\config\resources\pg-config.properties . The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication. To create an XPath expression for the routing rule, do the following: 1. Click the **Edit** button (next to the **If True** column). 2. In the XPath Editor that appears, view the **Namespace Map** tab, which displays all predefined namespaces (e.g., soapenv, soapenc, etc.). If you want to add custom namespaces, click **Add Custom Namespace/prefix**, specify a name and value for the namespace and click **OK**. If you need to add additional rows, use the plus button. 3. In the XPath Editor's **All Nodes** tab, expand the namespace's node, choose the method you want for the XPath expression, and click **OK**. 4. In the XPath Editor's **XPATH Evaluator** tab, evaluate the XPath expression by specifying an argument in the **XPath Expression** field, and then click **Evaluate**. 5. After you have evaluated the XPath expression, click **OK**.
Default To	A native service endpoint to route the request to in case all **Routing Rules** evaluate to False. Click the **Search for Endpoint** button next to the **Default To** field and select the URL of the native service to route the request to. For example:

Field	Description

	`http://mycontainer/creditCheckService`
	
	To specify additional services, use the plus button next to the field to add rows.
	
	Alternatively, Mediator offers "Local Optimization" capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	
	`local://<Service-full-path>`
	
	OR
	
	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options:

Field	Description
	■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *Integration Server_directory*`\config` directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

"Context-based" Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Context-based" routing protocol. (Alternatively, you can choose "Straight Through", "Content-based" or "Load Balancing" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the Context-Based protocol to route specific types of messages to specific endpoints.

The requests are routed according to the context-based routing rules you create. A routing rule specifies where requests should be routed, and the criteria by which they should be routed there. For example, requests can be routed according to certain consumers, certain dates/times, or according to requests that exceed/fall below a specified metric (Total Count, Success Count, Fault Count, etc.). You can create one or more rules.

You may also specify how to authenticate requests (as with all routing protocols).

If a SOAP request contains a WS-Security header, Mediator passes it to the native service.

To configure "Context-Based" Routing

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

Field	Description
HTTP or **JMS**	Select **HTTP**.
Routing Type	Select **Context Based**.
Routing Rules	To create a routing rule, click the **Add Rule** button and complete the **Add Routing Rule** dialog box as follows. 1. In the **Variable** column, select **Time**, **IP Address**, **Date**, **Consumer**, **Predefined Context Variable** or **Custom Context Variable**. For more information, see ***Using Context Variables in Virtual Services***. 2. In the **Value** column, specify an applicable value. For **Date** choose **Before**, **After** or **Equal To** and enter a date. For **Time** choose **Before** or **After** and enter a time. For **IP Address**, enter numeric values for **Between** and **And**. For **Consumer**, click **Browse** and select a consumer application name. For **Predefined Context Variable** or **Custom Context Variable**, choose the **String** or **Integer** data type. Select a predefined variable name or custom variable name from the drop-down list. For **String,** choose **Equal To** or **Not Equal To** and enter a value. For **Integer**, choose **Greater Than**, **Less Than**, **Not Equal To**, **Equal To** or and enter a value. **Note:** a. For the list of the predefined context variables, see ***Using Context Variables in Virtual Services***. b. The predefined context variable PROTOCOL_HEADER is not available in the drop-down list; to include PROTOCOL_HEADER in the rule, define the variable as Custom Context Variable. For more information, see ***The API for Context Variables***. c. If you define a custom context variable in the routing rule, you must write a **webMethods IS service** and invoke it in the virtual service's Request Processing step. In this Integration Server service, use the API to get/set the custom context variable. For more information, see ***The API for Context Variables***. 3. If you need to specify multiple variables, use the plus button to add rows. 4. In the **Combination Uses** field, choose an operator for the expression: **AND** (the default) or **OR**.

Field	Description
	5. In the **Route To** field, click the **Endpoint** button and choose the URL of the native service to route the request to, if the rule criteria are met. 6. Click the **Configure Endpoint Properties** icon (next to the **Endpoint** button) if you want to configure a set of properties for each endpoint individually. In the dialog box, click the endpoint you want to configure and specify the following fields: ■ **SOAP Optimization Method**: Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests: ■ **MTOM**: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service. ■ **SwA**: Indicates that Mediator expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service. ■ **None** (the default). **Note:** a. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received. b. When sending SOAP requests that do *not* contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an **IS service callout** that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly. ■ **WSS Header Customization**: Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service. ■ **Pass all security headers**: Passes the security header, even if it is processed by Mediator (i.e., even if Mediator processes the header according to the virtual service's security run-time policy). **Note:** If the virtual service does not contain a security run-time policy, and the `mustUnderstand` attribute of the security header is 0/false, then Mediator will *always* forward the security header to the native service. ■ **Remove processed security header from request before routing**: Removes the security header if it is processed by Mediator (i.e., if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will *not* remove the security header if *both* of the following conditions are true: 1) Mediator did not process the security header, and 2) the `mustUnderstand` attribute of the security header is 0/false).

Field	Description
	■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *`Integration Server_directory`*`\packages\WmMediator\config\resources\pg-config.properties`. The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *`Integration Server_directory`*`\packages\WmMediator\config\resources\pg-config.properties`. The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication. 7. Click **OK**.

Route To	A native service endpoint to route the request to in case all routing rules evaluate to False. Click the **Search for Endpoint** button and select the URL of the native service to route the request to. For example:
	`http://mycontainer/creditCheckService`
	To specify additional services, use the plus button next to the field to add rows.
	Alternatively, Mediator offers "Local Optimization" capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	`local://<Service-full-path>`
	OR

Field	Description

	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options: ■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:**

Field	Description
	1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *Integration Server_directory*`\config` directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

"Load Balancing" Routing (SOAP)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Load Balancing" routing protocol. (Alternatively, you can choose "Straight Through", "Content-Based" or "Context-Based" routing.)

If you have a Web service that is hosted at two or more endpoints, you can use the Load Balancing option to distribute requests among the endpoints.

Requests are distributed across multiple endpoints. The requests are intelligently routed based on the "round-robin" execution strategy. The load for a service is balanced by directing requests to two or more services in a pool, until the optimum level is achieved. The application routes requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.

Load-balanced endpoints also have automatic Failover capability. If a load-balanced endpoint is unavailable (for example, if a connection is refused), then that endpoint is marked as "down" for the number of seconds you specify in the "Suspend the Failed Endpoint" field (during which the endpoint will not be used for sending the request), and the next configured endpoint is tried. If all the configured load-balanced endpoints are down, then a SOAP fault is sent back to the client. After the suspension period expires, each endpoint marked will be available again to send the request.

To configure "Load Balancing" routing

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

In this field...	Specify...
HTTP or **JMS**	**HTTP**.
Routing Type	**Load Balancing**

Route To	The URLs of two or more services in a pool to which the requests will be routed. The application routes the requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.
	
	To specify the first service, click the **Endpoint** button and select the URL of the Web service to route the request to. For example:
	
	`http://mycontainer/creditCheckService`
	
	To specify additional services, use the plus button next to the field to add rows.
	
	Alternatively, Mediator offers "Local Optimization" capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	
	`local://<Service-full-path>`
	
	OR
	
	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

Configure Endpoint Properties icon	This button displays a dialog box that enables you to configure a single set of properties that will be shared by all the endpoints. In the dialog box, specify the following fields: ■ **SOAP Optimization Method**: Optional. Mediator can accept the following optimization methods to optimize the payloads of SOAP requests: ■ **MTOM**: Indicates that Mediator expects to receive a request with a Message Transmission Optimization Mechanism (MTOM) attachment, and will forward the attachment to the native service.

In this field...	Specify...
	■ **SwA**: Indicates that Mediator expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service. ■ **None** (the default). **Note:** 1. Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, Mediator can only forward SwA to the native provider. The same is true for MTOM, and applies to responses received from the native provider. That is, an SwA or MTOM response received by Mediator from a native provider will be forwarded to the caller using the same format it received. 2. When sending SOAP requests that do *not* contain a MTOM or SWA attachment to a virtual service for a native provider endpoint that returns an MTOM or SWA response, the request 'Accept' header must be set to 'multipart/related' (or the virtual service's Request Processing Step should include an **IS service callout** that sets the BUILDER_TYPE context variable to 'multipart/related'). This is necessary so Mediator knows how to parse the response properly. ■ **WSS Header Customization**: Indicates whether Mediator should pass the WS-Security headers of the incoming requests to the native service. ■ **Pass all security headers**: Passes the security header, even if it is processed by Mediator (i.e., even if Mediator processes the header according to the virtual service's security run-time policy). **Note:** If the virtual service does not contain a security run-time policy, and the `mustUnderstand` attribute of the security header is 0/false, then Mediator will *always* forward the security header to the native service. ■ **Remove processed security header from request before routing**: Removes the security header if it is processed by Mediator (i.e., if Mediator processes the header according to the virtual service's security run-time policy). Note that Mediator will *not* remove the security header if *both* of the following conditions are true: 1) Mediator did not process the security header, and 2) the `mustUnderstand` attribute of the security header is 0/false). ■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *Integration Server_directory**packages**WmMediator**config**resources**pg-config.properties* . The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *Integration Server_directory**packages**WmMediator**config**resources**pg-config.properties* . The default of that property is 30 seconds.

In this field...	Specify...
	■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.
Suspend the Failed Endpoint	A numeric timeout value (in seconds). Default: 30. When this timeout value expires, the system routes the execution of the virtual service to the next consecutive Web service endpoint specified in the **Route To** field.
HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options:

In this field...	Specify...
	■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *Integration Server_directory**config* directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

The Routing Protocol For Services Exposed over JMS (SOAP)

You can use the Routing Protocols step to specify a JMS queue to which the Mediator is to submit the request, and the destination to which the native service is to return the response.

To configure the Routing Protocols step for virtual services exposed over JMS

1 In CentraSite Control, display the virtual service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab and select the **JMS** button. On this tab specify the following JMS routing protocol fields, and click **Save**.

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

In this field...	Specify...
JMS URI	A connection alias for connecting to the JMS provider (e.g., an Integration Server alias or a JNDI URL). For example, a JNDI URL of the form: `jms:queue:dynamicQueues/MyRequestQueue?wm-wsendpointalias=MediatorConsumer&targetService=vs-jms-in-echo` Note that the `wm-wsendpointalias` parameter is required for Integration Server/Mediator to look up the JMS consumer alias to send the request to the specified queue (e.g., `MyRequestQueue`), which is a dynamic queue in ActiveMQ. Also, the `targetService` parameter is required if sending to another virtual service that uses JMS as the entry protocol.
Priority	Optional. A numeric value that specifies the priority of the JMS message in the queue.
Reply to Destination	Optional. A queue name for the incoming JMS request.
Time to Live	Optional. A numeric value (in milliseconds) that specifies the lifespan of the JMS message.
Delivery Mode	Optional. The type of message delivery to the endpoint. ■ **None** (default). ■ **Persistent**: The message is stored by the JMS server before delivering it to the consumer. ■ **Non-Persistent**: The message is not stored before delivery.
Message Properties	The message properties to use. ■ **Use Existing** (default): Use existing properties. ■ **Customize**: Specify one or more property names and values. To add additional rows, use the plus button.
JMS Headers	The JMS headers that you want to use to authenticate the requests. ■ **Use Existing** (default): Use existing headers. ■ **Customize**: Specify one or more header names and values. To add additional rows, use the plus button.

5 Configuring REST or XML Virtual Services

The CentraSite Control user interface enables you to configure the following processing steps for a REST or XML virtual service:

The Entry Protocol Step (REST or XML)

The Entry Protocol step specifies:

- The protocol (HTTP or HTTPS) over which the REST or XML virtual service should accept requests.
- The HTTP methods (GET, POST, PUT, DELETE) that the virtual service should be allowed to perform on a REST resource.

This step allows you to bridge protocols between the consuming application and the native service. For example, suppose you have a native service that is exposed over HTTPS and a consuming application that submits SOAP requests over HTTP. In this situation, you can configure the virtual service's Entry Step to accept HTTP requests and configure its Routing Rule step to route the request to the Web service using HTTPS.

Important: It is important that the client's requests contain an HTTP Content-Type header. At run time, Mediator determines which message builder to use based on the message's HTTP method and its Content-Type. If a client does not specify the accept Content-Type in the request, by default Mediator sets the accept Content-Type to application/xml. For a list of valid HTTP method/Content-Type combinations, see *The Request Message's HTTP Methods and Content-Types for REST and XML Services*. For information about the JSON Content-Type, see *Working With the JSON Content-Type*.

To configure the Entry Protocol step for a REST or XML virtual service

1 In CentraSite Control, display the REST or XML virtual service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Entry Protocol** tab. On this tab specify the following fields and click **Save**.

In this field...	Specify...
Protocol	The protocol (HTTP and/or HTTPS) over which the virtual REST service or virtual XML service accepts requests. You can select *both* **HTTP** and **HTTPS** if needed. **Note:** CentraSite supports HTTP version 1.1 only. **Important:** Before you deploy a service over HTTPS, ensure that the Integration Server on which the Mediator is running has been configured for SSL. In addition, make sure you specify an HTTPS port in the Mediator's Ports Configuration page. (In the Integration

In this field...	Specify...
	Server Administrator, go to "Solutions > Mediator > Administration > General" and specify the port in the **HTTPS Ports Configuration** field.) For details on the Port Configuration page, see the section *Configuring Mediator* in the document *Administering webMethods Mediator*.)
HTTP Methods	The HTTP methods (GET, POST, PUT, DELETE) that the virtual service should be allowed to perform on a REST resource. It is important to specify all the HTTP methods that are supported for the service. For example, if the virtual service is deployed to Mediator and you selected only the GET method in its Entry Protocol step, then Mediator will only permit GET invocations. In this case, POST requests will be rejected with a return of statusCode 405 even if the native service happens to support POSTs.

The Request Processing Step (REST or XML)

As long as a consumer sends a REST or XML request with the proper Content-Type (see ***The Request Message's HTTP Methods and Content-Types for REST and XML Services***), and as long as the HTTP method and endpoint URI are in the expected form, then Mediator can determine the correct service and operation; in this case, no message transformation is required. However, in some cases a virtual REST service or virtual XML service might need to transform XML messages.

For example, you might need to accommodate differences between the message content that a consuming application is capable of submitting and the message content that a native service expects. For example, if the consuming application submits an order record using a slightly different structure than the structure expected by the native service, you can use the Request Processing step to transform the record submitted by the consuming application to the structure required by the service.

Specifically, you would need to configure the virtual REST service or virtual XML service to:

- Transform or pre-process the request messages into the format required by the native service, before Mediator sends the requests to the native services. Additionally in this case, the transformation is required if the virtual REST service or virtual XML service has a schema validation policy, which validates the requests.
- Transform or pre-process the native service's response messages into the format required by the consumer applications, before Mediator returns the responses to the consumer applications.

There are two ways to transform or pre-process a message:

- By passing the message to an XSLT transformation file.
- By passing the message to a webMethods IS service.

Use the following procedure to configure the Request Processing step of a virtual REST service or virtual XML service.

To configure the Request Processing step

1 In CentraSite Control, display the virtual REST service or virtual XML service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Request Processing** tab. On this tab you can specify one or more **Transform** steps and one or more **webMethods IS Service** steps as follows.

1. Click **Add Step**, select one of the following kinds of request processing steps and click **OK**.

Request Processing Step	Description
Transform	Configure this step if you want to perform an XSLT message transformation on the request message before it is submitted to the native service. **Important:** The XSL file uploaded by the user should *not* contain the XML declaration in it (e.g., `xml version="1.0" encoding="UTF-8"`). This is because when the virtual service is deployed to Mediator, Mediator embeds the XSL file in the virtual service definition (VSD), and since the VSD itself is in XML format, there cannot be an XML declaration line in the middle of it. This can lead to unexpected deployment issues which can be avoided by making sure the XSL file does not contain the declaration line.
webMethods IS Service	Configure this step if you want to invoke a webMethods IS service to preprocess the request before it is submitted to the native service. For more information, see ***Invoking webMethods IS Services in Virtual Services***.

2. In the **Step** list, click the step (**Transform** or **webMethods IS Service**) and complete the appropriate dialog box as follows.

For this step...	Do the following...
Transform	Click **Browse**, select the XSLT transformation file from your file system and click **OK**. **Note:** If you make changes to the XSLT file in the future, you must re-deploy the virtual REST service or virtual XML service.
webMethods IS Service	Type the fully qualified service name or, to display a list of webMethods IS services that are published to CentraSite, type a keyword phrase in the **Service** field. The wildcard character * is supported. For example, to return all IS services that start with `Test`, type `Test*`. (The list that appears also identifies the application server instance on which each service is located.) Then select one or more services to be used to manipulate the request (the axis2 MessageContext instance) and click **OK**.

For this step...	Do the following...
	Mediator will pass to the invoked IS service the request message context (the axis2 MessageContext instance), which contains the request-specific information. Thus, you can use the public IS services that accept MessageContext as input to manipulate the response contents. For more information, see *Invoking webMethods IS Services in Virtual Services*. **Note:** The webMethods IS service must be running on the same Integration Server as Mediator.

3. Configure additional request processing steps if desired, and click **Save**.

Note: Specify the steps in the order in which they should be invoked. Use the arrow buttons to rearrange the sequence of steps. To delete a step, select the check box next to the step and click **Delete**.

The Response Processing Step (REST or XML)

The Response Processing step is similar to the Request Processing step. This step specifies how the response message from the native service provider is to be transformed or processed before it is returned to the consuming application.

You can configure this step to:

- Return a custom error message (and/or the native provider's service fault content) to the consuming application when the native provider returns a service fault. In addition, you can invoke one or more webMethods IS services to process the error message before and/or after the custom error message is added to the response.
- Perform message transformations using a specified XSLT file.
- Process the response message by passing it to a **webMethods IS service**.

Use the following procedure to configure the Response Processing step of a virtual REST service or virtual XML service.

To configure the Response Processing step

1 In CentraSite Control, display the virtual REST service or virtual XML service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Response Processing** tab. On this tab you can specify one **Error Messaging** step, one or more **Transform** steps and one or more **webMethods IS Service** steps as follows.

Note: You may configure and test a virtual REST service or virtual XML service without specifying response processing. Then you may go back later and specify response processing, if desired.

1. Click **Add Step**, select one of the following steps and click **OK**.

Step	Description
Error Messaging	You use this step to configure error responses for this particular virtual service. Alternatively, you can configure global error responses for *all* virtual services, using Mediator's Service Fault Configuration screen, as described in the document *Administering webMethods Mediator*. The precedence of the Error Messaging instructions is as follows: ■ If you configure an Error Messaging step for a virtual service, the error messaging step takes precedence over any settings on the global Service Fault Configuration page. ■ If you do not create an Error Messaging step for a virtual service, the settings on the Service Fault Configuration page take precedence.
Transform	Configure this step if you want to invoke an XSLT transformation file to transform the response payloads from XML format to the format required by the consumer.
webMethods IS Service	Configure this step if you want to invoke one or more webMethods IS services to process the response message from the native service before it is returned to the consuming application. For more information, see ***Invoking webMethods IS Services in Virtual Services***.

2. In the **Step** list, click **Error Messaging** and configure it as follows.

   You use this step to configure error responses for this particular virtual service. Select one or both of the following options:

   Send Native Provider Fault: When you select this option, Mediator returns the native service provider's fault content (if available) to the consuming application. Mediator will send whatever content it received from the native service provider. If you select this option, the **Response** option is ignored when a fault is returned by the native service provider. (Faults returned by internal Mediator exceptions will still be handled by the **Response** option.)

   Response: When you select this option, Mediator returns the following fault response to the consuming application:

```
Mediator encountered an error:$ERROR_MESSAGE while executing operation:$OPERATION ↩
service:$SERVICE at time:$TIME on date:$DATE. The client ip was:$CLIENT_IP. The ↩
current user:$USER. The consumer application:$CONSUMER_APPLICATION".
```

This fault response is returned in both of the following cases:

- When a fault is returned by the native service provider.

  In this case, the `$ERROR_MESSAGE` variable in the fault response will contain the message produced by the provider's exception that caused the error. This is equivalent to the `getMessage` call on the Java Exception. This maps to the **faultString** element for SOAP 1.1 or the **Reason** element for SOAP 1.2 catch. For REST service calls, the message is returned inside an `</Exception>` tag. Mediator discards the native service provider's fault and does not return this content to the web service caller since it could be considered a security issue, especially if the native provider is returning a stack trace with its response.

- When a fault is returned by internal Mediator exceptions (such as policy violation errors, timeouts, etc.).

  In this case, `$ERROR_MESSAGE` will contain the error message generated by Mediator.

The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.) which are described in the table below.

You can customize the default fault response using the following substitution variables, where Mediator replaces the variable reference with the real content at run time:

- The predefined context variables listed in *The Predefined Context Variables*.
- Custom context variables that you declare using Mediator's API (see *The API for Context Variables*).

> **Note:** If you want to reference a custom context variable that you have already defined in a **context-based routing rule** (as opposed to one you have **declared using Mediator's API**), then you must add the prefix `$mx` to the variable name in order to reference the variable. For example, if you defined the variable TAXID, you would reference it as `$mx:TAXID`.

The fault handler variables are described below.

> **Note:** If no value is defined for a fault handler variable, then the returned value will be the literal string "null". For example, `$CONSUMER_APPLICATION` will always be "null" if the service's policy does not contain the Identify Consumer action.

Fault Handler Variable	Description
$ERROR_MESSAGE	The error message produced by the exception that is causing the error. This is equivalent to the getMessage call on the Java Exception. This maps to the **faultString** element for SOAP 1.1 or the **Reason** element for SOAP 1.2 catch. For REST service calls, the message is returned inside an </Exception> tag.
$OPERATION	The operation that was invoked when this error occurred.
$SERVICE	The service that was invoked when this error occurred.
$TIME	The date (as determined on the Container side) at which the error occurred.
$DATE	The date (as determined on the Container side) at which the error occurred.
$CLIENT_IP	The IP address of the client invoking the service. This might be available for only certain invoking protocols, such as HTTP(S).
$USER	The currently authenticated user. The user will be present only if the Transport/SOAP Message have user credentials.
$CONSUMER_APPLICATION	The currently identified consumer application.

Pre-Processing: Optional. Configure this step if you want to invoke one or more webMethods IS services to manipulate the response message before the **Error Messaging** step is invoked. The IS service will have access to the response message context (the axis2 MessageContext instance) before it is updated with the custom error message. For example, you might want to send emails or perform custom alerts based on the response payload. For more information, see *Invoking webMethods IS Services in Virtual Services*.

Post-Processing: Optional. Configure this step if you want to invoke one or more webMethods IS services to manipulate the service fault after the **Error Messaging** step is invoked. The IS service will have access to the entire service fault and the custom error message. You can make further changes to the fault message structure, if needed. For more information, see *Invoking webMethods IS Services in Virtual Services*.

Notes:

a. Unlike with SOAP specifications, there is no agreed upon format to suggest an error condition for XML and REST services (there is no element nested in a </soap:Fault> element nested in a </soap:Body>). Mediator assumes that XML and REST services will follow HTTP conventions and return responses with return codes in the 200-299 range when service calls are successful. This is the only way Mediator can determine that a native provider's response should be interpreted as a failure.

b. You can specify the Error Messaging step only once.

3. In the **Step** list, click **Transform**, click **Browse** and select the XSLT transformation file from your file system and click **OK**. If you make changes to the XSLT file in the future, you must redeploy the virtual service.
4. In the **Step** list, click **webMethods IS Service** and configure it as follows.

Type the fully qualified service name or, to display a list of webMethods IS services that are published to CentraSite, type a keyword phrase in the **Service** field. The wildcard character * is supported. For example, to return all IS services that start with `Test`, type `Test*`. (The list that appears also identifies the application server instance on which each service is located.) Then select one or more services to be used to manipulate the response (the axis2 MessageContext instance) and click **OK**.

Mediator will pass to the invoked IS service the response message context (the axis2 Message-Context instance), which contains the response-specific information. Thus, you can use the public IS services that accept MessageContext as input to manipulate the response contents. For more information, see ***Invoking webMethods IS Services in Virtual Services***.

Note: The webMethods IS service must be running on the same Integration Server as Mediator.

5. Configure additional response processing steps if desired, and click **Save**.

Note: Specify the steps in the order in which they should be invoked. Use the arrow buttons to rearrange the sequence of steps. To delete a step, select the check box next to the step and click **Delete**.

The Routing Protocols Step (REST or XML)

You can choose to configure one of the following routing protocols for a REST or XML virtual service:

- The "Straight Through" Routing Protocol Step (REST or XML)
- The "Content-based" Routing Protocol Step (REST or XML)
- The "Context-based" Routing Protocol Step (REST or XML)
- The "Load Balancing" Routing Protocol Step (REST or XML)

The "Straight Through" Routing Protocol Step (REST or XML)

If your Entry Protocol is HTTP or HTTPS, you can choose to use the "Straight Through" routing protocol. (Alternatively, you can choose "Content-Based", "Context-Based" or "Load Balancing" routing.)

When you select the "Straight Through" routing protocol, the virtual REST service or virtual XML service will route the requests directly to the native service endpoint you specify. You may specify how to authenticate requests.

To configure "Straight Through" routing

1 In CentraSite Control, display the virtual REST service or virtual XML service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab. On this tab specify the following fields, and click **Save**.

Field	Description
Service Type	Select **XML** (for a native REST or XML service).
Routing Type	Specify **Straight Through**
HTTP Method	The HTTP method to pass to the native service. Typically you want to pass each request to the native service with the same HTTP method that is contained in the request. For example, if a request contains a GET method, you allow the GET method to be passed to the native service. In this case, select the value **Use Context Variable**. This variable contains the HTTP method that is contained in the request. However, in other cases you might want to change the HTTP method of a request to a different HTTP method. In this case, you can specify the different method explicitly (by selecting the GET, POST, PUT or DELETE value), or you can specify the different method dynamically on a per-request basis. If you want to specify the method dynamically, select the **Use Context Variable** option and then write a webMethods IS service to set the value of the context variable. For more information, see ***Changing the HTTP Method of a REST or XML Request***.

Default To	Click the **Endpoint** button and select the URL of the native service to route the request to. For example:
	
	`http://mycontainer/creditCheckService`
	
	Alternatively, Mediator offers “Local Optimization” capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	
	`local://<Service-full-path>`
	
	OR
	
	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

Field	Description
Configure Endpoint Properties icon	This icon displays a dialog box that enables you to configure a set of properties for the endpoint as follows: ■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *Integration Server_directory**packages**WmMediator**config**resources**pg-config.properties* . The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *Integration Server_directory**packages**WmMediator**config**resources**pg-config.properties* . The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.
HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its

Field	Description
	list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options: ■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *`Integration Server_directory\config`* directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	Specify the HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

The "Content-based" Routing Protocol Step (REST or XML)

If your Entry Protocol is HTTP, you can choose to use the "Content-Based" routing protocol. (Alternatively, you can choose "Straight Through", "Context-Based" or "Load Balancing" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the Content-Based routing protocol to route specific types of messages to specific endpoints.

You can route messages to different endpoints based on specific values that appear in the request message. You might use this capability, for example, to determine which operation the consuming application has requested, and route requests for complex operations to an endpoint on a fast machine.

The requests are routed according to the content-based routing rules you create. That is, they are routed based on the successful evaluation of one or more XPath expressions that are constructed utilizing the content of the request payload. For example, a routing rule might allow requests for half of the methods of a particular service to be routed to Service A, and the remaining methods to be routed to Service B.

You may also specify how to authenticate requests (as with all routing protocols).

To configure "Content-Based" routing

1 In CentraSite Control, display the virtual REST service or virtual XML service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab.

4 Specify values for the remaining fields on the **Routing Protocols** page as follows:

Field	Description
Service Type	Select **XML** (for a native REST or XML service).
Routing Type	Select **Content Based**.
Routing Rules	You can create one or more routing rules. To create a routing rule, do the following: 1. Click the **Endpoint** button (next to the **Route To** column). 2. In the **Search for Endpoint** dialog that appears, click the **Search** button to search for the native service endpoint to route the requests to, select a service and click **OK**. 3. Click the **Configure Endpoint Properties** icon (next to the **Endpoint** button) if you want to configure a set of properties for each endpoint individually. In the dialog box, click the endpoint you want to configure and specify the following fields: ■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property

Field	Description
	`pg.endpoint.connectionTimeout` located in the file *`Integration Server_directory`*`\packages\WmMediator\config\resources\pg-config.properties`. The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *`Integration Server_directory`*`\packages\WmMediator\config\resources\pg-config.properties`. The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication. To create an XPath expression for the routing rule, do the following: 1. Click the **Edit** button (next to the **If True** column). 2. In the XPath Editor that appears, view the **Namespace Map** tab, which displays all predefined namespaces (e.g., soapenv, soapenc, etc.). If you want to add custom namespaces, click **Add Custom Namespace/prefix**, specify a name and value for the namespace and click **OK**. If you need to add additional rows, use the plus button. 3. In the XPath Editor's **All Nodes** tab, expand the namespace's node, choose the method you want for the XPath expression, and click **OK**. 4. In the XPath Editor's **XPATH Evaluator** tab, evaluate the XPath expression by specifying an argument in the **XPath Expression** field, and then click **Evaluate**. 5. After you have evaluated the XPath expression, click **OK**.
HTTP Method	The HTTP method to pass to the native service. Typically you want to pass each request to the native service with the same HTTP method that is contained in the request. For example, if a request contains a GET method, you allow the GET method to be passed to the native service. In this case,

Field	Description
	select the value **Use Context Variable**. This variable contains the HTTP method that is contained in the request. However, in other cases you might want to change the HTTP method of a request to a different HTTP method. In this case, you can specify the different method explicitly (by selecting the GET, POST, PUT or DELETE value), or you can specify the different method dynamically on a per-request basis. If you want to specify the method dynamically, select the **Use Context Variable** option and then write a webMethods IS service to set the value of the context variable. For more information, see *Changing the HTTP Method of a REST or XML Request*.

Default To	A native service endpoint to route the request to in case all **Routing Rules** evaluate to False. Click the **Search for Endpoint** button next to the **Default To** field and select the URL of the native service to route the request to. For example:
	
	`http://mycontainer/creditCheckService`
	
	Alternatively, Mediator offers "Local Optimization" capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	
	`local://<Service-full-path>`
	
	OR
	
	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its

Field	Description
	list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options: ■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *`Integration Server_directory\config`* directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

The "Context-based" Routing Protocol Step (REST or XML)

If your Entry Protocol is HTTP, you can choose to use the "Context-Based" routing protocol. (Alternatively, you can choose "Straight Through", "Content-Based" or "Load Balancing" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the Context-Based protocol to route specific types of messages to specific endpoints.

The requests are routed according to the context-based routing rules you create. A routing rule specifies where requests should be routed, and the criteria by which they should be routed there. For example, requests can be routed according to certain consumers, certain dates/times, or according to requests that exceed/fall below a specified metric (Total Count, Success Count, Fault Count, etc.). You can create one or more rules.

You may also specify how to authenticate requests (as with all routing protocols).

To configure "Context-Based" Routing

1 In CentraSite Control, display the virtual REST service or virtual XML service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

Field	Description
Service Type	Select **XML** (for a native REST or XML service).
Routing Type	**Context Based**
Routing Rules	To create a routing rule, click the **Add Rule** button and complete the **Add Routing Rule** dialog box as follows: 1. In the **Variable** column, select **Time**, **IP Address**, **Date**, **Consumer**, **Predefined Context Variable** or **Custom Context Variable**. For more information, see ***Using Context Variables in Virtual Services***. 2. In the **Value** column, specify an applicable value. For **Time** choose **Before** or **After** and enter a time. For **IP Address**, enter a range of numeric IP addresses in the v4 or v6 format for **Between** and **And**. For **Date** choose **Before**, **After** or **Equal To** and select a date. For **Consumer**, click **Browse** and select a consumer application name. For **Predefined Context Variable** or **Custom Context Variable**, choose the **String** or **Integer** data type. Select a predefined variable name or custom variable name from the drop-down list. For **String,** choose **Equal To** or **Not Equal To** and enter a value. For **Integer**, choose **Greater Than**, **Less Than**, **Equal To** or **Not Equal To** and enter a value. **Note:**

Field	Description
	a. For the list of the predefined context variables, see ***Using Context Variables in Virtual Services***. b. The predefined context variable PROTOCOL_HEADER is not available in the drop-down list; to include PROTOCOL_HEADER in the rule, define the variable as Custom Context Variable. For more information, see ***The API for Context Variables***. c. If you define a custom context variable in the routing rule, you must write a **webMethods IS service** and invoke it in the virtual REST service or virtual XML service Request Processing step. In this Integration Server service, use the API to get/set the custom context variable. For more information, see ***The API for Context Variables***. 3. If you need to specify multiple variables, use the plus button to add rows. 4. In the **Combination Uses** field, choose an operator for the expression: **AND** (the default) or **OR**. 5. In the **Route To** field, click the **Endpoint** button and choose the URL of the native service to route the request to, if the rule criteria are met. 6. Click the **Configure Endpoint Properties** icon (next to the **Endpoint** button) if you want to configure a set of properties for each endpoint individually. In the dialog box, click the endpoint you want to configure and specify the following fields: ■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *`Integration Server_directory`*`\`*packages*`\WmMediator\config\resources\`*pg-config.properties*. The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *`Integration Server_directory`*`\`*packages*`\WmMediator\config\resources\`*pg-config.properties*. The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur. **Note:** SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses.

Field	Description
	■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication. 7. Click **OK**.
HTTP Method	The HTTP method to pass to the native service. Typically you want to pass each request to the native service with the same HTTP method that is contained in the request. For example, if a request contains a GET method, you allow the GET method to be passed to the native service. In this case, select the value **Use Context Variable**. This variable contains the HTTP method that is contained in the request. However, in other cases you might want to change the HTTP method of a request to a different HTTP method. In this case, you can specify the different method explicitly (by selecting the GET, POST, PUT or DELETE value), or you can specify the different method dynamically on a per-request basis. If you want to specify the method dynamically, select the **Use Context Variable** option and then write a webMethods IS service to set the value of the context variable. For more information, see ***Changing the HTTP Method of a REST or XML Request***.

Default To	A native service endpoint to route the request to in case all routing rules evaluate to False. Click the **Endpoint** button and select the URL of the native service to route the request to. For example:
	
	`http://mycontainer/creditCheckService`
	
	Alternatively, Mediator offers "Local Optimization" capability if the native service and the virtual service (in Mediator) are located on the same machine. With local optimization, service invocation happens in-memory and not through a network hop. In the **Default To** field the Routing Protocols tab, specify the native service in either of the following forms:
	
	`local://<Service-full-path>`
	
	OR
	
	`local://<server>:<port>/ws/<Service-full-path>`
	
	For example:
	
	`local://MediatorTestServices:NewMediatorTestServices_Port`
	
	which points to the endpoint service NewMediatorTestServices_Port which is present under the folder MediatorTestServices in Integration Server.

HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options:

Field	Description
	■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options: ■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server. ■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *`Integration Server_directory`*`\config` directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option:

Field	Description
	■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

The "Load Balancing" Routing Protocol Step (REST or XML)

If your Entry Protocol is HTTP, you can choose to use the "Load Balancing" routing protocol. (Alternatively, you can choose "Straight Through", "Content-Based" or "Context-Based" routing.)

If you have a native service that is hosted at two or more endpoints, you can use the Load Balancing option to distribute requests among the endpoints.

Requests are distributed across multiple endpoints. The requests are intelligently routed based on the "round-robin" execution strategy. The load for a service is balanced by directing requests to two or more services in a pool, until the optimum level is achieved. The application routes requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding.

Load-balanced endpoints also have automatic Failover capability. If a load-balanced endpoint is unavailable (for example, if a connection is refused), then that endpoint is marked as "down" for the number of seconds you specify in the "Suspend the Failed Endpoint" field (during which the endpoint will not be used for sending the request), and the next configured endpoint is tried. If all the configured load-balanced endpoints are down, then a SOAP fault is sent back to the client. After the suspension period expires, each endpoint marked will be available again to send the request.

Use the following procedure to configure "Load Balancing" routing of a virtual REST service or virtual XML service.

To configure "Load Balancing" routing

1 In CentraSite Control, display the virtual REST service or virtual XML service detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Processing Steps** profile.

3 Select the **Routing Protocols** tab, specify the following fields, and click **Save**.

Field	Description
Service Type	Select **XML** (for a native REST or XML service).
Routing Type	Select **Load Balancing**.
HTTP Method	The HTTP method to pass to the native service. Typically you want to pass each request to the native service with the same HTTP method that is contained in the request. For example, if a request contains a GET method, you allow the GET method to be passed to the native service. In this case, select the value **Use Context Variable**. This variable contains the HTTP method that is contained in the request. However, in other cases you might want to change the HTTP method of a request to a different HTTP method. In this case, you can specify the different method explicitly (by selecting the GET, POST, PUT or DELETE value), or you can specify the different method dynamically on a per-request basis. If you want to specify the method dynamically, select the **Use Context Variable** option and then write a webMethods IS service to set the value of the context variable. For more information, see ***Changing the HTTP Method of a REST or XML Request***.
Route To	The URLs of two or more services in a pool to which the requests will be routed. The application routes the requests to services in the pool sequentially, starting from the first to the last service without considering the individual performance of the services. After the requests have been forwarded to all the services in the pool, the first service is chosen for the next loop of forwarding. To specify the first service, click **Endpoint** and select the URL of the Web service to route the request to. For example: `http://mycontainer/creditCheckService` To specify additional services, use the plus button next to the field to add rows.
Configure Endpoint Properties icon	This icon displays a dialog box that enables you to configure a single set of properties that will be shared by all the endpoints. In the dialog box, specify the following fields: ■ **HTTP Connection Timeout**: The time interval (in seconds) after which a connection attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.connectionTimeout` located in the file *`Integration Server_directory`*`\`*packages*`\WmMediator\config\resources\pg-config.properties` . The default of that property is 30 seconds. ■ **Read Timeout**: The time interval (in seconds) after which a socket read attempt will timeout. If a value is not specified (or if the value 0 is specified), Mediator will use the value of the global property `pg.endpoint.readTimeout` located in the file *`Integration Server_directory`*`\`*packages*`\WmMediator\config\resources\pg-config.properties` . The default of that property is 30 seconds. ■ **SSL Options**: To enable SSL client authentication for the endpoint, you must specify values for both the **Client Certificate Alias** field and the **IS Keystore Alias** field. If you specify a value for only one of these fields, a deployment error will occur.

Field	Description
	Note: SSL client authentication is optional; you may leave both fields blank. ■ **Client Certificate Alias**: The client's private key to be used for performing SSL client authentication. If you specify a client certificate alias, you must also include in the virtual service's policy the "Require SSL" action and select that action's "Client Certificate Required" option. The "Client Certificate Required" option specifies whether client certificates are required for the purposes of: 1) Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests, and 2) Signing SOAP responses or encrypting SOAP responses. ■ **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which Mediator is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.
Suspend the Failed Endpoint	A numeric timeout value (in seconds). Default: 30. When this timeout value expires, the system routes the execution of the virtual REST service or virtual XML service to the next consecutive native service endpoint specified in the **Route To** field.
HTTP Authentication	**Authentication Scheme**: Specify the mode of authentication: **Basic Authentication** (default), **NTLM** (Windows only), **OAuth2** or **None**. **Basic Authentication**. Select one of the following options: ■ **Use credentials from incoming request**: (default): Authenticates requests based on the credentials specified in the HTTP header. Mediator passes the "Authorization" header present in the original client request to the native service. ■ **Use specified credentials**: Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields. **NTLM** (Currently Windows only). Note that if Mediator is used to access a native service protected by NTLM (which is typically hosted in IIS), then the native service in IIS should be configured to use NTLM as the authentication scheme. If the authentication scheme is configured as "Windows", then "NTLM" should be in its list. The "Negotiate" handshake will be supported in the near future. This note applies to all three of the following options for NTLM: ■ **Use credentials from incoming request**: Default. Mediator uses the user credentials passed in the request header for an NTLM handshake with the server. ■ **Use specified credentials**: Mediator uses the values you specify in the **User**, **Password** and **Domain** fields for an NTLM handshake with the server. ■ **Transparent**: Mediator will behave in "pass by" mode, allowing an NTLM handshake to occur between the client and server. Note: If the client is a WCF application, then the client should be configured with clientCredentialType set to NTLM. **OAuth2**. Select one of the following options: ■ **Use credentials from incoming request**: Default. Mediator will pass the OAuth2 Access token (a "Bearer" type token) unchanged to the native OAuth server.

Field	Description
	■ **Use specified token**: Specify an OAuth access token to be deployed by Mediator. If you select this option, the consumer need not pass the OAuth token during service invocation. Click the **Show Token** button to view the OAuth access token. Users who do not have the permissions to create and manage virtual services will not see this button. For more information, see **Who Can Create and Manage Virtualized Services?**. **Note:** 1. You must set the Integration Server property `watt.server.auth.skipForMediator` to "true" and then restart Integration Server for the change to take effect. This property is located in the server configuration file (server.cnf), which is located in the *Integration Server_directory**config* directory. For details, see the *webMethods Integration Server Administrator's Guide*. 2. The run-time actions "Require HTTP Basic Authentication" and "Identify Consumer" (with the value **HTTP Authentication Token** as the identifier) will not be enforced when using the authentication scheme **OAuth2**. **None**. Select the following option: ■ **Invoke Service Anonymously**: Does not authenticate requests.
HTTP Headers	The HTTP headers that you want to use to authenticate the requests. ■ **Use Existing**: Use the HTTP headers that are contained in the requests. ■ **Customize**: Use the HTTP headers that you specify in the **Name** and **Value** columns below. If you need to specify multiple headers, use the plus button to add rows.

6 Invoking webMethods IS Services in Virtualized Services

A webMethods Integration Server (IS) service is a user-defined Integration Server flow service that you can invoke in:

- Request Processing steps, to preprocess the request message before it is submitted to the native service.
- Response Processing steps, to preprocess the response message from the native service before it is returned to the consuming application.

A webMethods IS service must be running on the same Integration Server as webMethods Mediator. It can call out a C++ or Java or .NET function. It can also call other Integration Server services to manipulate the SOAP message.

The input pipeline for a webMethods IS service should have the following input variables:

- `proxy.name`: This is the name of the virtualized service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope.
- `EnvelopeString`.
- `MessageContext`: Mediator will automatically place a MessageContext variable into the pipeline before executing the webMethods IS service call. MessageContext is of the Java type org.apache.axis2.context.MessageContext.

  Integration Server users can use the Axis2 MessageContext object to manipulate the incoming SOAP request. The Integration Server provides built-in services (i.e., the pub.soap.* services) to work with the MessageContext object to get/set/modify the SOAP body, header, properties, etc. Integration Server users should use these services to extract the information they need from the MessageContext to build the necessary business logic. Users do not need to understand Axis2 or Axiom (the xml object model based on StAX) to work with the SOAP request, because if they are familiar with the Integration Server pub.soap services, they can accomplish most of the tasks. For more information about these related Integration Server services, see the *webMethods Integration Server Built-In Services Reference*.
- `JSONRESTContentString`: Will appear only for REST services with the Content-Type application/json or application/json/badgerfish. For more information, see ***Multiple Root Nodes in JSON REST Services***.
- `UpdatedJSONRESTContentString`: Will appear only for REST services with the Content-Type application/json or application/json/badgerfish. For more information, see ***Multiple Root Nodes in JSON REST Services***.

You can use the following constructs in a webMethods IS service:

- Predefined or custom context variables. For more information, see ***Using Context Variables in Virtual Services***.
- The Security API provided by Mediator (for SOAP-based services only). For more information, see ***Using the Security API in webMethods IS Services***.

Using the Security API in webMethods IS Services

Note: This API is for SOAP-based services only.

Mediator provides Java services that you can use to support WS-Security functionality in a webMethods IS service that you invoke in the Request Processing step.

These services include the following:

- pub.mediator.security.ws:AddUsernameToken
- pub.mediator.security.ws:AddX509Token
- pub.mediator.security.ws:AddSamlSenderVouchesToken
- pub.mediator.security.ws:AddTimestamp
- pub.mediator.addressing:AddWSAddressingHeaders

pub.mediator.security.ws:AddUsernameToken

Adds the WS-Username Token 1.0 and 1.1 to the request. This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

Parameter	Data Type	Required?	Java Type	Description	Default Value
username	String	Yes	String	The value that will be added as the Username element in the token.	""
MessageContext	Object	Yes	`org.apache.axis2.context.MessageContext`	Mediator will place a MessageContext variable into the pipeline before executing the webMethods IS service call.	`org.apache.axis2.context.MessageContext` instance
password	String	No	String	The password for the token; must be specified if the passwordType (see below) is specified as either TEXT or DIGEST.	""
passwordType	String	No	String	Specifies how the password will be added in the token. It takes 3 values: ■ NONE: The password will not be added. ■ TEXT: The password is added in plain text.	NONE

Parameter	Data Type	Required?	Java Type	Description	Default Value
				■ DIGEST: The password is added in digested form (as specified in the UsernameToken profile).	
addNonce	Boolean	No	Boolean	Specifies whether the Nonce element will be added to the token.	False
addCreated	Boolean	No	Boolean	Specifies whether the Created element will be added to the token	False
salt	byte[]	No	byte[]	The value for the /wsse11:UsernameToken/wsse:Salt element. Its value is a 128 bit number serialized as xs:base64Binary.	null
iteration	int	No	Integer	Indicates the number of times the hashing operation is repeated when deriving the key. It is expressed as a xs:unsignedInteger value. If it is not present, a value of 1000 is used for the iteration count.	1000
useMac	Boolean	No	Boolean	Indicates if the derived key will be used as a Message Authentication Code (MAC) or as a symmetric key for encryption.	False
useBasic AuthCredentials	Boolean	No	Boolean	If this parameter is set to True, Mediator will try to use the username and password from the “Authorization” HTTP header. In this case the ‘username’ and ‘password’ fields need not be specified.	False
actor	String	No	String	Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.	""
mustUnderstand	Boolean	No	Boolean	Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it.	False

pub.mediator.security.ws:AddX509Token

Adds a X.509 certificate (or certificate chain) as a BinarySecurityToken (BST) element in the out-bound SOAP request. This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

Parameter	Data Type	Required?	Java Type	Description	Default Value
MessageContext	Object	Yes	`org.apache.axis2.context.MessageContext`	Mediator will place a MessageContext variable into the pipeline before executing the webMethods IS service call.	`org.apache.axis2.context.MessageContext` instance
keystoreFile	String	Yes	String	The absolute path to a keystore file on the system where Mediator is running .	""
keystorePassword	String	Yes	String	The password for the keystore.	""
keystoreType	String	No	String	The type of keystore represented by the file (can be JKS, JCEKS or PKCS12).	JKS
keyAlias	String	Yes	String	The key alias whose X509 certificate will be sent in the soap request as a BST.	""
useCertificatePath	Boolean	No	Boolean	If set to true will use the entire certificate chain represented by the key alias instead of just a single certificate; default is false.	False
actor	String	No	String	Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.	""
mustUnderstand	Boolean	No	Boolean	Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then	False

Parameter	Data Type	Required?	Java Type	Description	Default Value
				this value will not overwrite it.	

pub.mediator.security.ws:AddSamlSenderVouchesToken

This service enables a Security Token Service (STS) client to send a WS-Trust request to a configured STS to obtain a SAML v1/v2 assertion. For the details about configuring Mediator to act as an STS client, see the section *Configuring Mediator* in the document *Administering webMethods Mediator*.

This service adds the obtained SAML assertion to the original request that is sent by the client to the native service, and includes the following parameters.

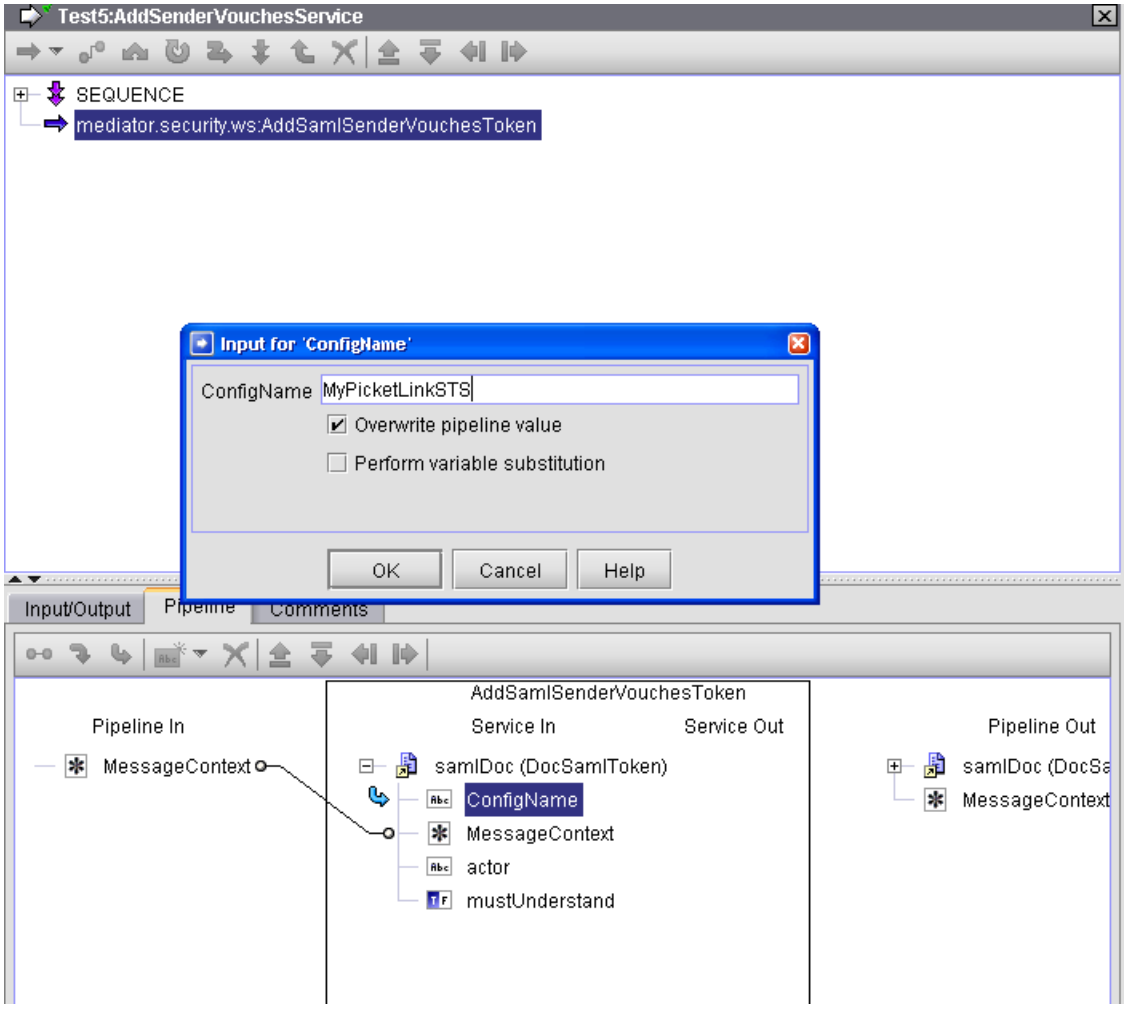
Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

Parameter	Data Type	Required?	Java Type	Description	Default Value
ConfigName	String	Yes	String	References a previously configured STS configuration name.	""
MessageContext	Object	Yes	`org.apache.axis2.context.MessageContext`	Mediator will place a MessageContext variable into the pipeline before executing the webMethods IS service call.	`org.apache.axis2.context.MessageContext` instance
addTimeStamp	Boolean	No	Boolean	Adds a Timestamp element (with the duration specified in timeToLive) to the WS-Security header of the request, and includes it in the signature. (The other items that are signed are the body and SAML assertion.)	False
timeToLive	Integer	No	Integer	If addTimeStamp is true, timeToLive specifies the duration (in seconds) for which the request is valid.	300 seconds (5 minutes)
actor	String	No	String	Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.	""

Parameter	Data Type	Required?	Java Type	Description	Default Value
mustUnderstand	Boolean	No	Boolean	Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it.	False

Example of using AddSamlSenderVouchesToken

The sample service shown below is configured by providing the `MessageContext` and `ConfigName` parameters. The value of `ConfigName` must be the name of a previously configured STS name, which is configured on the Mediator Configuration page.



pub.mediator.security.ws:AddTimestamp

Adds a timestamp to the outbound SOAP request WS-Security header. This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

Parameter	Data Type	Required?	Java Type	Description	Default Value
timeToLive	Integer	No	Integer	Specifies the duration (in seconds) for which the request is valid.	300 seconds (5 minutes)
signTimestamp	Boolean	No	Boolean	Indicates whether the generated timestamp must be signed by Mediator using the configured keystore and signing alias. **Note:** For signTimestamp to work, you must ensure that a valid IS keystore and signing alias are configured in Mediator. For details, see *Configuring Mediator* in the document *Administering webMethods Mediator*.	False
useMilliSecondPrecision	Boolean	No	Boolean	Indicates whether the generated timestamp must have millisecond precision.	True
MessageContext	Object	Yes	`org.apache.axis2.context.MessageContext`	Mediator will place a MessageContext variable into the pipeline before executing the webMethods IS service call.	`org.apache.axis2.context.MessageContext` instance
actor	String	No	String	Indicates the value of the SOAP actor attribute if a new security header is being added to the	""

Parameter	Data Type	Required?	Java Type	Description	Default Value
				SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.	
mustUnderstand	Boolean	No	Boolean	Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it.	False

pub.mediator.addressing:AddWSAddressingHeaders

Adds WS-Addressing headers to a SOAP request sent by the client before Mediator forwards the request to the native service.

This service includes the following input parameters:

Note: For reasons of legibility some of the examples below contain break lines and may not work when pasted into applications or command line tools.

Parameter	Data Type	Required?	Java Type	Description	Default Value
isVersionSubmission	Boolean	No	Boolean	The WS-Addressing version that should be used. ■ If true, the WS-Addressing submission namespace will be used. `http://schemas.xmlsoap.org/ws/2004/08/addressing` ■ If false, the Final specification namespace will be used.	False

Parameter	Data Type	Required?	Java Type	Description	Default Value
				`http://www.w3.org/2005/08/addressing`	

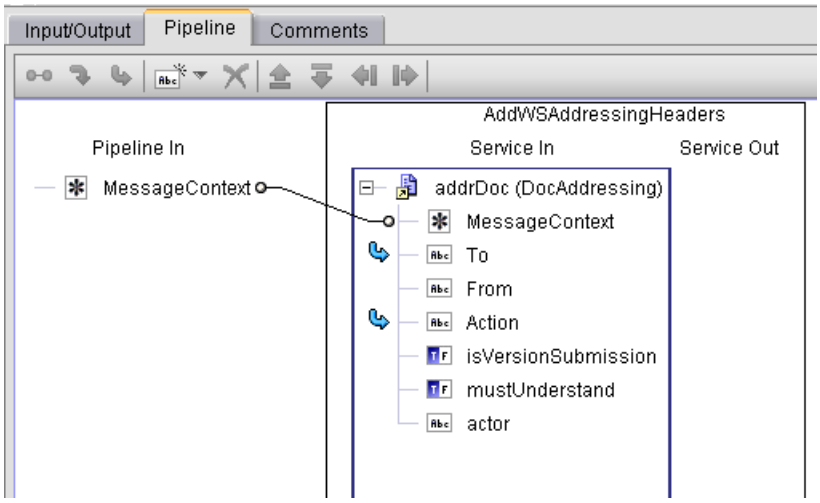
To	String	No	String	This value corresponds to the `/wsa:To` addressing header. You must specify a value that corresponds to the destination of the request message. If this value is not specified, then depending on the `isVersionSubmission` property value, one of the following anonymous EPR values will be sent:
				
				If `isVersionSubmission` is set to true, the anonymous EPR value is:
				
				`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous`
				
				If `isVersionSubmission` is set to false, the anonymous EPR is:
				
				`http://www.w3.org/2005/08/addressing/anonymous`	

From	String	No	String	This value corresponds to the `/wsa:From` addressing header and refers to the source of the message.	""
Action	String	No	String	This value corresponds to the `/wsa:Action` addressing header. By default, this property has the same value as the operation on the virtualized service being invoked (which will usually correspond to the same operation on the native service). But the user can specify a different value corresponding to the native service being called.	URI identifying input operation corresponding to a WSDL port type being called on the virtualized service.

Parameter	Data Type	Required?	Java Type	Description	Default Value
MessageContext	Object	Yes	`org.apache.axis2.context.MessageContext`	Mediator will place a MessageContext variable into the pipeline before executing the webMethods IS service call.	`org.apache.axis2.context.MessageContext` instance
actor	String	No	String	Indicates the value of the SOAP actor attribute if a new security header is being added to the SOAP request. If the request already has a security header with the actor specified in it, then this value will not overwrite it.	""
mustUnderstand	Boolean	No	Boolean	Specifies whether the security header will have the mustUnderstand attribute set to 0 or 1 (false / true). If the security header already has this attribute set, then this value will not overwrite it.	False

Example of using AddWSAddressingHeaders

The sample service shown below is configured by providing the `MessageContext` parameter.



7 Using Context Variables in Virtualized Services

Mediator provides predefined context variables, and you can declare your own custom context variables. You can use both predefined and custom context variables when you configure various processing steps of a virtualized service. Specifically, you can use them:

- In a **webMethods IS service** that you create and invoke during the Request Processing step or the Response Processing step.
- In a routing rule that you create in the Context-Based Routing Protocols step.

This section discusses:

The Predefined Context Variables

You can use the predefined context variables listed below. Any context variable state defined during the inbound request processing steps will still be available during the outbound response processing steps.

Notes:

1. To set, get or remove the predefined context variables, use *The API for Context Variables* provided by Mediator.
2. You do not need to declare the predefined context variables. If you attempt to declare an existing predefined context variable, an error will occur.

Context Variable Display Name	Context Variable Name	Description
Average Response	AVG_SUCCESS_TIME	The average amount of time it took the service to complete all invocations in the current interval. This is measured from the moment Mediator receives the request until the moment it returns the response to the caller. **Note:** By default, Average Response Time does not include metrics for failed invocations. You can include metrics for failed invocations by setting the `pg.PgMetricsFormatter.includeFaults` parameter to true. For details, see the section *Advanced Settings* in the document *Administering webMethods Mediator*.
Client IP Address	INBOUND_IP	The IP address used to send the request to Mediator.
Consumer	CONSUMER_APPLICATION	The name of the consumer application accessing the service, if known.
Fault Count	INTERVAL_FAULT_COUNT	The number of service faults for the interval.
Inbound Content Type	MESSAGE_TYPE	A Content-Type defined in axis2.xml for a message formatter. This value must be a key in the axis2 message

Context Variable Display Name	Context Variable Name	Description
		formatters list, since it is used to control message serialization. (The valid choices are defined as attributes of <messageFormatters/> group in the Integration Server's axis2.xml configuration.)
Inbound HTTP Method	INBOUND_HTTP_METHOD	The HTTP method used by the client to send the request (GET, POST, PUT, DELETE, Use Context Variable).
Inbound Protocol	INBOUND_PROTOCOL	The protocol (HTTP or HTTPS) of the request.
Maximum Response	SLOWEST_SUCCESS_INVOKE	Maximum Response Time. **Note:** By default, Maximum Response Time does not include metrics for failed invocations. You can include metrics for failed invocations by setting the `pg.PgMetricsFormatter.includeFaults` parameter to true. For details, see the section *Advanced Settings* in the document *Administering webMethods Mediator*.
Mediator Host Name	MEDIATOR_HOSTNAME	Mediator host name.
Mediator IP Address	MEDIATOR_IP	Mediator IP address.
Mediator Target Name	TARGET_NAME	Mediator target name.
Minimum Response	FASTEST_SUCCESS_INVOKE	Minimum Response Time. **Note:** By default, Minimum Response Time does not include metrics for failed invocations. You can include metrics for failed invocations by setting the `pg.PgMetricsFormatter.includeFaults` parameter to true. For details, see *Advanced Settings* in the document *Administering webMethods Mediator*.
Outbound HTTP Method	ROUTING_METHOD	The HTTP method to be sent to the native service if the inbound HTTP method is custom. Otherwise, this value will be null. For more information, see ***Changing the HTTP Method of a REST or XML Request***.
Success Count	INTERVAL_SUCCESS_COUNT	The number of success counts for a given service.
Total Count	INTERVAL_TOTAL_COUNT	The total number of counts for a given service.
Virtual Service Name	SERVICE_NAME	Virtual service name.
N/A	BUILDER_TYPE	A Content Type defined in axis2.xml for a message builder. This value must be a key in the axis2 message builders list, since it is used to control building of native service response messages. (The valid choices are defined as

Context Variable Display Name	Context Variable Name	Description
		attributes of <messageBuilders/> group in the Integration Server's axis2.xml configuration.)

N/A	INBOUND_REQUEST_URI	A partial reference to a virtual service (for HTTP/HTTPS only). The protocol, host and port are not part of the value. For example, if the following virtual service is invoked:
		
		`http://mcusawco:5555/ws/TC1`
		
		then the expected value of this variable would be `/ws/TC1`.
		
		For a REST or XML service, the URL might also include query string parameters. For example, if the following virtual service is invoked:
		
		`http://mcusawco:5555/ws/cars?vin=1234`
		
		the expected value of this variable would be `/ws/cars?vin1234`.
		
		This is useful to know because by the time you are able to access the request inside of Mediator, the REST request would contain a top-level element that looks like this:
		
		`<vin>1234</vin>`
		
		So it is not obvious from an XSLT expression or a webMethods IS service callout what part of a REST request came in as a query parameter.
		
		Therefore, using this variable along with INBOUND_HTTP_METHOD and INBOUND_PROTOCOL, you can determine the exact entry point URI that was used when a virtual service was invoked.

N/A	NATIVE_PROVIDER_ERROR	The reason returned by the native service provider in the case where it produced a SOAP fault. This will not contain Mediator-hosted errors such as security policy enforcement errors. This variable will only contain the "reason" text wrapped in a SOAP fault. **Note:** When you are using this variable in a custom SOAP fault message that you are specifying in the Response Processing Step, note the following. If a request is denied due to security policy enforcement, the fault handler variable $ERROR_MESSAGE variable would contain a native service provider error message or other error messages that result from enforced security assertions. However, $NATIVE_PROVIDER_ERROR will be null in

Context Variable Display Name	Context Variable Name	Description
		this case. For information about $ERROR_MESSAGE, see *The Response Processing Step*.
N/A	OPERATION	The virtual service operation selected to execute a request.
N/A	PROTOCOL_HEADERS	Contains a map of key-value pairs in the request, where the values are typed as strings. To get/set this variable, see *pub.mediator.ctxvar:getContextVariable*.
N/A	SOAP_HEADERS	(For use in webMethods IS services only.) Contains an array of the SOAP header elements in the request. To get/set this variable, see *pub.mediator.ctxvar:getContextVariable*.
N/A	USER	The value defined for the Integration Server session executing the request message. If the request is not authenticated, it will use a default unprivileged account. Otherwise, it will set the Integration Server session to the user credentials used for transport security. Also, if credentials were included for message based security (e.g., an X509 token was included and this certificate was mapped to an Integration Server user), then this information would override any transport security (e.g., basic authentication).

Note: When you use predefined context variables in a custom error message in *The Response Processing Step*, note the following:

- To reference a predefined context variable in a custom fault message, you need to prepend a $ symbol to the context variable name to indicate that variable's value should be referenced. Think of this usage as being similar to the '&' address operation for C variables. A predefined context variable expression might look like this: `$USER="Administrator:"`
- The $ reference symbol may appear in the text as needed. (e.g., as a currency symbol). There is no escape concept used with this operator. That is, no special meaning is attached to two occurrences of this symbol: "$$".
- If no value is defined for a valid context variable reference, the string is left unmodified for that context variable.

The API for Context Variables

Mediator provides an API that you can use to:

- Set, get, declare and remove custom context variables.
- Set and get the predefined context variables. (It is not necessary (or even legal) to declare or remove the predefined context variables.)

Mediator provides the following Java services, which are defined in the class ISMediatorRuntimeFacade.java:

- **pub.mediator.ctxvar:getContextVariable**
- **pub.mediator.ctxvar:setContextVariable**
- **pub.mediator.ctxvar:declareContextVariable**
- **pub.mediator.ctxvar:removeContextVariable**

The following sample flow services are described in this section as well:

- **Sample Flow Service: Getting a Context Variable Value**
- **Sample Flow Service: Setting a Context Variable Value**

pub.mediator.ctxvar:getContextVariable

Use this Java service to retrieve a context variable's value and assign it to a pipeline variable. All parameter names are case-sensitive.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by Mediator.	N/A

varName	in	String	Context variable name (predefined or custom).	`PROTOCOL_HEADERS`
				`SOAP_HEADERS`
				`mx:CUSTOM_VAR`

serValue	out	Object ref	Java.io.serializable value. (Usually a string).	

Notes on Getting and Setting the PROTOCOL_HEADERS and SOAP_HEADERS Variables

All context variable values are typed as either "string" or "int" except for the predefined context variables PROTOCOL_HEADERS and SOAP_HEADERS, which are of the type "IData". You can set/get values for PROTOCOL_HEADERS and SOAP_HEADERS in one of two ways:

- **Set/get the entire structure.**

  To set the entire structure, you must:

- Set the `varName` parameter in `pub.mediator.ctxvar:setContextVariable` to `PROTOCOL_HEADERS` or `SOAP_HEADERS`.
- Use the method `ISMediatorRuntimeFacade.setContextVariableValue()`.

To get the entire structure, you must:

- Set the `varName` parameter in `pub.mediator.ctxvar:getContextVariable` to `PROTOCOL_HEADERS` or `SOAP_HEADERS`.
- Use the method `ISMediatorRuntimeFacade.getContextVariableValue()`.

If varName is set to PROTOCOL_HEADERS, you will get/set the entire IData structure containing all of the transport headers. The key is the transport header name (e.g., Content-Type) and the value is a String. The IData object for PROTOCOL_HEADERS will contain a set of string values where each IData string key matches the header name in the transport headers map. The set of possible keys includes the HTTP v1.1 set of headers as well as any custom key-value pairs you might have defined.

If varName is set to SOAP_HEADERS, you will get/set the entire IData structure containing all of the SOAP headers in the SOAP envelope. The key is the array position starting with '0', and the value is an Axiom OMElement containing that SOAP header block.

Alternatively, you can set the varName parameter to address a specific element in the array. For example, setting it to PROTOCOL_HEADERS[Content-Type] would apply to the Content-Type transport header. Similarly, setting it to SOAP_HEADERS[0] would return a String representation of the first SOAP header block (as opposed to an Axiom OMElement).

- **Set/get a nested value.**

  Set a nested value in one of the following ways:

  - Set the `varName` parameter in `pub.mediator.ctxvar:setContextVariable` to `PROTOCOL_HEADERS[`*`arrayElement`*`]`, where `[`*`arrayElement`*`]` refers to a specific element. For example, `PROTOCOL_HEADERS[Content-Type]` or `SOAP_HEADERS[0]` (to indicate the first array element in the set).
  - Alternatively, use the method `ISMediatorRuntimeFacade.setContextVariableValue()`. You would use this method only if you are writing a Java service and you want to access it through the Java source code.

  Get a nested value in one of the following ways:

  - Set the `varName` parameter in `pub.mediator.ctxvar:getContextVariable` to `PROTOCOL_HEADERS[`*`arrayElement`*`]`, where `[`*`arrayElement`*`]` refers to a specific element. For example, `PROTOCOL_HEADERS[Content-Type]` or `SOAP_HEADERS[0]` (to indicate the first array element in the set).
  - Alternatively, use the method `ISMediatorRuntimeFacade.getContextVariableValue()`. You would use this method only if you are writing a Java service and you want to access it through the Java source code.

You can set/get a nested value inside PROTOCOL_HEADERS and SOAP_HEADERS via an additional keyName. In this case, the object reference will *not* be an IData object.

- For PROTOCOL_HEADERS, the keyName must match the transport header name in a case-sensitive manner (e.g., `PROTOCOL_HEADERS[Content-Type]` or `PROTOCOL_HEADERS[Authorization]`). In this case, the Serializable value will be a string.
- For SOAP_HEADERS, the keyName must match the 0-based array element. If a request has a SOAP security header element (i.e. </wsse:Security>), then it would be addressed as SOAP_HEADERS[0]. In this case, the element will be in its string format.

pub.mediator.ctxvar:setContextVariable

Use this Java service to set a value on a context variable. The pipeline variable containing the context variable value should be an object reference that implements java.io.Serializable. All parameter names are case-sensitive.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by Mediator.	N/A

varName	in	String	Context variable name (predefined or custom).	`PROTOCOL_HEADERS`
				`SOAP_HEADERS`
				`mx:CUSTOM_VAR`

serValue	in	Object ref	Java.io.serializable value. (Usually a string).	

pub.mediator.ctxvar:declareContextVariable

Use this Java service to declare a *custom* context variable. All custom-defined context variables must be declared in a custom namespace that is identified by using the prefix `mx` (e.g., `mx:CUSTOM_VARIABLE`). All parameter names are case-sensitive.

Note: It is not legal to use this service to declare the predefined context variables; you can only declare custom variables.

Parameter	Pipeline Type	Data Type	Description
ctxVar	in	Object ref	The document type defining the context variable object. Use the ctxVar Document Type provided in the Java service pub.mediator.ctxvar:ctxVar and map it to this input variable. Define the name (name of the context variable mx:CUSTOM_VARIABLE), schema_type: string or int, isReadOnly: true or false.
ctxVar	out	Object ref	The set Context variable document type.
varNameQ	out	Object ref	javax.xml.namespace.QName value. The QName of the variable.

Note the following:

- After declaring the context variable, you can use the setContext variable to set a value on the context variable.
- You do *not* need to declare the following kinds of context variables:
  - The **predefined context variables** provide by Mediator. If you attempt to declare an existing predefined context variable, an error will occur.
  - Any custom context variable that you define in a routing rule that you create in the **Context-Based Routing** step.
- Any custom context variables that you explicitly declare in source code using the API will have a declaration scope of SESSION.
- Any custom context variable's state that is defined during the inbound request processing steps will still be available during the outbound response processing steps.
- All context variable values are typed as either "string" or "int" (excluding the SOAP_HEADERS and PROTOCOL_HEADERS variables, which are of the type "IData").
- Valid names should be upper case (by convention) and must be a valid Java Identifier. In general, use alpha-numerics, $ or _ symbols to construct these context names. Names with punctuation, whitespace or other characters will be considered invalid and will fail deployment.
- All custom context variables must be declared in a custom namespace that is identified by using an `mx` prefix (e.g. `mx:CUSTOM_VARIABLE`).
- To reference a custom context variable in a flat string, you need to prepend a $ symbol to the context variable name to indicate that variable's value should be referenced. Think of this usage as being similar to the '&' address operation for C variables.

  An expression that references a custom context variable might look like this:

  `$mx:TAXID=1234` or `$mx:ORDER_SYSTEM_NAME="Pluto"`

  Notice that the values of the data type "int" are not enclosed in quotation marks, while the values of the data type "string" are. The quotation marks are only needed if a context variable *expression* (as opposed to a reference) is defined.
- Referencing an undefined context variable does not result in an error.
- Once a variable has been declared it cannot be declared again.

pub.mediator.ctxvar:removeContextVariable

Use this Java service to remove a *custom* context variable from a request/response session. All parameter names are case-sensitive.

Notes:

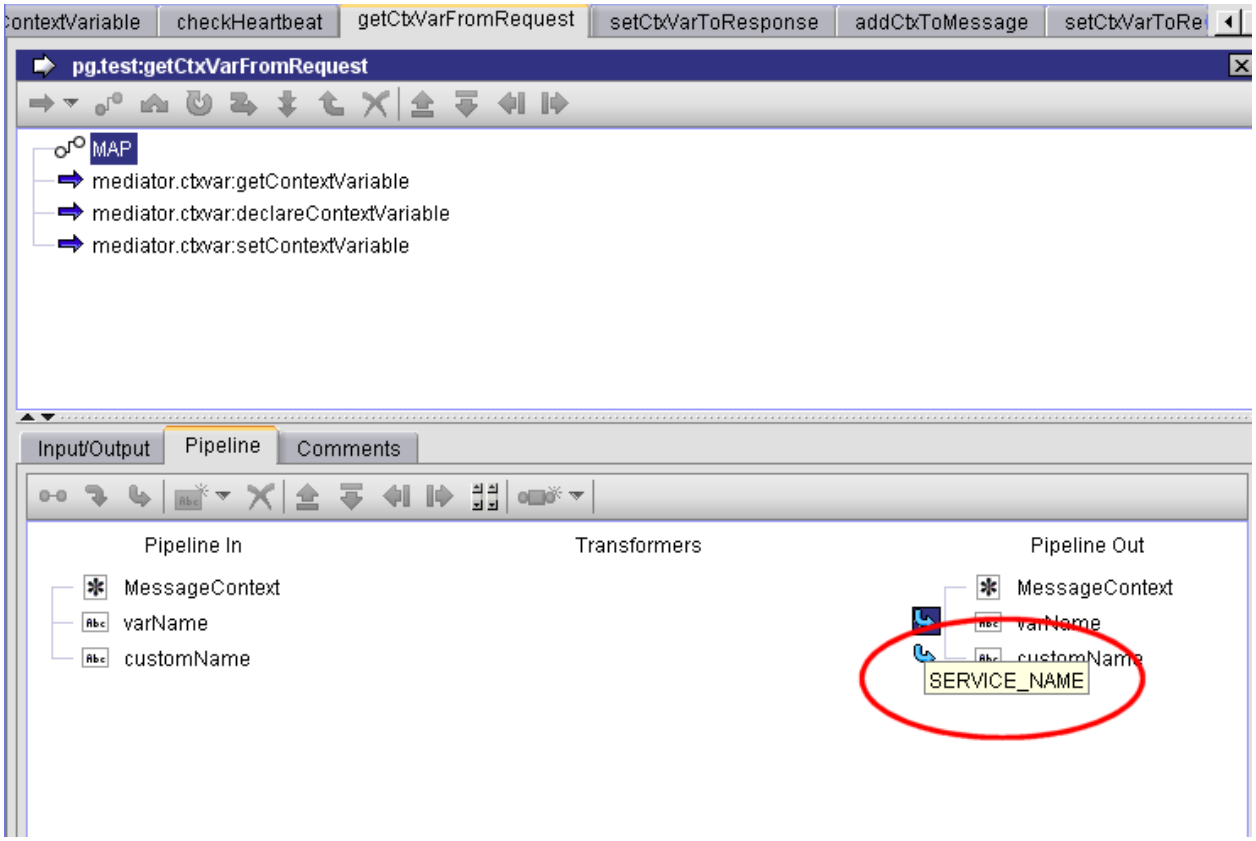
1. It is not legal to use this service to remove any predefined context variables; you can only remove custom variables.
2. Attempting to remove a non-existent context variable will *not* result in an error.

Parameter	Pipeline Type	Data Type	Description	Examples
MessageContext	in	Object ref	This object is inserted into the pipeline by Mediator.	N/A
varName	in	String	Custom context variable name.	`mx:CUSTOM_VAR`

Sample Flow Service: Getting a Context Variable Value

This flow service gets the value of a custom context variable from a request. The service declares a context variable using the name defined in the pipeline variable `customName` (i.e., `mx:COMP_TEST`). It is hard-coded to store the predefined context variable `SERVICE_NAME` in this custom context variable name.

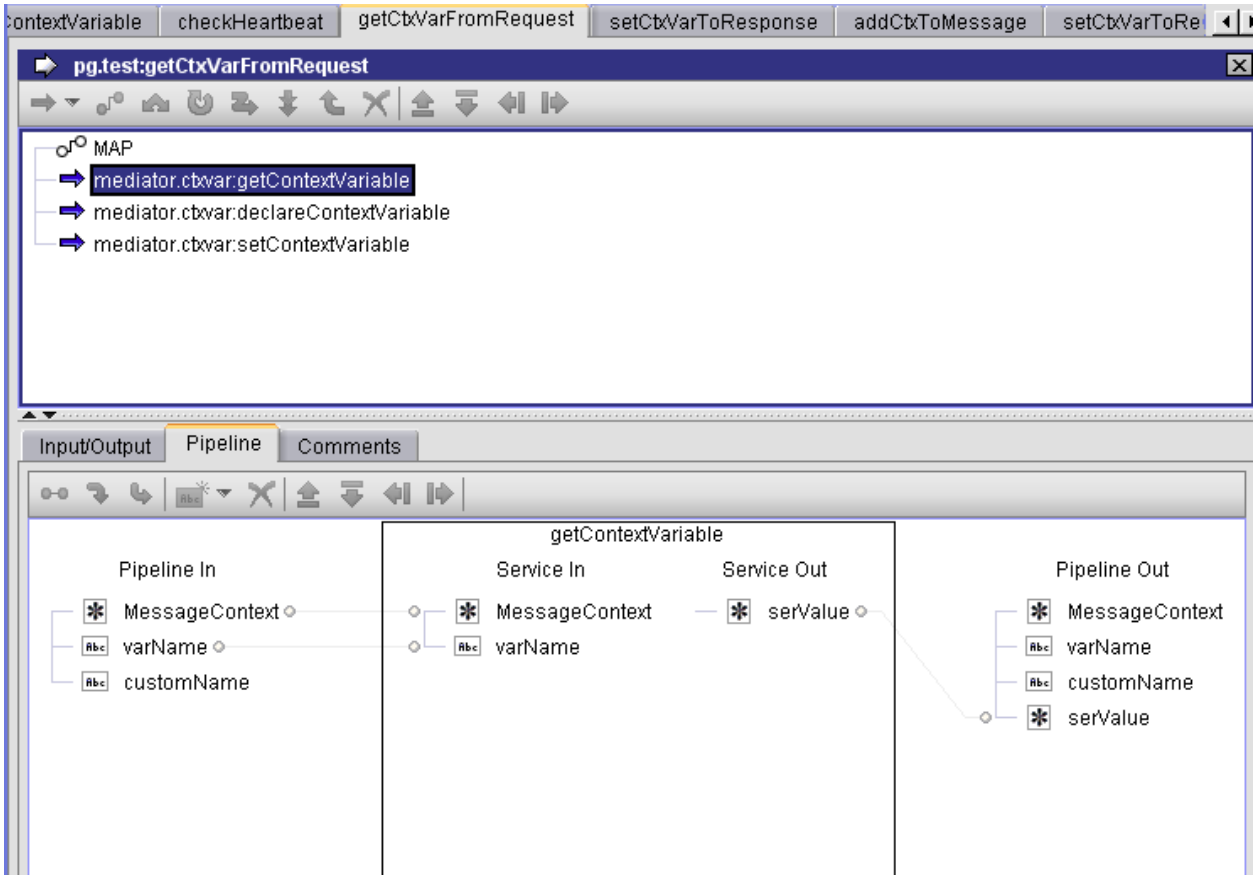
Step 1. Setting varName and customName



In this flow service, the following pipeline variables that are hard-coded as follows:

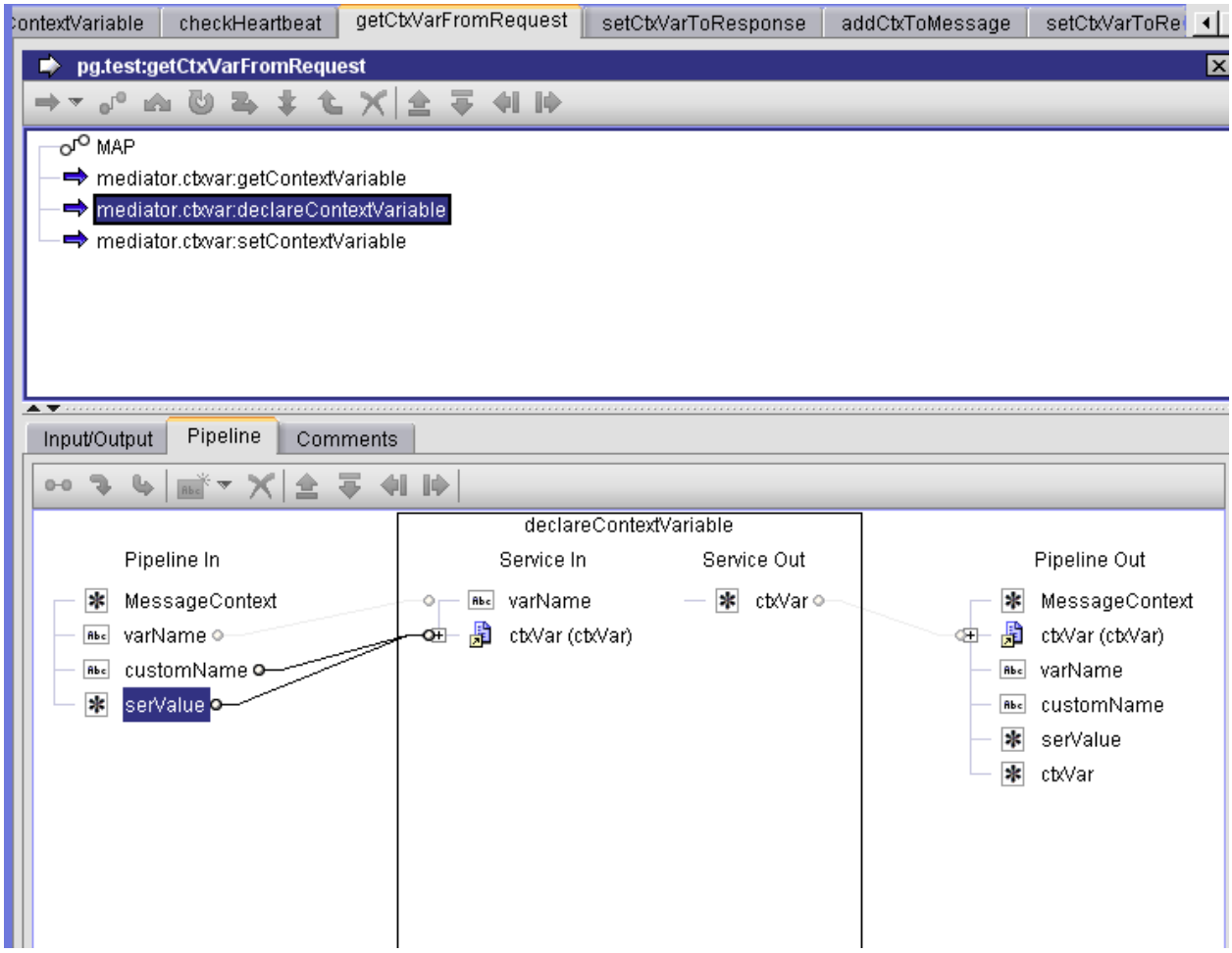
- `varName` is set to the predefined context variable `SERVICE_NAME`.
- `customName` is set to `mx:COMP_TEST. SERVICE_NAME` is stored in `customName` in the "Pipeline Out".

Step 2. Calling the pub.mediator.ctxvar:getContextVariable service



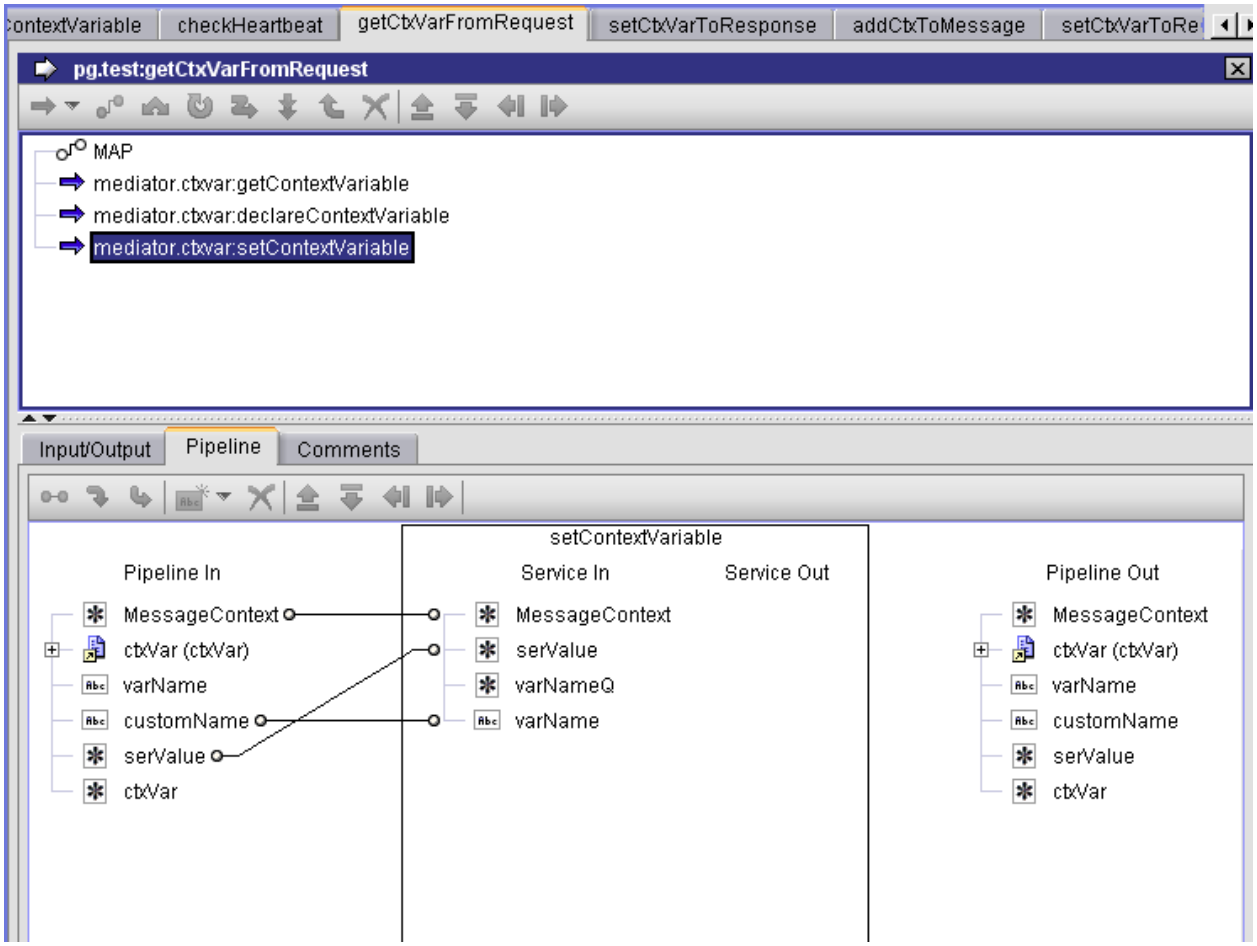
The call to `pub.mediator.ctxvar:getContextVariable` looks up the context variable value for the "Pipeline In" variable `varName` and sets the `Serializable` value in the output pipeline variable named `serValue`.

Step 3. Calling the pub.mediator.ctxvar:declareContextVariable service



The call to `pub.mediator.ctxvar:declareContextVariable` takes the serialized value we looked up for `varName` and assigns it to a newly declared custom context variable named `ctxVar`.

Step 4. Calling the pub.mediator.ctxvar:setContextVariable service



The call to `pub.mediator.ctxvar:setContextVariable` sets the context variable value back into `MessageContext` so it will be available for the rest of the service invocation steps.

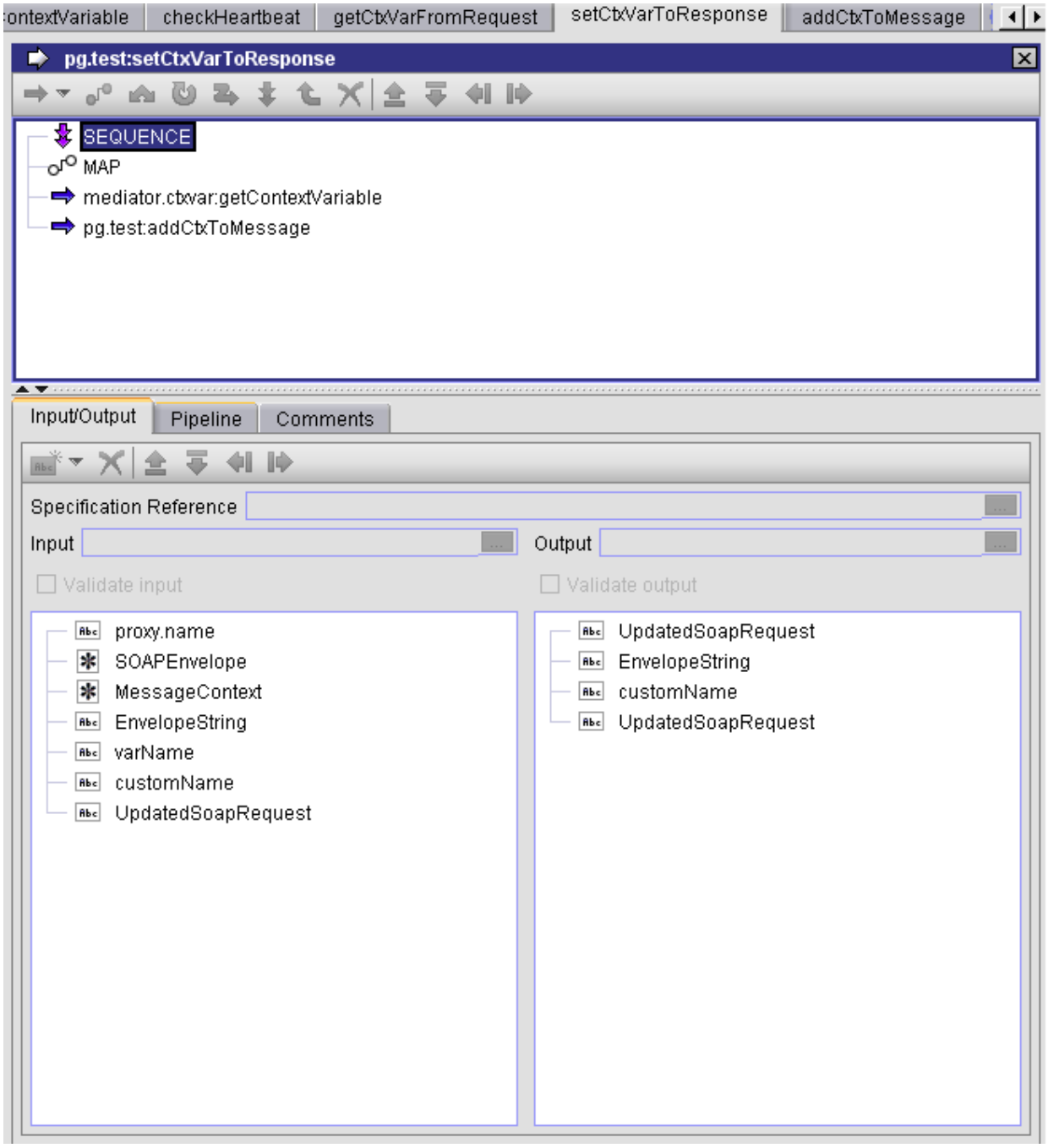
Sample Flow Service: Setting a Context Variable Value

This flow service sets the value of a custom context variable to be used in a response.

This flow service declares a pipeline variable named `customName`, which is set to the value `mx:COMP_TEST`.

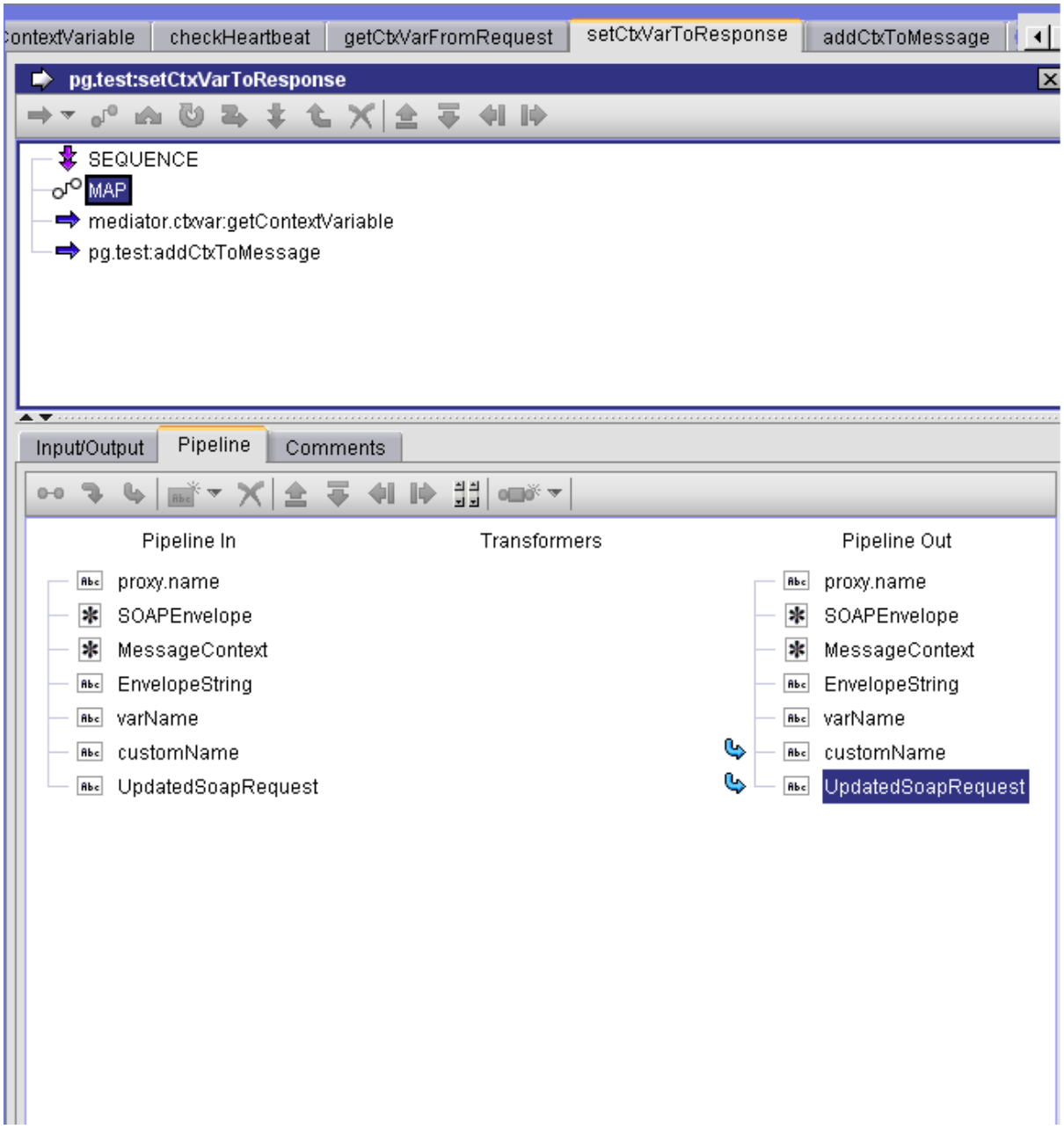
This flow service will retrieve the context variable for `customName` and create an element for its context variable value in the response message return to the consumer.

Step 1. Declaring customName



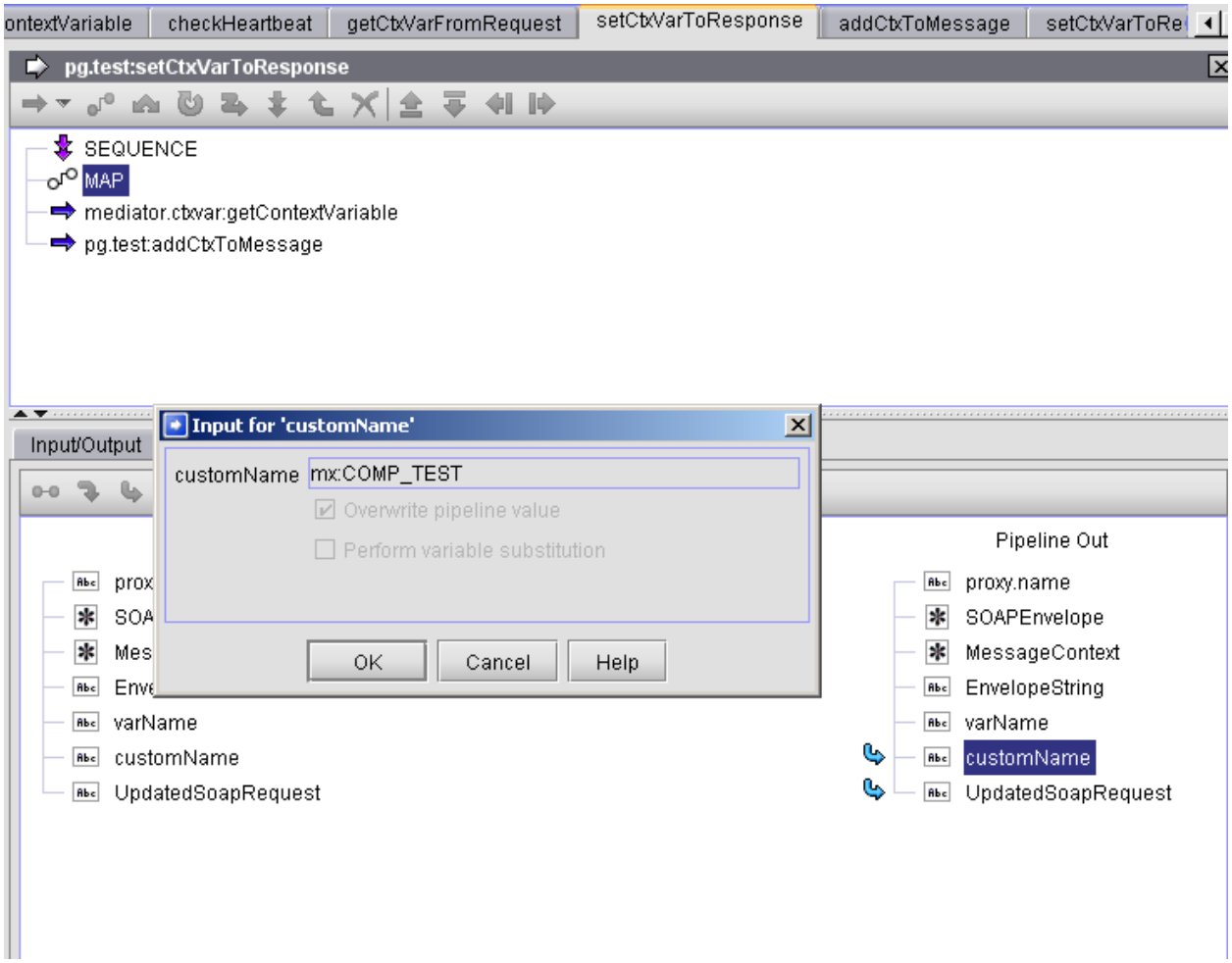
We define the `customName` variable value to be `mx:COMP_TEST` so we can use this variable to lookup the custom variable name that was seeded in the previous example.

Step 2. Setting customName to mx:COMP_TEST



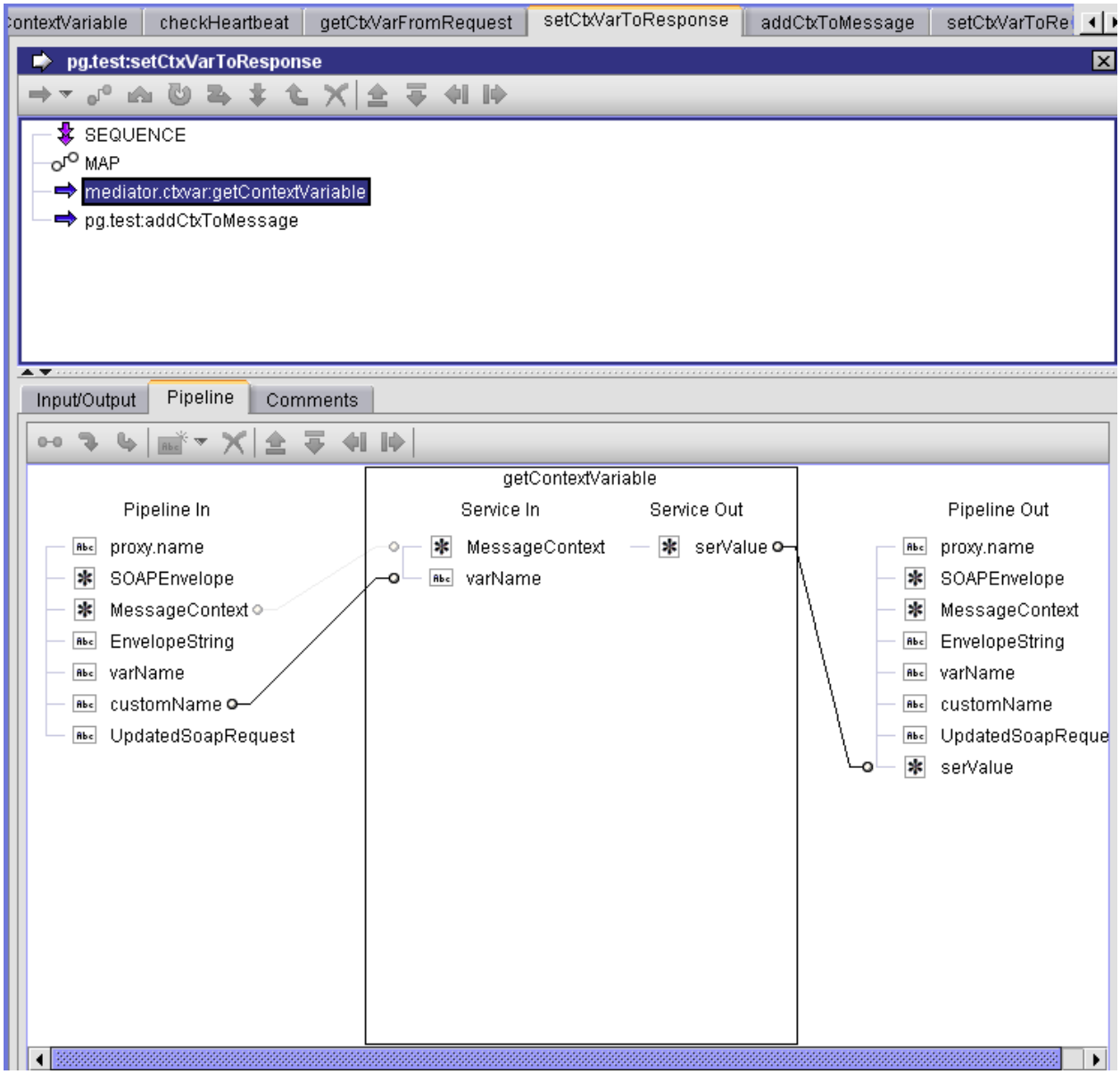
Clicking on the `customName` pipeline variable will display the name.

Step 3. Displaying the value of customName



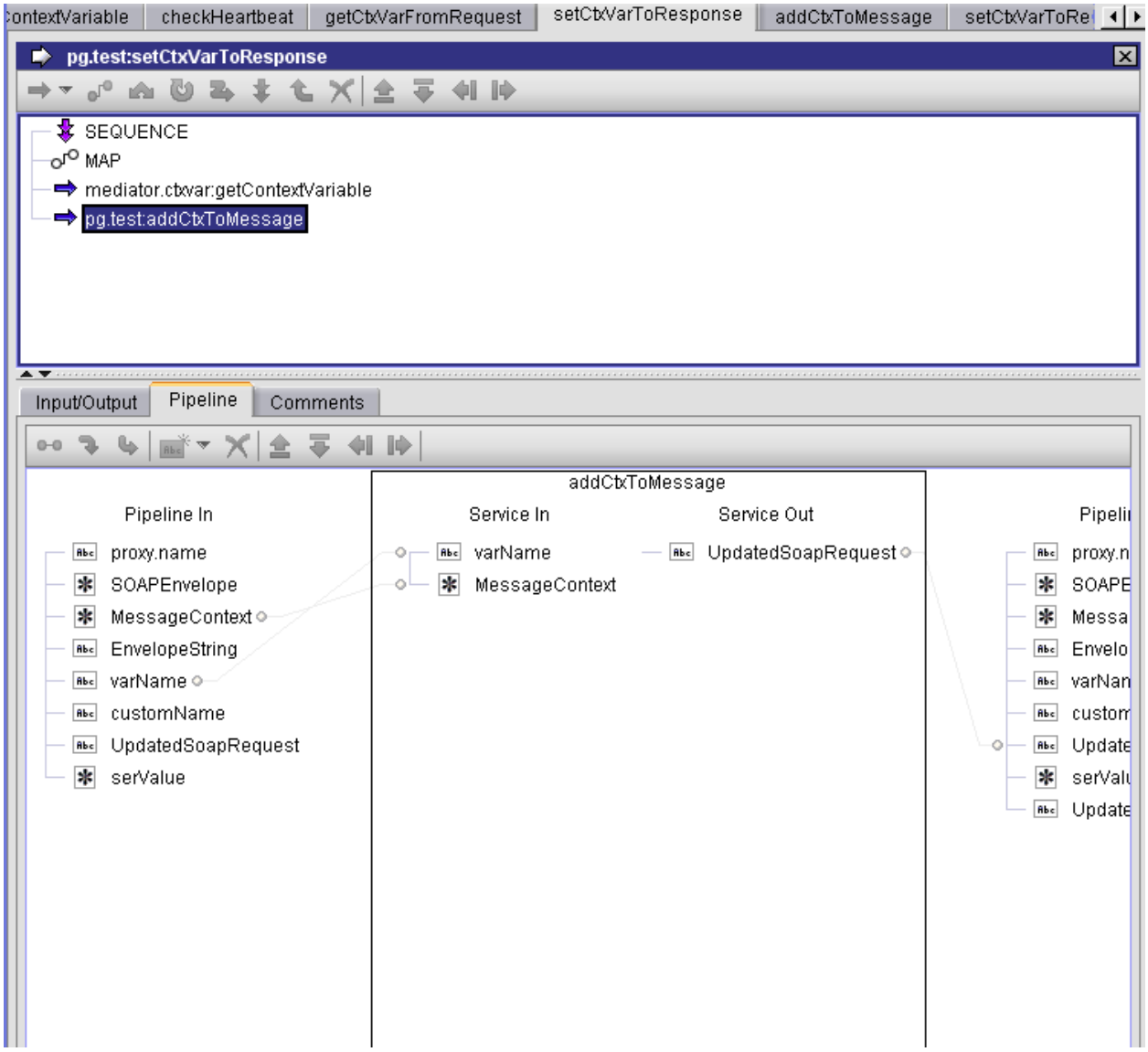
The call to `pub.mediator.ctxvar:getContextVariable` retrieves the value of the custom context variable from the context variable map.

Step 4. Calling meditor.ctxvar:getContextVariable



This is just a sample Java service that takes the context variable and creates a top-level element in the response message using the same name and value.

Step 5. Sample service using the context variable


contextVariable
checkHeartbeat
getCtxVarFromRequest
setCtxVarToResponse
addCtxToMessage
pg.test:setCtxVarToResponse
SEQUENCE
MAP
mediator.ctxvar:getContextVariable
pg.test:addCtxToMessage
Input/Output
Pipeline
Comments
addCtxToMessage
Pipeline In
Service In
Service Out
proxy.name
SOAPEnvelope
MessageContext
EnvelopeString
varName
customName
UpdatedSoapRequest
serValue
varName
MessageContext
UpdatedSoapRequest

8 Viewing or Editing the Profiles of Virtualized Services

Each tab on the detail page of a virtualized service represents a collection of attributes called a profile. You will only see the profiles for which you have View permission. The asset type Service has a unique set of profiles. However, your administrator can configure this asset type to display a customized set of profiles and attributes.

When editing attributes, keep the following points in mind:

- If you are not the owner of the virtualized service, you cannot edit the virtualized service unless you have Modify permission on the virtualized service (granted through either a role-based permission or an instance-level permission).
- When you view the details for the virtualized service, you will only see profiles for which you have View permission. You will only be able to edit the profiles on which you have Modify permission.
- Some attributes accept only specific types of information. For example, if the virtualized service includes a URL type attribute, you must supply a URL when you edit that attribute. Other attribute types that require a specific type of value include Date attributes and Email attributes.
- Some attributes are designed to be read-only and cannot be edited even if they appear in a virtualized service on which you have Modify permission.

To view or edit the detail pages of a virtualized service

1 In CentraSite Control, go to **Asset Catalog > Browse Catalog**.

2 On the **Browse Catalog** page, perform a keyword search or advanced search to display the virtualized services. For procedures, see the section *Using the Asset Catalog*.

3 Locate the virtualized service whose details you want to view or edit and, from its context menu, select **Details**.

If you want to edit multiple services, mark the checkboxes of all desired services, and from the **Actions** menu, click **Details**. The detail page for each of the selected services is now displayed.

4 To edit an asset's **Name**, **Description** or user-defined version number, make sure the service is undeployed, then place the cursor in the appropriate field and modify the text as required.

5 To modify the extended attributes associated with the virtualized service, do the following:

1. Select the profile that contains the attribute(s) that you want to modify.
2. Edit the attributes on the profile as necessary.

Editing an Asset File

Certain assets contain one or more associated files. For example, a SOAP-based virtual service includes a WSDL file, while a REST or XML virtual service includes a schema file. You can upload a new file, or update an existing file, for virtual services.

For a SOAP-based Virtual Service only: You can attach a WSDL file to the catalog entry using the **Attach WSDL** command in the virtual service's **Actions** menu. If you are attaching a WSDL file to a service that already has a WSDL, the service name in the new WSDL must be identical to the service name in the existing one, or the process will fail.

For a Virtual REST Service or Virtual XML Service only: You can attach a schema file to the catalog entry using the **Add** (resource) button in the **Technical Details** profile of the virtual service. For more information about attaching a schema file to virtual REST services or virtual XML services, see ***Resourcing a Virtual REST or XML Service***

Editing a Service Endpoint

Assets such as REST virtual services and webMethods IS REST services can contain one or more service endpoints. You can specify multiple endpoints or modify existing endpoints as required.

3. Repeat steps 5.a and 5.b for each profile that you want to edit.

Note: If at any time you want to abandon your unsaved edits, click **Close**. CentraSite will ask you if you want to save your edits. Click **No** to abandon your edits and return the virtualized service's attributes to their previous settings.

6 When you have finished making your edits, click **Save**.

You can view a tooltip text for any attribute in a profile of the virtualized service's detail page by moving the cursor to the attribute name. The tooltip text gives a summary of the attribute's purpose. The tooltip text shown is the content of the attribute's `Description` field, as defined for the virtualized service in the Service type definition. See the section *Object Type Management* for information on defining attributes for Service type.

The following profiles are described below:

The Summary Profile

The virtualized service's **Summary** profile displays general information about the service, including:

For this service type...	The profile displays...
Virtual Service	The WSDL file and the native service's endpoint(s).
Virtual REST Service	The schema file, the native service's endpoint(s) and the HTTP methods (GET, POST, PUT, DELETE or Use Context Variable) that the native service accepts in requests.
Virtual XML Service	The schema file and the native service's endpoint(s).

The Technical Details Profile

The **Technical Details** profile displays the following:

For this virtualized service type...	The profile displays...
Web service	■ The WSDL file of the Web service. ■ The native service's endpoint.
REST service or XML service	■ The schema file of the REST service or XML service. ■ The native service endpoint. *For Virtual REST services only:* If you want to specify multiple endpoints, use the plus button to add additional rows. ■ The MIME type of the data supported by the service. For a REST service, this is often application/xml or application/json type but can be any other valid MIME type. For an XML service, this is only application/xml type. ■ The set of operations supported by the service using HTTP methods (e.g., POST, GET, PUT or DELETE). ■ The search string used by the server to find a set of matching resources.

Resourcing a Virtual REST or XML Service

General

A virtual REST service or a virtual XML service has the "Resource" object internally representing an "Operation" object in the CentraSite registry. When adding or importing a virtual REST service or virtual XML service, CentraSite, by means of an *invoke* operation, automatically sets the **Resource:invoke** entry in the **Technical Details** profile of the new service. Based upon the type of asset (i.e., virtual XML Service or virtual REST Service), it populates the basic attributes, namely **HTTP Method**, **Content Type**, **Query String** and **Schema Name** fetched from the service's schema file. You can add or modify these resource attributes as required.

Important Considerations When Resourcing a Virtual REST or XML Service

When resourcing a virtual REST service or a virtual XML service, keep the following points in mind:

- A virtual XML service can contain only one resource object. However, if you want to specify a new resource or modify an existing resource object, you can delete the existing resource and add the new resource details as required.
- Remember that CentraSite adds the resource object that is fetched from the original REST service as the **Top-level Resource** for a virtual REST service. A *Top-level Resource* is one whose attribute values are used for the actual deployment process in Mediator.
- For virtual REST services only: If the chosen content type is **application/xml**, specifying the HTTP method and schema is *mandatory*. However, if the chosen content type is **application/json**, specifying the HTTP method and schema is *optional*.

How to Resource a Virtual REST or XML Service

To resource a virtual REST service or a virtual XML service

1 In CentraSite Control, display the details page for the virtual REST service or virtual XML service to which you want to add a resource. If you need procedures for this step, see the section *Using the Asset Catalog*.

2 Choose the **Technical Details** profile, which allows you to associate a resource object to the service.

3 Click the **Add** button.

The **Add Resource** dialog is displayed.

4 In panel 1, specify the following attributes:

In this field...	Do the following...
Resource	*Mandatory*. Enter a name for the resource. A resource name can contain any character (including spaces).
HTTP Method	*Mandatory for application/xml type*. Choose the HTTP request method(s) for bridging protocols (e.g., GET, POST, PUT, DELETE).
Content Type	Specify the MIME type of the data supported by the service. **Note:** For a REST service, this is often the application/xml or application/json type, but it can be any other valid MIME type. For an XML service, only the application/xml type is valid.
Top-level Resource	*For REST services only:* Use this checkbox to choose the resource to expose to Mediator for the deployment process.

In this field...	Do the following...
Query String	*For REST services and virtual REST services only:* Specify the search string as required. The text is case insensitive.

5 If you have chosen the content type application/xml, click **Next**.

-OR-

If you have chosen the content type application/json, click **Finish**.

6 In panel 2, specify the following attributes to associate the schema(s) with this service.

In this field...	Do the following...
Import from	Specify whether the input file will be read from a URL-addressable location on the network (the **URL** option) or from your local file system (the **File** option).
URL or **File**	If the file you are importing resides on the network, specify its URL. If the file resides in your local file system, specify the file name. You can use the **Browse** button to navigate to the required folder. If you want to specify multiple schemas, use the plus button to add additional rows.
URL Authentication	If you have specified a URL, and the site you want to access via the URL requires user authentication, check this box. This opens an **Authentication** sub-dialog in which you can enter a username and password for authentication at the URL site.
Overwrite existing imported schema files	This option determines how referenced schemas are handled when the schema that is referenced already exists in the registry. When this option is enabled, the existing schemas in the registry are replaced with the new ones specified by the input schema file(s). When this option is disabled, the importer simply references the schema that already exists in the registry.

7 Click **Finish**.

Deleting a Resource from a Virtual REST Service or XML Service

Use the following procedure to delete a resource from a virtual REST service or a virtual XML service.

To delete a resource from the virtual REST service or virtual XML service

1 In CentraSite Control, display the details page for the virtual REST service or virtual XML service whose resource you want to delete. If you need procedures for this step, see the section *Using the Asset Catalog*.

2 On the **Technical Details** profile, select the resource(s) that you want to delete.

3 Click **Delete**.

The Specification Profile

You use the virtualized service's **Specification** profile to view or modify the following fields.

Field	Description
Functional Requirements	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library. For details about the Supporting Document Library, see the section *Using the Asset Catalog*.
Non-functional Requirements	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Error Messages and Codes	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Usage Examples	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Release Notes	You can attach a document by either pointing to a URL, or selecting a document from the Supporting Document Library. You can also upload a new document to the library.
Demo WSDL URL	You can view or modify the Demo WSDL URL.
Documentation URL	You can view or modify the Documentation URL.
Consumer WSDL	If the built-in policy **Consumer WSDL Generator** has been activated, this field will display a *Consumer WSDL*. For information about this policy, see the section *Built-In Design/Change-Time Reference* . For information about activating policies, see the section *Working with Design/Change-Time Policies*.) A Consumer WSDL is created in addition to the virtual service's WSDL. The virtual service's WSDL will be used by Mediator. The Consumer WSDL can be used by the consumer (the user) to create a request for the service. The Consumer WSDL will contain WS-Security policies inline in the consumer WSDL as follows: ■ If the virtual service contains WS-Security policies, then the WS-Security policies are included inline in the Consumer WSDL. Any WS-Security policies contained in the native service's WSDL are removed. ■ If the virtual service does *not* contain WS-Security policies, then the WS-Security policies contained in the native service's WSDL (if any) are included inline in the Consumer WSDL.

The Consumers Profile

You use the virtualized service's **Consumers** profile to specify the consumer applications, users and groups that are allowed to request the service. Specify the following fields and click **Save**.

In this field...	Specify...
Application	One or more application assets, which represent the consumer applications that are allowed to request the service. Click **Add Applications** to select from the list of application assets that are registered in CentraSite. For more information, see the section *Using the Asset Catalog*. To remove an application asset from this list, select the check box next to its name and click **Delete**.
Users/Groups	One or more users and/or groups that are allowed to request the service. Click **Add Users/Groups** to select from the list of users and groups that are registered in CentraSite. To remove a user or group from this list, select the check box next to its name and click **Delete**.

The Permissions Profile

You use the virtualized service's **Permissions** profile to set permissions for the service. For information, see the section *Users, Groups, Roles, and Permissions* .

Important: To set permissions on a virtualized service, you must have the Full instance-level permission on the native service from which the service was generated (or you must belong to a role that has the "Manage Assets" permission).

The Policies Profile

The virtualized service's **Policies** profile displays a list of all design-time policies and run-time policies that apply to the service. To view a policy's detail page, click the hyperlinked policy name.

The Deployment Profile

For instructions on deploying, undeploying and redeploying virtualized services, see *Deploying Virtualized Services and Consumer Applications* .

The Performance Profile

The virtualized service's **Performance** profile displays the Key Performance Indicator (KPI) metrics that have been published for the service. You can filter the list by target and time period.

Notes:

1. Ensure that Mediator is configured to collect and report run-time events to CentraSite, as described in the section *Configuring Mediator* in the document *Administering webMethods Mediator*.
2. Ensure that CentraSite is configured to receive run-time events from Mediator, as described in the section *Run-Time Governance Reference > Run-Time Events and Key Performance Indicator (KPI) Metrics*.

Note: If you receive a Javascript error when trying to display the **Performance** profile, please install the latest versions of the Adobe Flash Player/Shockwave Player plug-ins on your Microsoft Internet Explorer. The plug-ins can be obtained from **http://www.adobe.com/shockwave/welcome/**.

To view KPI metrics

1 In CentraSite Control, display the virtualized service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Open the **Performance** profile.

3 Use the **Switch to** button to switch between a tabular view of the metrics or a graphical view.

4 When viewing metrics in Tabular View, specify the following fields:

In this field...	Specify...
Select Target	A target to which the virtualized service is deployed, or select **All** to view the metrics of all targets to which the virtualized service is deployed.
Start Date/End Date	The time period from which to view the metrics.

5 Click **Search**.

The table displays metrics for all performance categories (Success Request Count, Total Request Count, Fault Count, etc.).

Note: When viewing metrics in Graphical View, you view one performance category at a time. Select the category from the **Metrics** field drop-down list.

The Events Profile

The virtualized service's **Events** profile displays the run-time events for the service. You can filter the list by target, event type and time period.

Notes:

1. Ensure that Mediator is configured to collect and report run-time events to CentraSite, as described in the section *Configuring Mediator* in the document *Administering webMethods Mediator*.
2. Ensure that CentraSite is configured to receive run-time events from Mediator, as described in the section *Run-Time Governance Reference > Run-Time Events and Key Performance Indicator (KPI) Metrics* .

To view run-time events for a virtualized service

1 In CentraSite Control, display the virtualized service's detail page. For procedures, see the section *Using the Asset Catalog*.

2 Filter the event list you want to generate as follows:

In this field...	Specify...
Target Type	The type of the target on which the event occurred.
Target	The target on which the event occurred, or select **All Targets**.
Event Type	A run-time event type, or select **All Events**.
Date Range	A range of dates from which to view the events. If you select a value for Date Range, then Start Date/Time and End Date/Time are ignored.
Start Date	Click the calendar to specify a start date and time.
End Date	Click the calendar to specify an end date and time.

3 Click **Search**.

4 The generated event list displays the following information:

Column	Description
Date/Time	The date/time of the event. Click this hyperlinked value to view the payload of the request/response.
Session ID	The SOAP invocation session ID of the event.
Event Type	The type of run-time event.
Service Name	The name of the service involved in the event.
Service Type	The service's type.
Target	The target on which the event occurred.
Target Type	The target's type.

To configure CentraSite to log run-time events, see the section *Logging* .

Note: To view lists of all events for all virtualized services of a particular target (or for all targets), see the section *Managing Targets and Run-Time Events* .

Revising Virtualized Services

For details, see *Managing Virtualized Services* in the *Implementation Concepts* section.

9 Important Considerations when Configuring SOAP-based Virtual Services

Following are some things you should consider when you configure SOAP-based virtual services.

Handling Services with Multiple Ports and Bindings

Mediator implicitly assumes that there is a one-to-one mapping between a WSDL service and a Virtual Service. A problem arises if a `<service>` element contains multiple `<port>` elements that point to different bindings (and consequently different port types) -- the problem is that Mediator will create a virtual service that has the operations from only *one* of the `portType`s. Mediator chooses the first port available under `<service>` and exposes the operations corresponding to the equivalent binding/portType.

For example, consider the following WSDL fragment that shows the structure of the `portType`, `binding` and `service` elements in the WSDL. Note that there are:

- Two distinct `<portType>` elements: `SystemPortType` and `CustomerPortType`.
- Two equivalent bindings defined for each `<portType>`: `SystemBinding` and `CustomerBinding`.
- A single `<service>` element that defines the two ports with distinct endpoints (one for each binding available).

Example <portType> Elements

```
<portType name="SystemPortType">
    <operation name="ping">
        …
    </operation>
</portType>

<portType name="CustomerPortType">
    <operation name="getOperation1">
        …
    </operation>
    <operation name="getOperation2">
        …
    </operation>
    <operation name="getOperation3">
        …
    </operation>
    <operation name="getOperation4">
        …
    </operation>
</portType>
```

Example <binding> Elements

```
<binding name="SystemBinding" type="tns:SystemPortType">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="ping">
        …
    </operation>
</binding>

<binding name="CustomerBinding" type="tns:CustomerPortType">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getOperation1">
        …
    </operation>
    <operation name="getOperation2">
        …
    </operation>
    <operation name="getOperation3">
        …
    </operation>
    <operation name="getOperation4">
        <soap:operation soapAction="urn:customer.service.cm.be/getOperation4"/>
        …
    </operation>
</binding>
```

Example <service> Element

```
<service name="CustomerRefService">
    <port name="SystemPort" binding="tns:SystemBinding">
        <soap:address location="http://.../v4/SystemPort"/>
    </port>
    <port name="CustomerPort" binding="tns:CustomerBinding">
        <soap:address location="http://.../v4/CustomerRefPort"/>
    </port>
</service>
```

There are two options for a workaround to this problem:

Workaround Option 1

You can create two different virtual services. That is, you can expose this native service as two virtual services -- one for each operation that needs to be invoked:

- A service for the `get`*`XXX`* operations (for example, a service called `VS_Customer`).
- A service for the `ping` operation (for example, a service called `VS_ping`).

To create a different virtual service for each operation

1 Create a virtual service for the native service `CustomerRefService` and name it `VS_Customer`, for example.

2 Configure `VS_Customer` and configure its routing protocol as "Straight Through".

3 Specify the routing address as `http://... /v4/CustomerRefPort` (for `CustomerBinding`, where all the `get`*`XXX`* operations are supported).

4 On the virtual service's **Summary** profile, click on the URL for the WSDL (this a copy of the virtual service template WSDL, very similar to the original native service WSDL) and download/save it to your local file system.

(Make an additional copy of this downloaded WSDL in case you make a mistake in your editing.)

5 Remove the following elements from the WSDL and then save it:

- `SystemPortType`
- `SystemBinding`
- `SystemPort`

Note: Make sure your browser or XML tool can read this modified WSDL without any syntax error.

6 Attach the modified WSDL file to the virtual service by selecting the **Attach WSDL** command in the virtual service's **Actions** menu.

7 Create another virtual service for the native service `CustomerRefService` and name it `VS_ping`, for example. Repeat the above steps but with the following differences:

- Specify the routing address as `http://... /v4/SystemPort`.
- Remove the following elements from the WSDL:
  - `CustomerPortType`
  - `CustomerBinding`
  - `CustomerPort`

8 Deploy both virtual services to Mediator.

Workaround Option 2

With this option, you expose the native service as one virtual service. The web service client will access the service through one address to the virtual service for all the possible operations (`ping` and `get`*`XXX`*). The virtual service then takes care of routing to the correct endpoint for the different operations. This is accomplished by using "Content-based" routing (instead of "Straight Through" routing) to determine the operation being called (based on the SOAP request content) and then forwarding the request to the correct endpoint.

To create a virtual service with "Content-based" routing

1 Create a virtual service for the native service `CustomerRefService` and name it `VS_Customer-RefService`, for example.

2 Configure `VS_CustomerRefService` and configure its routing protocol as "Content-based".

3 On the **Routing Protocols** tab, construct the routing rule as follows:

1. Click the **Endpoint** button (next to the **Route To** column).
2. In the **Search for Endpoint** dialog that appears, click the **Search** button to search for the Web service endpoint to route the requests to.
3. Select `http://... /v4/SystemPort` (the Accessing URI that goes to ping operation).
4. To create an XPath expression for the routing rule, click the **Edit** button (next to the **If True** column).
5. In the XPath Editor that appears, click the **All Nodes** tab, expand the namespace's node, click to highlight the `tns:ping` element, and click **OK**.
6. Double check that you have something like this in the rule and it is routed to `SystemPort`:

```
/soapenv:Envelope/soapenv:Body/tns:pingRequest
```

4 Set the **Default To** routing field to the routing address `http://... /v4/CustomerRefPort` (for `CustomerBinding`, where `get`*`XXX`* operations are supported).

5 On the virtual service's **Summary** profile, click on the URL for the WSDL (this a copy of the virtual service template WSDL, very similar to the original native service WSDL) and download/save it to your local file system.

(Make an additional copy of this downloaded WSDL in case you make a mistake in your editing.)

6 Modify the WSDL as follows and then save it:

- Copy the `ping` operation from `SystemPortType` and add into `CustomerPortType`.
- Delete the `SystemPortType`. The objective here is to make *one* port type only.

- Update the `SystemBinding` to also refer to `CustomerPortType`, since `SystemPortType` has been deleted.

  Note: The `soapAction` attribute must be specified for the `soap:operation` element to ensure that Mediator can resolve the operation being invoked for this service.

- Save the WSDL.

  Note: Make sure your browser or XML tool can read this modified WSDL without any syntax error.

7 Attach the modified WSDL file to the virtual service by selecting the **Attach WSDL** command in the virtual service's **Actions** menu.

8 Deploy the virtual service to Mediator.

10 Important Considerations when Configuring REST or XML Virtual Services

With Web services, the native service endpoint that is sent by CentraSite to Mediator inside a virtual service definition is a static element. Once the virtual service is successfully deployed to Mediator, the real endpoint is returned to CentraSite as part of the response message during deployment. At run time, when a SOAP request is received, that request is POSTed to the endpoint that is statically defined in the virtual service definition.

With REST services or XML services, however the endpoint is flexible. The REST services or XML services describe data as resources. The resources are accessible via logical endpoints that have application meaning to users. For example, a collection of textbooks might be defined as a resource with the following URL:

```
http://{host}:{port}/books
```

A specific book with an identifier of 1234 would be accessible with the following URL:

```
http://{host}:{port}/books/1234 ↩
```

Due to this difference and others, you should keep the following topics in mind when you configure a REST or XML virtual service:

Endpoint Manipulation of REST or XML Virtual Services

When you configure the processing steps of a virtual REST or XML service, you specify the native service name, an endpoint, and the HTTP method type(s) that are included in the message (POST, GET, PUT, DELETE). From this information, CentraSite will generate a virtual service definition that includes service and operation elements, as well as an endpoint and binding element pair for each HTTP method specified.

CentraSite will automatically generate an operation name to be included when the virtual service definition deployment message is sent to Mediator. This means that if you create a virtual service called VS1, and you specify a native endpoint, then the endpoint exposed by Mediator for calling the virtual service will be /ws/VS1/Invoke.

For example, assume the following endpoints are deployed.

Native service endpoint:	http://localhost:8080/services/mtc/member
Virtual service endpoint:	http://localhost:5555/ws/VS1/Invoke

Assume that the example virtual service is deployed with two HTTP method bindings: GET and POST. Both of these bindings have operation elements that include the same HTTP location attribute: member. To better illustrate the functionality, the examples below show a series of sample requests from a consumer including the requests' HTTP method and Content-Type. (At run time, REST message detection is dependent upon a consumer using the correct Content-Type when a request is sent.) Each example shows the expected endpoint that Mediator will send after it has rewritten the endpoint prior to native service invocation.

Example 1

For a GET, assume that:

The request Content-Type is: application-x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member

The application function is: The native service returns a collection of members with summary information.

Example 2

For a GET, assume that:

The request Content-Type is: application-x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/1234

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member/1234

The application function is: The native service returns summary data for a member with this key.

Example 3

For a GET, assume that:

The request Content-Type is: application-x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/1234?detail=true

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member/1234?detail=true

The application function is: Query parameters remain intact. Returns a response message with more member details.

Example 4

For a POST, assume that:

The request Content-Type is: application/xml or application/json

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/1234

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member/1234

The application function is: The request message provides the contents needed to create the member resource.

Example 5

For a GET, assume that:

The request Content-Type is: application-x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke/joe

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member/joe

The application function is: Fetches the member defined with the login joe. (Mediator contains no metadata in its service deployment to differentiate between the “login” vs. “key” GET requests.)

Example 6

For a GET, assume that:

The request Content-Type is: application-x-www-form-urlencoded

The endpoint received by Mediator is: http://localhost:5555/ws/VS1/Invoke?type=login&value=joe

The native service endpoint sent by Mediator is: http://localhost:8080/services/mtc/member?type=login&value=joe

The application function is: The native service might also support a static endpoint with constraints defined in query parameters. Mediator also supports this approach.

The Request Message's HTTP Methods and Content-Types for REST and XML Services

When you configure the **Entry Protocol step** of a virtual REST or XML service, it is important to specify all the HTTP methods that are supported for the service. For example, if the virtual service is deployed to Mediator and you selected only the GET method in the virtual service's Entry Step, then Mediator will only permit GET invocations. In this case, a POST request will be rejected with a return of statusCode 405 even if the native service happens to support POSTs.

It is important that the client's requests contain an HTTP Content-Type header. At run time, Mediator determines which message builder to use based on the message's HTTP method and its Content-Type. (The absence of the soapAction header will indicate to Mediator that the message is an XML message.)

The valid HTTP method/Content-Type combinations are as follows:

This method...	Can be included in a message of this Content-Type...
POST	application/xml application/json application-x-www-form-urlencoded or multipart/form-data
PUT	application/xml application/json application-x-www-form-urlencoded or multipart/form-data
GET	application-x-www-form-urlencoded
DELETE	application-x-www-form-urlencoded

Notes:

1. If Mediator receives a request sent with an HTTP method that is not specified in the virtual REST service or virtual XML service definition, it will return a 405 error.
2. If Mediator receives a request sent with a wrong Content-Type, it will return a 415 error. In addition, if the wrong Content-Type is used with a GET or DELETE, then the query parameters contained in the message (if any) will not be processed.

Changing the HTTP Method of a REST or XML Request

When configuring the routing step of a REST or XML virtual service, you specify whether to route the requests to the native service with the same HTTP method that is contained in the requests (GET, POST, PUT, DELETE), or whether to route the requests with a different HTTP method.

Typically you want to pass each request to the native service with the same HTTP method that is contained in the request. For example, if a request contains a GET method, you allow the GET method to be passed to the native service. However, there might be rare cases in which you want to change the HTTP method of a request to different HTTP method. For example, you might want to:

- Expose an XML service as a REST service.

  In this case, the service you create would be a Virtual XML service that exposes the HTTP methods GET, POST, PUT and DELETE, but the routing method would always be POST.
- Expose a REST service whose virtual REST service only exposes the POST method.

To change the HTTP method of a REST or XML request

- On the REST or XML virtual service's **Routing Protocols** tab, set the value of the **HTTP Method** field either statically (by explicitly setting the value to **GET**, **POST**, **PUT**, or **DELETE**) or dynamically (by setting the value to **Use Context Variable**).

  In order to use the **Use Context Variable** option to set the field dynamically, you must write a **webMethods IS service** that sets a value of GET, POST, PUT or DELETE for a predefined context variable named ROUTING_METHOD. You need to invoke this service in the virtual service's Request Processing step. For details, see *Changing HTTP Methods in Requests Dynamically using a Context Variable*.

Caution: Use this feature carefully, since changing HTTP methods to certain other HTTP methods could result in unintended results or errors.

For example, changing an inbound GET request to a DELETE request would be a serious mistake if the deletion was not intended and the native REST service actually deleted a resource when invoked with a DELETE method. Additionally, an incoming POST or PUT request cannot be translated into a GET or DELETE if the request has nested elements. For more information, see *The Implications of Changing HTTP Methods*.

This section discusses the following topics:

The Implications of Changing HTTP Methods

Note the following.

When changing this incoming HTTP method...	To...	Note that...
GET	POST	■ The Content-Type of the changed request is sent as application/xml or application/json, and the charset is UTF-8. ■ Depending on the structure of the native service, be aware that the native service might not be expecting the same payload structure that is being sent. In this case, you would need to transform the request message into the format required by the native service before Mediator sends the requests to the native service. For more information, see *Sample XSLT Transformation for GET-to-POST or GET-to-PUT*.
GET	PUT	Identical to GET-to-POST, except that Mediator changes the request's HTTP method from GET to PUT.
GET	DELETE	No comment.

POST	GET	■ Mediator will translate the POSTed request elements into query string parameters, in a root element.
		Note: An incoming POST or PUT request cannot be translated into a GET or DELETE if the request has nested elements. For example:
		`(this is correct)`
		`<person>`
		`  <lastName>Smith</lastName>`
		`</person>`
		
		`(this is incorrect)`
		`<person>`
		`  <name>`
		`  <last>Smith</last>`
		`  </name>`
		`</person>`
		■ If you want to send additional parameters as part of the request URL, you can transform this payload. To do this, you can use an XSLT file or a webMethods IS service call to add parameters before the request is sent to the native service. For more information, see *The Request Processing Step*.

POST	DELETE	Identical to POST-to-GET, except that Mediator changes the request's HTTP method from POST to DELETE.

When changing this incoming HTTP method...	To...	Note that...
POST	PUT	The Content-Type of the changed request is sent as application/xml or application/json, and the charset is UTF-8.
PUT	GET	Identical to POST-to-GET, except that Mediator changes the request's HTTP method from PUT to GET.
PUT	POST	The Content-Type of the changed request is sent as application/xml or application/json, and the charset is UTF-8.
PUT	DELETE	Identical to POST-to-DELETE, except that Mediator changes the request's HTTP method from PUT to DELETE.
DELETE	GET	No comment.
DELETE	POST	Identical to GET-to-POST, except that Mediator changes the request's HTTP method from DELETE to POST.
DELETE	PUT	Identical to GET-to-PUT, except that Mediator changes the request's HTTP method from DELETE to PUT.
GET, POST, PUT or DELETE	**Use Context Variable**	See *Changing HTTP Methods in Requests Dynamically using a Context Variable*.
GET or DELETE	POST or PUT	Note that the query parameters will be picked off the URL and stored as top-level elements when the message is sent to the native service. The query parameters are ignored on the endpoint URL and lost when we POST to the native provider (i.e. don't change the protocol method).

Changing HTTP Methods in Requests Dynamically using a Context Variable

Alternatively, instead of changing an HTTP method explicitly (statically) to PUT, POST, GET or DELETE, you can change the HTTP method to the value of a predefined context variable (ROUTING_METHOD) that dynamically resolves to a different HTTP method (PUT, POST, GET or DELETE, as appropriate).

To change the HTTP method dynamically, you create a **webMethods IS service** and invoke it in the virtual service's **Request Processing** step. This webMethods IS service should reference the predefined context variable ROUTING_METHOD (see *The Predefined Context Variables*). To set the value of ROUTING_METHOD, use the setContextVariableValue method, which is defined in the following class:

```
com/softwareag/mediator/api/MediatorRuntimeFacade.java
```

For example:

```
public static final void updateHttpMethod(IData pipeline)
   throws ServiceException {

String mcKey = “Message Context”;

 Object obj = IDataUtil.get(pipeline.getCursor(), mcKey);
 if (obj!=null && obj instanceof
org.apache.axis2.context.MessageContext) {

  MessageContext msgCtx = (MessageContext) obj;

QName varName =
new QName(MediatorContextVariableType.ROUTING_METHOD.getName());

MediatorRuntimeFacade.setContextVariableValue(varName, "PUT", msgCtx );
 }
}
```

Sample XSLT Transformation for GET-to-POST or GET-to-PUT

As stated in the above table, depending on the structure of the native service, the native service might not be expecting the same payload structure that is being sent. In this case, you would need to transform the request message into the format required by the native service before Mediator sends the requests to the native service. To do this, you invoke an XSLT file during the **Request Processing step**.

Assume that:

- The native service name is "authors".
- The virtual REST service or virtual XML service for "authors" is named "vs-authors" and is made available in Mediator at this endpoint: http://localhost:5555/ws/vs-authors/Invoke. The target-Namespace of the virtual REST service or virtual XML service is "http://example.com/authors".

Following is a sample XSLT transformation file for the GET-to-POST or GET-to-PUT scenario.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ns="http://example.com/authors"
    version="1.0">

   <xsl:output method="xml" omit-xml-declaration="no" standalone="yes" indent="yes"/>

    <xsl:strip-space elements="*"/>
    <xsl:template match="node()|@*">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
        </xsl:copy>
    </xsl:template>

    <xsl:template match="//ns:invoke/node()">
```

```
        <xsl:element name="{local-name(.)}">
            <xsl:value-of select="."/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="//ns:invoke">
        <xsl:element name="authors">
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Working with the JSON Content-Type

Mediator can accept a REST service request that specifies the Content-Type “application/json” (or “application/json/badgerfish”) and the HTTP methods PUT, GET, DELETE and POST.

Assuming that the native service supports both JSON and the HTTP method(s) specified in the request, Mediator can determine the correct service, operation and output format (JSON) to return to the consuming application. There are different ways in which a native service provider can be prompted to return response content. It will vary with the provider. For example, some providers may rely on the Accept transport header to specify the format the consumer wants. Others may use an element in the request or a query parameter on the URL.

However, suppose for example that the native service does not support the HTTP method specified in the request (e.g., POST). As a workaround, you can configure the virtual service so that it “bridges” this difference between the consumer request and the native service. In this case, you can configure the virtual service so that it takes the POST and bridges it into an HTTP GET query, and then returns the service to the consumer in the expected JSON format. To implement this, you set the following predefined context variables in a user-defined webMethods IS service that you can invoke in the virtual service’s Request Processing step and Response Processing step:

- `MESSAGE_TYPE`: A Content-Type defined in axis2.xml for a message formatter. This value must be a key in the axis2 message formatters list, since it is used to control message serialization. (The valid choices are defined as attributes of the <messageFormatters/> group in the Integration Server's axis2.xml configuration.)
- `BUILDER_TYPE`: A Content-Type defined in axis2.xml for a message builder. This value must be a key in the axis2 message builders list since it is used to control building of native service response messages. (The valid choices are defined as attributes of the <messageBuilders/> group in the Integration Server's axis2.xml configuration.)

This and other “bridging” scenarios are discussed in more detail below.

- How Mediator Determines Which Builder and Formatter Classes to Use (and How You Can Override Them)

How Mediator Determines Which Builder and Formatter Classes to Use (and How You Can Override Them)

Mediator makes these determinations at run time as follows. This table also summarizes how you can override the default determinations.

Run-time Step	Description
Mediator receives the request from the client	It is important that the client's PUT or POST requests contain the HTTP header Content-Type because the Content-Type header determines the message builder Mediator uses to parse the input stream. (GET or DELETE request do not require a Content-Type header.)

Mediator sends the request to the service provider	Mediator uses a message formatter to serialize the request, and then sends the serialized request to the native service provider.
	
	Mediator determines the message formatter to use as follows:
	
	■ If you explicitly specify a message formatter (by setting the MESSAGE_TYPE context variable in a webMethods IS service and invoking this service in the virtual service's Request Processing step), then Mediator uses that formatter. The Content-Type header that Mediator sends to the native provider is the one that is associated with the MESSAGE_TYPE context variable.
	■ Else, Mediator will use the message formatter associated with the Content-Type sent by the client (and sends the Content-Type to the native provider).
	■ Else, if no Content-Type was sent by the client, then:
	  ■ For PUT requests, the default formatter used (and the Content-Type header that Mediator sends to the native provider) is `application/xml`.
	  ■ For POST requests, the default formatter used (and the Content-Type header that Mediator sends to the native provider) is `SOAP` (and the request will fail).
	  ■ For GET or DELETE requests (which do not require a Content-Type header), the default formatter used (and the Content-Type header that Mediator sends to the native provider) is `application/x-www-form-urlencoded`.

Mediator receives a response from the service provider	When the provider returns a response to Mediator, a message builder parses the response stream into an Axiom message to be stored in the message context. Mediator determines which message builder to use as follows: 1. Mediator will select the message builder associated with the request's Accept transport header, if one was specified.

Run-time Step	Description
	2. Else, you can set the BUILDER_TYPE context variable in a webMethods IS service, and invoke this service in the virtual service's *Request* Processing step. Mediator will check that the builder type is a valid Content-Type for the list of builders in axis2.xml. This variable takes priority over the current setting specified in the Accept transport header. That is, Mediator will only use the Accept header to determine the builder type needed to parse a native provider response if no IS service was written to set the BUILDER_TYPE context variable. 3. Else, Mediator will use the builder associated with the Content-Type specified in the request (assuming that the Content-Type is one of the types that is mapped in axis2.xml). 4. Else, if no Content-Type was specified in the request (e.g., a PUT or POST request with no Content-Type, or any GET or DELETE request), or if the Content-Type is not one of the types that is mapped in axis2.xml, then Mediator will default to `application/xml`.
Mediator sends a response to the client	Mediator serializes the response and sends it to the client. By default, Mediator uses the formatter that was used to serialize the request sent to the provider. (If the formatter was `application/x-www-form-urlencoded` (for a GET or DELETE request), then Mediator will instead use `application/xml` so it can send the response.) You can override the Content-Type that is sent to the client by setting the MESSAGE_TYPE context variable in a webMethods IS service, and invoking this service in the virtual service's Response Processing step.

Scenarios for Requesting JSON Type Services

Following are some of the possible scenarios in which JSON type services can be requested. Many scenarios require that you "bridge" the differences between consumer requests and the native service (i.e., differing HTTP methods and Content-Types). Three of the scenarios are discussed in more detail following the table.

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging?
GET	GET	JSON	JSON	Request Processing step bridging (see **JSON Example 1: GET Request, JSON Response**)
POST/JSON	POST/JSON	JSON	JSON	No bridging needed
GET	GET	XML	JSON	Response Processing step bridging
POST/JSON	GET	JSON	JSON	Request Processing step bridging

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging?
POST/JSON	GET	XML	JSON	Request Processing step bridging and Response Processing step bridging (see **JSON Example 2: POST/JSON Request, JSON Response (where POST is not supported)**)
POST/JSON	XSLT/GET *	JSON	JSON	Request Processing step bridging
POST/JSON	XSLT/POST/XML *	XML	XML	Request Processing step bridging
POST/JSON	POST/JSON	JSON/XSLT *	JSON	Response Processing step bridging
GET	GET	JSON	XML	Request Processing step bridging and Response Processing step bridging (see **JSON Example 3: GET Request, XML Response**)
POST/XML	POST/JSON	JSON	JSON	Request Processing step bridging

* The XSLT references indicate where you can perform an XSLT message transformation at either the Request Processing or Response Processing step.

In the table above, the required Content-Type settings are not shown, but assume the following:

HTTP Method/Request Content	Required Axis2 Content Type
GET or DELETE	application/x-www-form-urlencoded
POST or PUT/XML	application/xml
POST or PUT/Mapped JSON	application/json
POST or PUT/Badgerfish	application/json/badgerfish

JSON Example 1: GET Request, JSON Response

In this example, a consumer sends a GET request to get a native JSON service. Mediator will send the response to the consumer in the requested JSON format (as indicated by the `"output=json"` parameter in the query).

The request looks like this:

```
http://localhost:5555/ws/YahooVS/search?query=wsdl20&output=json
```

and because this is a GET request, the Content-Type defaults to `application/x-www-form-urlencoded`.

Note: For GET or DELETE requests for REST services, it is not necessary to specify the Content-Type in the request; Mediator will default to `application/x-www-form-urlencoded` for GET or DELETE requests.

The run-time processing will be as follows:

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging
GET	GET	JSON	JSON	Request Processing step bridging

Since the request is a GET (i.e., of Content-Type `application/x-www-form-urlencoded`), but Mediator expects to receive a JSON stream from the provider, you must send the BUILDER_TYPE `application/json` to the native provider. To do this, write and invoke a webMethods IS service in the virtual service's Request Processing step. The IS service should include the following predefined context variable set to this value:

Context Variable	Value
BUILDER_TYPE	application/json

JSON Example 2: POST/JSON Request, JSON Response (where POST is not supported)

In this example, a consumer sends a POST request (of Content-Type `application/json`) to a native service, but the native service does not support POST.

The request's Content-Type is `application/json` and its `output` parameter is set to `xml`. The reason for this is explained below.

The run-time processing will be as follows:

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging
POST/JSON	GET	XML	JSON	■ Request Processing step bridging ■ Response Processing step bridging

Configure the virtual service as follows:

- In the virtual service's Routing Protocols tab, set the value of the HTTP Method field to the value GET. Doing this instructs Mediator to change the POST to an HTTP GET before Mediator sends it to the native service (which is necessary because the native service does not support POST).

- Write and invoke a webMethods IS service in the virtual service's *Request* Processing step. The IS service should include the following predefined context variables set to the values shown below. The request's `"query": "wsdl20"` and `"output": "xml"` parameters are transformed into query parameters on the URL before the native service is invoked. Thus, although the consumer is sending a JSON request, the native service is instructed to return an XML response to Mediator.

Context Variable	Value
MESSAGE_TYPE	application/x-www-form-urlencoded
BUILDER_TYPE	application/xml

- Write and invoke a webMethods IS service in the virtual service's *Response* Processing step. The IS service should include the following predefined context variable set to the value shown below. Doing this instructs Mediator to bridge the XML response from the native service into JSON format, to be returned to the consumer.

Context Variable	Value
MESSAGE_TYPE	application/json

JSON Example 3: GET Request, XML Response

In this example, a consumer sends a GET request to get a native service. Mediator will send the response to the consumer in the requested XML format (as indicated by the `"output=xml"` parameter in the query).

The request looks like this:

```
http://localhost:5555/ws/YahooVS/search?query=wsdl20&output=xml
```

and because this is a GET request, the Content-Type defaults to `application/x-www-form-urlencoded`.

The run-time processing will be as follows:

Consumer Sends	Mediator Sends Request to Provider	Mediator Receives Response from Provider	Mediator Sends Response to Consumer	Requirement for Bridging
GET	GET	JSON	XML	■ Request Processing step bridging ■ Response Processing step bridging

Since the request is a GET (i.e., of Content-Type `application/x-www-form-urlencoded`), but Mediator expects to receive a JSON stream from the provider, you must instruct Mediator to send the BUILDER_TYPE `application/json` to the native provider. To do this, write and invoke a webMethods IS service in the virtual service's *Request* Processing step. The IS service should include the following predefined context variable set to this value.

Context Variable	Value
BUILDER_TYPE	application/json

Since the provider will return a JSON stream to Mediator, but the consumer expects to receive the service in XML output format, you must set the MESSAGE_TYPE to `application/xml`. To do this, write and invoke a webMethods IS service in the virtual service's *Response* Processing step. The IS service should include the following predefined context variable set to this value:

Context Variable	Value
MESSAGE_TYPE	application/xml

Characteristics of the Mapped and Badgerfish JSON Conventions

The open source library that Axis2 uses to support JSON is called Jettison. The Jettison library supports two formats of JSON: Mapped JSON and Badgerfish. Both are syntactically correct from a JSON perspective.

Note: An important difference between the two is that the Mapped convention returns a service fault if a virtual service is configured for application/json (Mapped convention) and it encounters a message that has namespaces, while the Badgerfish convention attempts to avoid losing any meaning encoded in XML by preserving namespace declarations. The Axis2 JSON library MessageFormatter will complain if Mediator attempts to transform an XML response that contains namespace declarations. So, either make sure that the requests do not include namespaces, or else set the MESSAGE_TYPE to `application/json/badgerfish` instead of setting it to `application/json`.

Other characteristics include the following.

Mapped JSON Convention

1. An element with no characters or child elements is represented by:

```
{ "element" : "" }
```

2. No namespaces declarations are ever written.

Note: The Badgerfish convention does allow namespaces. If a client sends a request that contains XML namespaces, you need to bridge to the Badgerfish convention. To do this, in the virtual service's Routing Protocol's step, set the parameter **HTTP Headers** to

Customize and specify the Name as Content-Type and the Value as application/json/badgerfish. Doing this will override the existing Content-Type that will be sent to the native provider.

3. An element with multiple child elements of the same name is represented by an array.

   Simple case:

```
<price>10.00</price>
{ "acme.price" : { "10.00" }
```

   Array case:

```
<root><child>test</child><child>test</child></root>
{ "root" : { "child" : [ "test", "test" ] } }
```

4. The XML attributes for a message are prefixed with @ when a message is serialized (same as Badgerfish).

Badgerfish Convention

This convention is used to provide the means to translate between XML and JSON without losing any data (i.e., namespaces).

1. Element names become object properties.
2. Text content of elements goes in the $ property of an object.
3. Nested elements become nested properties.
4. Multiple elements at the same level become array elements.
5. Attributes go in properties whose names begin with @.
6. Active namespaces for an element go in the element's @xmlns property.
7. The default namespace URI goes in @xmlns.$.
8. Other namespaces go in other properties of @xmlns.
9. Elements with namespace prefixes become object properties, too.

   Simple example:

```
 <price xmlns="http://acme.com">10.00</price>
{ "price": { "@xmlns": { "$": "http://acme.com" }, "$1": "10.00" } }
```

A more complex example:

```
<alice xmlns="http://some-namespace" xmlns:charlie="http://some-other-namespace">
  <bob>david</bob>
  <charlie:edgar>frank</charlie:edgar>
</alice>

{ "alice" : { "bob" : { "$" : "david" , "@xmlns" : {"charlie"  :  ↩
"http:\/\/some-other-namespace" , "$" : "http:\/\/some-namespace"} } ,         ↩
     "charlie:edgar" : { "$" : "frank" , "@xmlns"  : ↩
{"charlie":"http:\/\/some-other-namespace", "$" :  "http:\/\/some-namespace"} }, ↩
              "@xmlns" : { "charlie" : "http:\/\/some-other-namespace",  "$" : ↩
"http:\/\/some-namespace"} } }
```

Multiple Root Nodes in JSON REST Services

With REST virtual services, when working with requests and responses of the Content-Type application/json, the message content can contain one or more root nodes. For example, a message might have the two root nodes `{“firstName”: “John”, “lastName”: “Smith”}`. Note the following points for messages with multiple root nodes.

- The XSLT provided in the Request/Response Processing step should have the XPath start with //, for example //firstName. This is because during the processing of the JSON content, Mediator will wrap the given content with system-defined elements.
- webMethods IS services that you invoke in the Request/Response Processing step will contain the JSON content in the variable called `JSONRESTContentString`. You can update the content in the IS service and put the updated content into the pipeline's input variable `UpdatedJSONRESTContentString`, which will be sent to the native service.

11 Creating Run-Time Policies

You create run-time policies and apply them to virtual services in order to govern the virtual services' run-time execution.

A run-time policy is a sequence of actions that are carried out by a policy-enforcement point (PEP) when a consumer requests a particular service through the PEP. The actions in a run-time policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance metrics. You create run-time policies using CentraSite Control and store them in the CentraSite registry/repository.

When you create a run-time policy in CentraSite Control, you:

- Specify the PEP target type (for example, webMethods Mediator) to which you will deploy the virtual services and their policies.
- Add run-time actions to the policy and configure their parameters. CentraSite provides a set of built-in run-time actions.
- Apply the policy to the desired virtual services.
- Activate the policy.

The content is organized under the following sections:

Actions that Run-Time Policies Can Execute

A run-time action is a single task that is included in a run-time policy and is evaluated by a policy-enforcement point (PEP). Actions in run-time policies perform tasks such as identifying/authenticating consumers and logging transaction activity. You specify actions when you define the policy. The PEP evaluates actions in the order in which they appear in the list of actions.

CentraSite provides *run-time action templates*. A run-time action template is a definition of an action that can be used in a run-time policy. Most action templates specify a set of parameters associated with a particular policy action. For example, when you configure the action that identifies consumers you specify an identifier (for example, an HTTP Authentication token) to identify the consumers who are trying to access the services. You can include multiple actions in a single policy.

Using the action templates, you can configure the following types of run-time actions:

- **WS-SecurityPolicy 1.2 actions**

  Mediator provides two kinds of actions that support WS-SecurityPolicy 1.2: authentication actions and XML security actions.

  - You use the authentication actions to verify that the service consumer has the proper credentials to access a virtual service. You can authenticate consumers by their WSS X.509 certificates, WSS Username tokens or WSS SAML tokens.

- You use the XML security actions to provide confidentiality (through encryption) and integrity (through signatures) for request and response messages.

- **Monitoring actions**

  Mediator provides the following run-time monitoring actions:

  - The “Monitor Service Performance” action, which monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when these conditions are violated.
  - The “Monitor Service Level Agreement” action, which provides the same functionality as “Monitor Service Performance”, but this action is different because it enables you to monitor a virtual service's run-time performance for particular consumers. You configure this action to define a *Service Level Agreement (SLA)*, which is set of conditions that defines the level of performance that a specified consumer should expect from a service.
  - The “Throttling Traffic Optimization” action (not available in Mediator versions below 9.0), which limits the number of service invocations allowed during a specified time interval, and sends alerts to a specified destination when the performance conditions are violated. You can use this action to avoid overloading the back-end services and their infrastructure, to limit specific consumers in terms of resource usage, etc.

- **Additional actions**

  Mediator provides the following actions, which you can use in conjunction with the actions above.

  - “Identify Consumer”, which you use in conjunction with an authentication action (“Require WSS Username Token”, “Require WSS X.509 Token” or “Require HTTP Basic Authentication”). Alternatively, you can use this action alone to identify consumers only by host name or IP address.
  - “Require HTTP Basic Authentication”, which uses HTTP basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header against the Integration Server's user account.
  - “Authorize User”, which authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which Mediator is running. You use this action in conjunction with an authentication action (“Require WSS Username Token”, “Require WSS SAML Token” or *Require HTTP Basic Authentication*).
  - “Authorize Against Registered Consumers”, which authorizes consumer applications against all Application assets that are registered in CentraSite as consumers for the service.
  - “Log Invocation”, which logs request/response payloads to a destination you specify.
  - “Validate Schema”, which validates all XML request and/or response messages against an XML schema referenced in the WSDL.

For detailed descriptions of these actions, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services* .

Who Can Create and Manage Run-Time Policies?

To create and manage run-time policies, you must belong to a role that includes the "Manage Runtime Policies" permission (or the "Manage System-wide Runtime Policies" permission).

By default, the following predefined roles include the "Manage Runtime Policies" permission:

- Organization Administrator
- Policy Administrator

By default, the following predefined roles include the "Manage System-wide Runtime Policies" permission:

- CentraSite Administrator
- Operations Administrator

Note: For more information about roles and permissions, see the section *Users, Groups, Roles, and Permissions* .

Creating a Run-Time Policy

To create a run-time policy in CentraSite, you perform the following high-level steps:

1. Create a new run-time policy. During this step, you select the actions you want the policy to execute, and you assign values to the action parameters.
2. Allow other users to view, edit and/or delete this policy by assigning permissions to those users. For procedures, see ***Setting Permissions on a Run-Time Policy***.
3. Activate the policy. During this step, you make the new policy ready to deploy to a PEP. For procedures, see ***Activating a Run-Time Policy***.

Perform these steps to create a run-time policy and save it to CentraSite.

To create a run-time policy

1 In CentraSite Control, go to **Policies > Run-Time**.

2 Click **Add Policy**.

3 In the **Policy Information** panel, specify the following fields:

In this field...	Specify...
Name	A name for the new policy. A policy name can contain any character (including spaces).
Description	*Optional*. A description for the new policy. This description appears when a user displays a list of policies in the user interface.

Version	*Optional*. A version identifier for the new policy.
	
	Note: The version identifier does not need to be numeric.
	
	Examples:
	
	`0.0a`
	`1.0.0 (beta)`
	`Pre-release 001`
	`V1-2007.04.30`
	
	Note: The version identifier you enter here is the policy's public, user-assigned version identifier. CentraSite also maintains an internal, system-assigned version number for the policy. For more information about user-assigned and system-assigned version identifiers, see *System-Assigned vs. User-Assigned Version Identifiers*.

4 In the **Scope** panel, specify the following fields. *Scope* refers to the set of properties that determine the target type, organization and asset type to which the policy applies.

In this field...	Specify...
Target Type	The target type to which the policy will be deployed. Select **webMethods Integration Server** (i.e., the webMethods Mediator target type).
Organization	The organization to which the policy applies. Choose **All** if you want to apply the policy to the specified services in all organizations. **Important:** Once you create a policy, its organizational scope is fixed and cannot be changed. That is, if you create a policy whose scope is specific to organization ABC, you cannot change its scope to make it system-wide or switch it to another organization. You must create a new policy and set its organizational scope as needed.
Asset Types	The type of asset to which this policy applies. Choose one of the following: ■ **Web Service** ■ **XML Service** ■ **REST Service** ■ **Virtual Service** ■ **Virtual XML Service** ■ **Virtual REST Service** **Note:** CentraSite does not provide out-of-the-box policy-enforcement for web services.

5 In the **Apply Policy to Services that Meet the Following Criteria** panel, specify criteria that identify the virtual services to which the policy applies.

To target a policy for a particular set of virtual services, you refine the policy's scope by specifying additional selection criteria based on the virtual service's Name, Description or Classification properties.

1. Choose an attribute (Name, Description or Classification) that identifies the services to which the policy applies.
2. Choose an operator for the attribute (if applicable).
3. Specify a value for the attribute (if applicable). Values are case-sensitive.
4. If you need to specify multiple values or attributes, use the plus button to add multiple rows. For example, for the Classification attribute you might choose multiple Taxonomy names. If you specify multiple criteria, they are connected by the AND operator.

After you save the policy, you will see the generated list of services displayed on the Policy Detail page's **Services** profile.

Notes:

1. If you specify no criteria, the policy will apply to all virtual services.
2. You can specify only *one* "Name Equals *<value>*" condition. However, you can specify multiple "Name Contains *<value>*" or "Name Starts With *<value>*" conditions.

Caution: CentraSite checks for policy conflicts when you deploy a virtual service to Mediator. If the service has only one policy applied to it (the policy you are applying here), that policy is deployed to Mediator, and Mediator executes the policy's run-time actions in the order in which they appear in the policy. However, if the service already has additional policies applied to it, a policy conflict might occur, which might cause unintended consequences. CentraSite will inform you of policy conflicts. For information about how Mediator evaluates actions (and how to avoid policy conflicts), see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services > Action Evaluation Order and Dependencies* .

6 Click **Next**.

7 In the **Available Actions** dialog, select the built-in actions that you want to include in the policy. Keep the following points in mind when you select the actions for the policy:

- If you are using webMethods Mediator as your PEP, you must include the "Identify Consumer" built-in action (and optionally other identification actions) in order to identify or authenticate consumers. For common usage cases of identification actions, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services > Usage Cases for Identifying/Authenticating Consumers* .

- Ensure that the actions in the **Selected Actions** list appear in the order in which you want them to run when the policy is enforced. If necessary, use the control buttons on the right side of the list to place them in the correct order.

8 Click **Finish** to save the new (as yet incomplete) policy. The Runtime Policy Detail page is displayed, showing details of the policy you just created.

9 Specify parameter values for each of the policy's actions as follows:

1. In the **Actions** profile, choose the action whose parameters you want to set.
2. In the **Edit Action Parameters** page, set the parameters as necessary and click **Save**. Required parameters are marked with an asterisk. For detailed information about the parameters, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services* .
3. When you have finished setting the parameters of all actions in the list, click **Save** and then **Close**. The icons next to the actions in the **Parameters Set** column will indicate whether the action parameters have been set.

Icon	Description
	The action has required input parameters that have not yet been set.
	All of the action's required input parameters have been set. **Note:** This icon automatically appears for actions that have no input parameters.

10 Complete the policy by doing the following:

1. If you want to allow other users to view, edit and/or delete this policy, go to the **Policy Detail** page, select the **Permissions** profile, and assign permissions to those users. You will not see this profile unless you belong to a role that has the "Manage Runtime Policies" permission. For procedures, see the section *Users, Groups, Roles, and Permissions* .
2. Activate the policy when you are ready to put it into effect. For procedures, see ***Activating a Run-Time Policy***.

Setting Permissions on a Run-Time Policy

If you want to permit other users to manage (i.e., view, edit and delete) run-time policies, you do so by adding to the policy's instance-level permission settings. A policy has the following types of instance-level permission settings.

Permission	Description
View	Enables users to see the policy in their policy list and view details for the policy.
Modify	Enables users to view and modify the properties of a policy (including the policy's scope and action list).
Full	Enables users to view, modify or delete the policy.

Who Can Set Permissions on a Run-Time Policy?

To set permissions on a policy, you must belong to a role that has "Manage Runtime Policies" permission or the "Manage System-wide Runtime Policies" permission.

Assigning Permissions to a Run-Time Policy

You can assign view, edit and delete permissions to any individual user or group defined in CentraSite.

Note: If you give a user permission to view, edit or delete a policy, and you want that user to be able to perform these operations using CentraSite Control, make sure that the user belongs to a role that also has the "Use the Policy UI" permission.

To assign permissions to a policy

If you are modifying the permissions of an active policy, you must first deactivate the policy.

1 Display the Policy Detail page for the policy that you want to activate. If you need procedures for this step, see *Modifying a Run-Time Policy*.

2 Select the **Permissions** profile.

3 To add a new a new user or group to the list, do the following:

1. Choose **Add Users / Groups**.
2. Select the users and/or groups to which you want to assign permissions.

   If you want to filter the user list, specify a pattern-matching string in the **Search** field. The pattern-matching string can consist of one or more characters and/or the % wildcard symbol. The string is matched against group names and user IDs.

If you type...	CentraSite will return...
b%	Groups and user names that start with the letter b.
b	Groups and user names that contain the letter b.
%b	Groups and user names that end with b.
%	All groups and user names.

Examples

3. Choose **OK**.

4 Use the **Full**, **Modify** and **View** check boxes to assign specific permissions to the users and groups in the list.

5 To remove an entry from the list, select the check box beside the group or user name and click **Delete**.

6 Click **Save** to save your settings.

Activating a Run-Time Policy

When you activate a run-time policy, CentraSite applies it to the services you specified in the policy.

To activate a policy, you change the policy's lifecycle state to the "Productive" state. This state change executes CentraSite's *Automatic Policy Activation* policy.

Note: The *Automatic Policy Activation* policy is a hidden system policy. You cannot edit or delete this policy.

When you activate a policy, be aware that:

- You will not be allowed to activate the policy unless all of its parameters have been set. When you switch the policy to the Productive state, CentraSite executes the *Validate Policy Activation* policy. This policy will not allow you to switch a policy to the Productive state if the policy's parameters have not yet been set.
- Some organizations require an approval to activate a policy. If your organization has an approval action associated with the activation of a policy, CentraSite will not activate the policy until the required approvals are obtained. For more information about approval actions, see the section *Working with Design/Change-Time Policies*.
- If an earlier version of the policy is already active, CentraSite will deactivate the old version before it activates the new one. For more information about working with versioned policies, see ***Versioning a Policy***.
- To activate a policy, you must have permission to change the policy to the Productive state.

- To successfully change a policy to the "Productive" state, you must also have the Modify permission on all virtual type services to which the policy is applied.

To determine whether a policy is active or inactive, examine the policy's **Active** indicator on the **Policies** > **Run-Time** page. The icon in the **Active** column indicates the policy's activation state as follows:

Icon	Description
	Policy is active.
	Policy is inactive.

The activation state of a policy is also reported next to the **State** field in the Run-Time Policy Details page.

To activate a policy

1 Display the Run-Time Policy Details page for the policy you want to activate. If you need procedures for this step, see ***Modifying a Run-Time Policy***.

2 Examine the **Actions** profile and verify that all of the actions on this profile display the green checkmark icon in the **Parameters Set** column. If any of the actions display the red circle icon in this column, set their parameters before you continue. For information about setting action parameters, see ***Modifying Action Parameters***.

3 In the **Policy Information** panel, click the **Change State** button. (If you do not see the **Change State** button, it is probably because you do not have permission to change the lifecycle state of a policy.)

4 In the **Change Lifecycle State** dialog box, select the **Productive** lifecycle state and click **OK**.

5 Examine the **State** field in the **Policy Information** panel to verify that the policy's state has been changed.

If this state change requires approval, the **State** field will indicate that the policy is in the "pending" mode. CentraSite will automatically switch the policy to the requested state (and activate the policy) after all the necessary approvals have been obtained. For information about checking the status of objects that you have submitted for approval, see the section *Working with Design/Change-Time Policies*.

Note: While the policy is in pending mode, it cannot be edited.

6 After you activate the policy, users with the proper permissions will deploy the services to your PEP. At that time, CentraSite will automatically validate the service's policies (e.g., check for policy conflicts or other violations). For information about how Mediator evaluates actions (and how to avoid policy conflicts), see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services > Action Evaluation Order and Dependencies* .

Deactivating a Run-Time Policy

You usually deactivate a policy if you want to edit the policy (for example, to modify the scope of a policy or change its action list).

To deactivate a policy, you change the policy to the Suspended state. Switching the policy to this state triggers the *Automatic Policy Deactivation* policy, which deactivates the policy. (Switching the policy to the Retired state also deactivates the policy, but you do not want to switch a policy to this state unless you intend to deactivate it permanently. After you place a policy in the Retired state, you cannot reactivate it.)

To deactivate a policy, you must have permission to change the policy to the Suspended state.

To deactivate a policy

1 Display the Run-Time Policy Details page for the policy you want to activate. If you need procedures for this step, see ***Modifying a Run-Time Policy***.

2 In the **Policy Information** panel, click the **Change State** button. (If you do not see the **Change State** button, it is probably because you do not have permission to change the lifecycle state of a policy.)

3 In the **Change Lifecycle State** dialog box, select the **Suspended** state (to deactivate it temporarily) or the **Retired** state (to deactivate it permanently), then click **OK**

Viewing the Run-Time Policy List

The Run-Time Policies page displays a list of all run-time policies residing on the CentraSite server.

To view the policy list

1 In CentraSite Control, go to **Policies > Run-Time**.

2 By default, all of the available policies are displayed.

If you want to filter the list to see just a subset of the available policies, type a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those policies whose name contains the specified characters. The wildcard character "%" is supported.

3 You can sort the list by type of asset. To specify the sorting order, choose either **Web Service**, **REST Service**, **XML Service**, **Virtual Service**, **Virtual REST Service** or **Virtual XML Service** from the drop-down list labeled **Browse by**.

The Run-Time Policies page provides the following information about each policy.

Note: Only the first six columns described below are displayed in this list by default. To display the additional columns, click the **Select Columns** button.

Column	Description	
Name	The name assigned to the policy.	
Description	Additional comments or descriptive information about the policy.	
System Version	The automatically generated system-assigned version identifier for the policy. For more information about system-assigned version identifiers, see, ***System-Assigned vs. User-Assigned Version Identifiers***.	
Organization	The organization to which the policy applies.	
	This value...	**Indicates that...**
	All	The policy is system-wide and applies to all organizations.
	OrgName	The policy applies to the specified organization.
State	The policy's current lifecycle state.	
Active	The policy's current enforcement state.	
	Icon	**Description**
		The policy is active. CentraSite enforces this policy when events within the scope of the policy occur.
		The policy is inactive. Inactive policies exist in the registry, but they are not enforced.
Version	The user-assigned version identifier for the policy. For more information about user-assigned version identifiers, see, ***System-Assigned vs. User-Assigned Version Identifiers***.	
Owner	The user to which the policy belongs.	

Modifying a Run-Time Policy

The following sections discuss:

Viewing or Changing a Run-Time Policy

You use the Run-Time Policy Details page to examine and/or edit the properties for a policy. When editing a policy, keep the following points in mind:

- To edit a policy, you must have Modify permission on the policy. If your user account belongs to a role that has the "Manage Runtime Policies" permission for an organization, you automatically have permission to modify all of the policies in that organization. If your user account belongs to a role that has the "Manage System-Wide Runtime Policies" permission, you have permission to edit any system-wide policy on the server.
- You cannot modify an active policy. If the policy that you want to edit is in the active state, you must deactivate it or use the versioning feature to create a new version of the policy. Creating a new version of the policy allows you to make your changes without having to deactivate the existing policy. For details, see ***Versioning a Policy***. For information about deactivating a policy, see ***Deactivating a Run-Time Policy***.
- If you modify a run-time policy for a virtual service that is already deployed to Mediator, CentraSite will automatically redeploy the modified virtual service.

To edit a policy

1 In CentraSite Control, go to **Policies > Run-Time** to display the policy list.

2 Locate the policy whose details you want to view or edit and choose **Details** from its context menu.

3 Examine or modify the properties on the **Policy Detail** page as necessary.

Field or Profile	Description
Name	The name of the policy. A policy name can contain any character (including spaces).
Description	*Optional*. Additional comments about the policy.
Version	The user-defined version number to be assigned to the new version. We recommend that you update the version number anytime you make significant modifications to a policy. The usage of this field is the same as described in ***Creating a Run-Time Policy***.

Field or Profile	Description	
State	The policy's current lifecycle state (e.g., New, Productive, Suspended, Retired). This field also displays an icon that indicates the activation state of the policy:	
	Icon	**Description**
		The policy is active (i.e., ready to be deployed to a PEP).
		The policy is inactive.
Organization	The organization to which the policy belongs.	
Owner	The user who created the policy.	
System Version	The automatically-generated system version identifier for the policy. For more information, see *Creating a Run-Time Policy*.	
Actions profile	The settings in this profile specify the actions that the PEP will execute when the policy is enforced. For more information about the properties on this profile, see *Creating a Run-Time Policy*.	
Scope profile	The settings in this profile determine the services to which the policy is applied. For more information about the properties on this profile, see *Modifying the Scope of a Run-Time Policy*.	
Services profile	Displays the list of Web services and/or virtual services to which the policy applies. For more information, see *Viewing the List of Services to Which a Policy Applies*.	
Permissions profile	The settings in this profile identify which users can view, edit and/or delete the policy. For more information about the properties on this profile, see *Setting Permissions on a Run-Time Policy*.	

4 If you edited the settings on the **Run-Time Policy Details** page, click **Save** to save the updated policy.

5 If you deactivated the policy in order to edit it, activate the policy as described in *Activating a Run-Time Policy*.

You can view the details page of multiple policies as follows:

To view the details page of multiple policies

1 In CentraSite Control, go to **Policies > Run-Time** to display the policy list.

2 Locate the policies whose details you want to view, and mark their checkboxes.

3 In the **Actions** menu, choose **Details**.

The details page of each of the selected policies will now be displayed.

Modifying Actions for a Run-Time Policy

This section discusses:

Modifying the Actions List

Use the procedure in this section to modify the actions list.

To modify the actions list

1 Display the **Policy Detail** page for the policy whose actions you want to edit. If you need procedures for this step, see ***Modifying a Run-Time Policy***.

2 If the policy is active, deactivate it. You cannot modify an active policy. If you need procedures for this step, see ***Deactivating a Run-Time Policy***.

3 Select the **Actions** profile to display the list of actions associated with the policy.

4 To add actions to, delete actions from, or modify the order of actions in the list, do the following:

1. Click **Edit Actions List**.
2. Use the controls in the **Edit Assigned Actions List** dialog box to add actions to the list and/or delete actions from the list.

   When editing the list of actions, keep the following point in mind:

   - If you are using webMethods Mediator as your PEP, you must include the built-in run-time action "Identify Consumer" (and possibly other authorization/identification actions) in order to identify or authenticate consumers. For common usage cases of identification actions, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services > Usage Cases for Identifying/Authenticating Consumers*.
   - If necessary, you can click **Previous** to return to the previous page and modify the information in the **Policy Information** panel.
3. Make sure that the actions in the **Selected Actions** list appear in the order that you want them to run when the policy is enforced. If necessary, use the control buttons on the right side of the list to place them in the correct order.
4. Click **OK** to save the modified list.

5 Use the procedure in ***Modifying Action Parameters*** to specify parameter values for any new actions that you might have added to the list, or to make any necessary updates to the parameter values for existing actions.

6 When the action list is complete and you have configured all of the input parameters for the actions correctly, click **Save** to save the updated policy.

7 If you deactivated the policy in order to edit it, activate the policy as described in *Activating a Run-Time Policy*.

Modifying Action Parameters

Policy actions have parameters that you set to configure the behavior of the action at enforcement time. When you display the **Actions** profile on the Policy Detail page, the icon in the **Parameters Set** column indicates whether an action has input parameters that need to be set.

This icon...	Indicates that...
	The action has required input parameters that have not yet been set.
	All of the action's required input parameters have been set. **Note:** This icon automatically appears for actions that have no input parameters.

Until the green check mark icon appears for all actions, you will not be able to activate the policy (if it is inactive) or save the policy (if it is already active).

To modify action parameters

1 Display the **Policy Detail** page for the policy whose actions you want to edit. If you need procedures for this step, see *Modifying a Run-Time Policy*.

2 If the policy is active, deactivate it. You cannot modify an active policy. If you need procedures for this step, see *Deactivating a Run-Time Policy*

3 In the **Actions** profile do the following for each action in the list.

1. Choose the action whose parameters you want to examine or set.
2. In the **Edit Action Parameters** page, set the parameters as necessary.

Note: Required parameters are marked with an asterisk. For more information about built-in actions and their parameters, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services* .

3. Click **Save** to save the parameter settings.

4 After you configure the parameters for all of the actions in the list, click **Save** to save the updated policy.

5 If you deactivated the policy in order to edit it, activate the policy as described in *Activating a Run-Time Policy*.

Modifying the Scope of a Run-Time Action

You can use the **Scope** profile on the Run-Time Policy Detail page to modify a policy's scope.

Note: Once you create a policy, its organizational scope is fixed and cannot be changed. That is, if you create a policy whose scope is specific to organization ABC, you cannot change its scope to make it system-wide or switch it to another organization. You must create a new policy and set its organizational scope as needed.

To modify the scope of a run-time policy

1 Display the Policy Detail page for the policy that you want to edit. If you need procedures for this step, see *Modifying a Run-Time Policy*.

2 If the policy is active, deactivate it. You cannot modify an active policy. If you need procedures for this step, see *Deactivating a Run-Time Policy*.

3 In the **Scope** profile, you can modify the following fields.

Field or Profile	Description
Target Type	The target type to which the policy will be deployed (e.g., webMethods Integration Server).
Asset Type	The type(s) of assets to which this policy applies. **Note:** CentraSite does not provide out-of-the-box policy-enforcement for web services.
Organization	Read-only field. You cannot change the policy's organization after the policy has been created.
Apply Policy to Services that Meet the Following Criteria	The services to which the policy applies. ■ Choose an attribute (e.g., Name, Description, Classification) that identifies the services to which the policy applies. ■ Choose an appropriate operator. ■ Specify a value. ■ If you need to specify multiple criteria, use the plus button to add multiple rows. For example, for the Classification attribute you might choose multiple Taxonomy names. After you save the policy, you will see the generated list of services displayed on the **Policy Detail** page's **Services** profile. **Note:** If you specify multiple criteria, the criteria are connected with the AND operator. That means that *all* criteria must be satisfied, in order for the policy to apply to any of the services. If not all the criteria are satisfied, the policy will not apply to any of the services.

4 Click **Save** to save the modified policy.

5 If you deactivated the policy in order to edit it, activate the policy as described in *Activating a Run-Time Policy*.

Viewing the List of Services To Which a Run-Time Policy Applies

Use the following procedure to view the list of services to which a run-time policy applies.

Important: The list only includes services that are in the "deployable state" (that is, services whose deployment state has been enabled by the Change Deployment State action in a design-time policy). Services that are within the scope of the policy, but have not yet been made "deployable", do not appear in this list.

To view the list of services

1 Display the Policy Detail page for the policy that you want to view. If you need procedures for this step, see *Viewing or Changing a Run-Time Policy*.

2 Select the **Services** profile to view the list of virtual services and/or Web services that was generated based on the criteria you specified in the **Scope** profile.

3 To view details of a service, click its hyperlinked name.

Note: To add or remove services to or from the list, return to the **Scope** profile and change the criteria in the **Apply Policy to Services that meet the following Criteria** panel. For more information, see the procedure in *Modifying the Scope of a Run-Time Policy*.

Deleting a Run-Time Policy

You delete a policy to remove it from CentraSite permanently.

CentraSite is installed with a system-wide policy called *Check State Validation Policy for Policy*. This policy will not allow you to delete a policy unless the policy is in the New or Retired state.

To delete a policy

1 In the CentraSite Control, go to **Policies > Run-Time** to display the policy list.

2 Ensure that the policy is deactivated (see **Deactivating a Run-Time Policy**).

3 Enable the checkbox next to the name of the policy that you want to delete.

4 Click **Delete**.

You can delete multiple policies in a single step. The rules described above for deleting a single policy apply also when deleting multiple policies.

To delete multiple policies in a single operation

1 In CentraSite Control, go to **Policies > Run-Time** to display the policy list.

2 Mark the checkboxes of the policies that you want to delete.

3 From the **Actions** menu, choose **Delete**.

Versioning a Run-Time Policy

When you need to make changes to an existing policy, creating a new version of the policy is an efficient way to accomplish this task. Versioning a policy enables you to create a new version of a policy (which is an identical copy of the existing policy) and make your changes to the new version. When you are ready to put the updated policy into effect, you simply "activate" the new version of the policy. When you activate the new version, CentraSite automatically deactivates and retires the old version of the policy.

When you create a new version of a policy, be aware that:

- You can only create a new version from the *latest version* of a policy. For example, if a policy already has versions 1.0, 2.0 and 3.0, CentraSite will only allow you to create a new version of the policy from version 3.0. (It makes no difference whether the policy that you are versioning is active or inactive. You can version a policy in either mode.)
- To create a new version of a policy, you must have permission to manage run-time policies for the organization specified in the scope of the original policy. For example, if the original policy is scoped for organization ABC, you will not be permitted to create a new version of that policy unless you have permission to manage design/change-time policies for organization ABC.
- When CentraSite creates a new version of a policy, it produces a version that is identical to the previous version, except that:
  - The new policy's system-assigned version identifier is incremented by one. (For additional information about system-assigned version numbers, see ***System-Assigned vs. User-Assigned Version Identifiers***.)
  - Ownership of the new policy is assigned to the user who created the new version.
- Like all new policies, the new version begins its lifecycle in the New state and is marked as inactive.
- CentraSite automatically establishes a relationship between the new version of the policy and the previous version. This relationship enables several capabilities and features in CentraSite that relate to versioned policies.

To version a policy

1 In CentraSite Control, go to **Policies > Run-Time** to display the policy list.

2 Locate the *most recent version* of the policy for which you want to create a new version and choose **Create New Version** from its context menu.

3 Modify the new version of the policy as necessary and then save it.

4 When you are ready to put the new version into effect, activate the new policy. CentraSite will automatically deactivate and retire the previous version.

Note: If you activate the new version of the policy while CentraSite is in the middle of executing the old version, your activation request will fail. If this occurs, wait for a period time and then try to activate the new version of the policy again.

System-Assigned vs. User-Assigned Version Identifiers

CentraSite maintains two version identifiers for a policy: a *system-assigned identifier* and a *user-assigned identifier*.

- The system-assigned identifier is a version number that CentraSite maintains for its own internal use. CentraSite automatically assigns this identifier to a policy when the policy is created. You cannot delete it or modify it. A policy's system-assigned identifier is numeric and always has the format `MajorVersion.Revision`. A policy always begins with a system-assigned version identifier of 1.0. The `MajorVersion` number is incremented by one each time you create a new version of a policy (e.g., 1.0, 2.0, 3.0).

  A policy's system-assigned version number is shown in the **System Version** column on the Run-Time Policies page and in the **System Version** field of the policy's detail page.

- The user-assigned identifier is an optional identifier that you can assign to a distinguish a specific version of a policy. This identifier does not need to be numeric. For example, you might use a value such as “V2.a (beta)” to identify a version.

  A policy's user-assigned version number is shown in the **Version** column on the Run-Time Policies page and in the **Version** field of the policy's detail page.

Asymmetric Binding Configuration

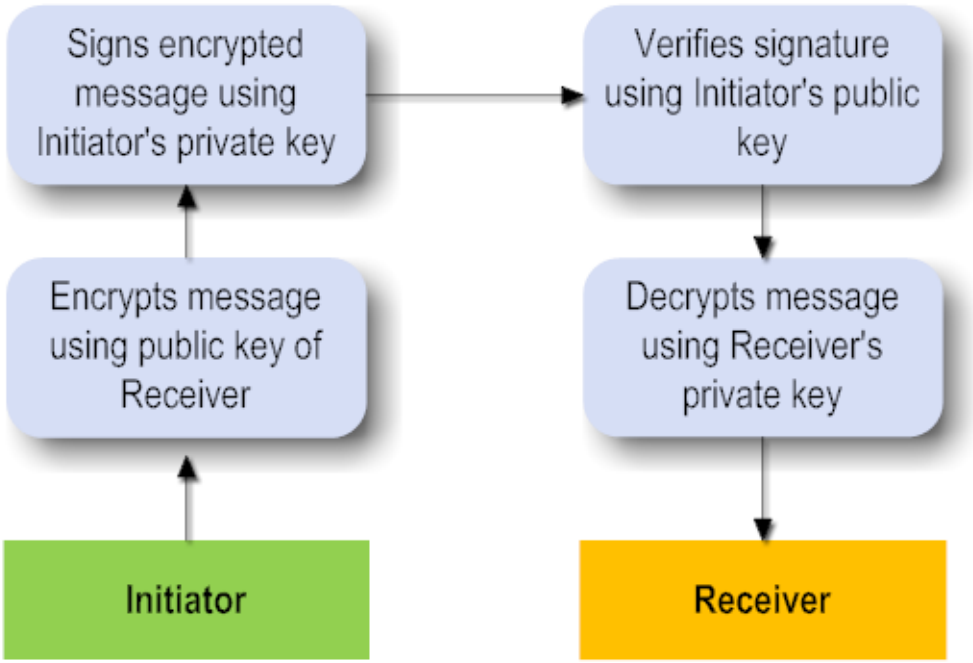
WS-SecurityPolicy specification deals with three types of Security Bindings. A security binding determines how the message transfer is to be done between the recipient and the initiator.

This section discusses:

- Asymmetric Binding
- The Asymmetric Binding Components
- The Asymmetric Binding Configuration Command Tool

Asymmetric Binding

Asymmetric Binding is used when both the Initiator and the Recipient possess public and private keys. The message transfer takes place using Public Key Infrastructure.



An Asymmetric Binding element in the WSDL looks like this:

```
<sp:AsymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <wsp:Policy>
       <sp:InitiatorToken>
       </sp:InitiatorToken>
       <sp:RecipientToken>
       </sp:RecipientToken>
       <sp:AlgorithmSuite>
          <wsp:Policy>
             <sp:TripleDesRsa15/>
          </wsp:Policy>
       </sp:AlgorithmSuite>
       <sp:Layout>
          <wsp:Policy>
             <sp:Strict/>
          </wsp:Policy>
       </sp:Layout>
       <sp:IncludeTimestamp/>
       <sp:OnlySignEntireHeadersAndBody/>
     </wsp:Policy>
 </sp:AsymmetricBinding>
```

The following run-time actions that support WS-Security policies use a common Asymmetric Binding element:

- The “Require Encryption” action.
- The “Require Signing” action.
- The “Require WSS SAML Token” action.
- The “Require WSS X.509 Token” action.

The Asymmetric Binding Components

The components of a security binding are as follows:

- Initiator Token Inclusion
- Recipient Token Inclusion
- Algorithm Suite

- Layout

Initiator Token Inclusion

The value of Initiator Token Inclusion specifies how to include the initiator token during message exchange from Initiator to Recipient or Recipient to Initiator.

Value	Description
Always	Always include the token.
AlwaysToRecipient	Include the token in all message exchanges from the Initiator to Recipient.
AlwaysToInitiator	Include the token in all message exchanges from the Recipient to Initiator.
Once	Only once.
Never	Do not include the token in any communications.

Recipient Token Inclusion

The value of Recipient Token Inclusion specifies how to include the Recipient token during message exchange from Initiator to Recipient or Recipient to Initiator. It takes the same values as Initiator Token Inclusion above.

Algorithm Suite

The value of Algorithm Suite specifies the algorithm suite to be used for this asymmetric binding. The possible algorithms supported are:

- Basic 256
- Basic192
- Basic128
- TripleDes
- Basic256Rsa15
- Basic192Rsa15
- Basic128Rsa15
- TripleDesRsa15
- Basic256Sha256
- Basic192Sha256
- Basic128Sha256
- TripleDesSha256
- Basic256Sha256Rsa15
- Basic192Sha256Rsa15

- Basic128Sha256Rsa15

Layout

Layout describes the way information is added to the message header. The possible values are:

Value	Description
Strict	Items are added to the security header in a principle of “declare before use”.
Lax	Items are added to the security header in any order that conforms to WSS: SOAP Message Security.
LaxTsFirst	Same as Lax except that the first item in the security header *must* be a `wsse:Timestamp`. Note that the `wsse:Timestamp` property *must* also be set to true in this case.
LaxTsLast	Same as Lax except that the last item in the security header *must* be a `wsse:Timestamp`. Note that the `wsse:Timestamp` property *must* also be set to true in this case.

The Asymmetric Binding Configuration Command Tool

You can change the Asymmetric Binding configuration by executing the following commands in the command line interface *CentraSiteCommand.cmd* (Windows) or *CentraSiteCommand.sh* (UNIX) of Command Central. The tool is located in `<CentraSiteInstallDir>`/utilities.

If you start this command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter "-url" must be specified as shown and not as "-URL".

The configuration is maintained in the form of an XML file which is loaded with default values.

```
<?xml version="1.0" encoding="UTF-8" ?>
<AsymmetricBindingConfiguration>
<initiatorTokenInclusion>AlwaysToRecipient</initiatorTokenInclusion>
<recipientTokenInclusion>Never</recipientTokenInclusion>
<algorithmSuite>TripleDesRsa15</algorithmSuite>
<layout>Strict</layout>
</AsymmetricBindingConfiguration>
```

The commands include:

- get Asymmetric Binding
- set Asymmetric Binding

- remove Asymmetric Binding

get Asymmetric Binding

Use this command to obtain the current Asymetric binding configuration values. Use a command of the following format:

```
CentraSiteCommand get Asymmetric Binding [-url <CENTRASITE-URL>] -user <USER-ID>
-password <PASSWORD>
```

The following table describes the complete set of input parameters that you can use with the `get Asymmetric Binding` utility:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user` .

For example:

```
CentraSiteCommand get Asymmetric Binding [-url ↩
"http://localhost:53307/CentraSite/CentraSite"] -user "Administrator"
-password "manage"
```

set Asymmetric Binding

Use this command to change the values in the CentraSite registry/repository.

```
CentraSiteCommand set Asymmetric Binding [-url <CENTRASITE-URL>] -user <USER-ID>
-password <PASSWORD> [-initiatorTokenInclusion <INITIATOR-TOKEN-INCLUSION>]
[-reciepientTokenInclusion <RECIPIENT-TOKEN-INCLUSION>] [-algorithmSuite
<ALGORITHM-SUITE>] [-layout <LAYOUT>]
```

The following table describes the complete set of input parameters that you can use with the `set Asymmetric Binding` utility:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user` .
`-initiatorTokenInclusion`	Inclusion value for the Initiator Token. *
`-reciepientTokenInclusion`	Inclusion value for the Recipient Token. *
`-algorithmSuite`	The algorithm to be used. *
`-layout`	The layout to be used. *

* At least one of the following parameters is required: initiatorTokenInclusion, reciepientTokenInclusion, algorithmSuite and layout.

remove Asymmetric Binding

Use this command to remove the Asymmetric Binding configuration. It has the same parameters as the get Asymmetric Binding command:

```
CentraSiteCommand remove Asymmetric Binding [-url <CENTRASITE-URL>] -user <USER-ID>
-password <PASSWORD>
```

The following table describes the complete set of input parameters that you can use with the `remove Asymmetric Binding` utility:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.

Input Parameter	Description
`-password`	The password of the user identified by the parameter `-user` .

For example:

```
CentraSiteCommand remove Asymmetric Binding [-url ↩
"http://localhost:53307/CentraSite/CentraSite"] -user "Administrator"
-password "manage"
```

12 Consumer Applications

A consumer application is a computer application that consumes (invokes) assets (services, BPEL processes or XML schemas) at run time. Typically, you create consumer applications to specify the consumers that are allowed to consume a particular asset. Then, you include the consumer application in the **Consumers** profile of the asset.

A consumer application in CentraSite is represented by an *Application* asset. The Application asset defines the precise consumer identifiers (for example, a list of user names in HTTP headers, a range of IP addresses, etc.). Thus the policy enforcement point (such as Mediator) can identify or authenticate the consumers that are requesting an asset.

You can use Application assets with any supported policy enforcement point (that is, webMethods Mediator or any supported third-party policy enforcement point).

Note: If you want to authenticate consumers (using LDAP or another external authentication mechanism), make sure that your policy enforcement point is configured to enable authentication. For information, see the documentation for your policy enforcement point.

The following subsections describe how to create and manage consumer applications:

Who Can Create and Manage Consumer Applications?

To create and manage consumer applications, you must belong to a role with the following permissions:

- Create Assets —OR— Manage Assets
- Manage Lifecycle Models (required to change state of consumer applications)

For more information about roles and permissions, see the section *About Roles and Permissions* in the document *Users, Groups, Roles and Permissions*.

Identifying Consumer Applications

To identify consumer applications, you perform the following high-level steps:

1. **Include the Identify Consumer action in the asset's run-time policy.**
   To determine the consumer application from which a request was submitted, an asset must have a run-time policy that includes the Identify Consumer action (and optionally other identification actions). In this action, you specify the consumer identifier(s) you want to use for identifying consumer applications. (Alternatively, you may configure this action to allow unrestricted access.) This action extracts a specified identifier from an incoming request and locates the application asset defined by that identifier. For more information about the Identify Con-

sumer action, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services* .

For example, if you configure the Identify Consumer action to identify consumers by IP address, the PEP extracts the IP address from a request's HTTP header at run time and searches its list of application assets for the application that is defined by that IP address.

You can configure the Identify Consumer action to identify consumer applications based on one or more of the following consumer identifiers in a request message:

Consumer Identifier	Description
IP Address	The IP address from which the request originated.
Host Name	The name of the host machine from which the request originated.
HTTP Authentication Token	The user ID submitted by the requestor when it was asked to provide basic HTTP credentials (user name and password).
WS-Security Authentication Token	The WSS username token supplied in the header of the SOAP or XML request that the consumer application submitted to the virtualized service.
Custom Identification	A string produced by applying a specified XPath expression to the SOAP or XML request that the consumer application submitted to the virtualized service.
Consumer Certification	The X.509 certificate supplied in the header of the SOAP or XML request that the consumer application submitted to the asset.

When deciding which type of identifier to use to identify a consumer application, consider the following points:

- Whatever identifier you choose to identify a consumer application, it must be unique to the application. Identifiers that represent user names are often not suitable because the identified users might submit requests for multiple applications.
- Identifying applications by IP address or host name is often a suitable choice, however, it does create a dependency on the network infrastructure. If a consumer application moves to a new machine, or its IP address changes, you must update the identifiers in the application asset.
- Using X.509 certificates or a custom token that is extracted from the SOAP or XML message itself (using an XPATH expression), is often the most trouble-free way to identify a consumer application.

For details of the Identify Consumer action, and for common usage cases of identification actions, see the section *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services > Usage Cases for Identifying/Authenticating Consumers*.

2. **Create an application asset in the registry.**
   In the application asset you specify precise values for the consumer identifier(s) that you specified in the Identify Consumer action. For details, see *Creating a Consumer Application Asset*.
3. **Specify the application asset in the Consumers profile of the asset to be consumed.**
   The Consumers profile is located in the asset's detail page.

The run-time behavior of identifying consumers is as follows:

1. CentraSite translates the application asset to the appropriate WS-Security policy assertions or an equivalent XML when the application asset is enforced by the PEP.
2. When a consumer application requests access to an asset, the PEP tries to map the consumer's identifier (which is found in the request) to an identifier in the application asset.
   - If the identifier is an IP address, a host name, a custom identification string or a consumer certificate, the PEP tries to identify the consumer (the consumer is not authenticated).
   - If the identifier is an HTTP Authentication token or a WS-Security Authentication token, the PEP tries to *authenticate* the consumer. If you use webMethods Mediator, authentication is handled by LDAP or by another external authentication mechanism, depending on how Mediator is configured. If you use a third-party PEP, authentication capabilities depend on the PEP.
3. The identified or authenticated consumer information is published back to the registry as part of the transaction or other events. This information is used to correlate the consumer-specific run-time dependencies.

Creating a Consumer Application

Use the following procedure to create a consumer application asset.

To create a consumer application asset

1 In CentraSite Control, go to **Asset Catalog > Browse**.

2 Click **Add Asset(s)**.

3 In the Add Asset dialog, enter values for the following fields:

In this field...	Specify...
Type	The **Application** asset type.
Name	A name for the application asset. An asset name can contain any character (including spaces).
Description	*Optional*. A comment or descriptive information about the new application asset.
Organization	The organization to which the application asset belongs.

In this field...	Specify...

Initial Version	*Optional*. An identifier for the initial version of the application asset. The default is "1.0", but you can use any versioning scheme you choose. The version identifier does not need to be numeric.
	
	Examples:
	
	`0.0a`
	`1.0.0 (beta)`
	`Pre-release 001`
	`V1-2007.04.30`
	
	You can later create new versions of the application asset (see the section *Versioning an Asset* in the document *Using the Asset Catalog*).

4 Click **OK**.

Now the detail page of the newly created asset opens. Here you can enter the values of various attributes of the new asset.

5 Configure the profiles of the detail page, as described in the following sub-sections.

6 If you belong to a role that has the "Register as Consumer" permission, the entry **Register as Consumer** is enabled in the **Actions** menu of the details page. If you select this menu entry, a dialog opens that lets you select users, groups and consumer applications that can use this asset. The request must be subsequently approved or rejected by the owner of the asset.

7 Specify the application asset in the Consumers profile of the asset to be consumed. To do this, open the detail page of the asset to be consumed and specify the application asset the **Consumers** profile.

Configuring the Profiles of a Consumer Application

To configure the profiles of a consumer application, see one of the following sections as appropriate:

- Configuring the Identification Profile
- Configuring the Permissions Profile
- Configuring the Versions Profile
- Configuring the Subscriptions Profile
- Viewing the Audit Log Profile

- Configuring the Object-Specific Properties Profile

Configuring the Identification Profile

In this profile, specify the precise values for the consumer identifier(s) that you specified in the Identify Consumer action.

Notes:

1. If you specify *multiple* identifiers, the system evaluates them with the identifier defined in the Identify Consumer action.
2. If you want to authenticate consumers, make sure that your policy enforcement point is configured to enable authentication. For information, see the webMethods Mediator documentation or the documentation for your third-party PEP.

To configure the Identification profile

- Specify values for one or more of the following fields.

Note: The value(s) that you specify in the Identification profile depend on how the run-time policy's Identify Consumer is configured. For example, if Identify Consumer is configured to identify consumers by their IP address, you should specify the consumer IP addresses here. For information about this action, see the *Run-Time Governance Reference > Built-In Run-Time Actions Reference for Virtual Services*.

In this field...	Do the following...
IPv4 Address	Identify consumers based on their originating 4-byte IP address. Use this field when the Identify Consumer action is configured to identify consumer applications by IP address ■ To specify an individual IP address, type the address in the From field. The application asset will identify onl those requests that originate from the specified IP address. ■ To specify a range of IP addresses, type the lowest IP address in the From field and the highest IP address in t To field. For example, the values 192.168.0.0 and 192.168.0.10 indicates that requests originating from any IP address that lies between the specified range will be identified by the application asset. If you need to specify additional IP addresses, use the plus button to add more rows.
IPv6 Address	As for **IPv4 Address**, but using the 128-bit IPv6 format. For example: "1234:5678:9ABC:DEF0:1234:5678:9ABC:DEF
Identification Token	Identify consumers based on one or more of the following kinds of identification tokens: Use this field when the Identify Consumer action is configured to identify consumer applications by host name HTTP user name, WSS user name or a custom token.

In this field...	Do the following...
	■ Host Name—To identify consumers based on a specified host name, type the host name (for example pcmachine.ab.com) in the Name field. The application asset will identify only those requests that orig the specified host name. ■ HTTP Authentication Token—To authenticate consumers based on the user name that is transmitted i authentication user token, type the user name (for example, testuser123) in the Name field. The applic will identify only the requests that contain the specified user name encoded and passed in the HTTP auth user token. Authentication is handled by LDAP or another external authentication mechanism. You c the kinds of HTTP headers that webMethods Mediator will pass from requests to services. The defau Authorization header. To configure webMethods Mediator to pass other kinds of HTTP headers, see webMethods Mediator documentation. ■ WS-Security Authentication Token—To authenticate consumers based on the user name that is trans the SOAP or XML message header (HTTP body), type the user name (for example, userwss) in the N The application asset will identify only the requests that contain the specified user name passed in th XML message header. Authentication is handled by LDAP or another external authentication mecha ■ Custom identification token (XPath)—To identify consumers based on the result of applying an XPath e on the SOAP or XML message or request, enter the XPath expression in the Name field. For example //*[local-name()='Envelope']/*[local-name()='Body']/*[local-name()='echoInt']/*[local-name()='echoIntInp in the Name field will identify the requests that contain the XPath and the consumers. If you need to specify additional tokens, use the plus button to add more rows.
Consumer Certificate	Identify consumers based on information in an X.509 v3 certificate. Use this field when the Identify Consumer action is configured to identify consumer applications by a certificate. Click **Browse** to locate the certificate (.cer) file and select the certificate file.

Configuring the Permissions Profile

Use this profile to set permissions for the application asset. For information, see the section *Setting Permissions on an Asset* in *Using the Asset Catalog*.

Configuring the Versions Profile

Use this profile to generate a new version of the application asset. For information, see the section *Versioning an Asset* in *Using the Asset Catalog*.

Configuring the Subscriptions Profile

Optional. If you want to be notified when the application asset is changed, click the **Actions** button and select **Notify me** from the drop-down list. Your owner name will appear in the **Subscriptions** profile. Any other users who have permission to access the asset can add their own user names to this list.

Viewing the Audit Log Profile

This profile displays an audit log of the changes made to the application asset (including changes in an asset's lifecycle state).

Configuring the Object-Specific Properties Profile

Optional. Use this profile to add object-specific properties to the application asset.

To configure the Object-Specific Properties tab

1 Click the **Add Property** button.

2 Specify values for the following fields:

In this field...	Specify...
Name	The name of the property.
Namespace	*Optional* The namespace of the property.
Values	*Optional* A value for the property. If you want to specify multiple values, use the plus button to add additional rows.

Editing a Consumer Application

Use the following procedure to edit the attributes associated with a consumer application asset in the catalog.

When editing attributes, keep the following general points in mind:

- If you are not the owner of the asset, you cannot edit the asset unless you have Modify permission on the asset (granted though either a role-based permission or an instance-level permission).
- When you view the details for the asset, you will only see profiles for which you have View permission. You will only be able to edit the profiles on which you have Modify permission.

To edit the attributes for a consumer application asset

1 In CentraSite Control, go to **Asset Catalog > Browse**.

2 On the Browse page, perform a keyword or advanced search to display the application asset(s). For procedures, see the section *Searching the Asset Catalog* in *Using the Asset Catalog*.

3 Locate the application asset whose details you want to view and, from its context menu, select **Details**.

4 To edit the application asset's **Name**, **Description** or user-defined version number, place the cursor in the appropriate field and modify the text as required.

5 To modify the extended attributes associated with the asset, do the following:

1. Select the profile that contains the attribute(s) that you want to modify.
2. Edit the attributes on the profile as necessary.

Note: If at any time you want to abandon your unsaved edits, click **Close**. CentraSite will ask you if you want to save your edits. Click **No** to abandon your edits and return the asset's attributes to their previous settings.

6 When you have finished making your edits, click **Save**.

Deploying a Consumer Application

You can deploy consumer application assets to a policy enforcement point (such as webMethods Mediator) in either of the following ways:

- **From the Operations > Deployment page**

  You can deploy multiple consumer applications to a Mediator target in a single step (see *Deploying Consumer Applications*).

- **Running a script file from a command line**

  You can run a script file to deploy multiple consumer applications to a Mediator target in a single step (see *Deploying Consumer Applications*).

Deleting a Consumer Application

Before you delete an application asset, we strongly recommend that you examine the asset's **Impact Analysis** profile to determine whether other assets will be affected by the asset's deletion.

To delete an application asset

1 In CentraSite Control, display the detail page of the application asset that you want to delete. If you need procedures for this step, see the section *Viewing Details for an Asset* in *Using the Asset Catalog*.

2 In the **Actions** menu, click **Delete**.

You can delete multiple application assets in a single step. The rules described above for deleting a single application asset apply also when deleting multiple application assets.

Important: If you have selected several application assets where one or more of them are predefined application assets, you can use the **Delete** button to delete the assets. However, as you are not allowed to delete predefined application assets, only assets you have permission for will be deleted. The same applies to any other application assets for which you do not have the required permission.

To delete multiple application assets in a single operation

1 In CentraSite Control, use either the Browse or the Search feature in the asset catalog to select a list of the application assets. If you need information on how to browse or search the asset catalog, refer to the section *Browsing the Asset Catalog* or *Searching the Asset Catalog* in *Using the Asset Catalog*.

2 Mark the checkbox of each application asset you want to delete.

3 In the **Actions** menu, click **Delete**.

Consumer Provisioning and Consumer-Provider Relationship Tracking

The term *consumer provisioning* means providing users with the ability to consume assets. Consumer provisioning with CentraSite enables you to control and monitor who consumes assets.

A consumer can be any of the following:

- A registered user of CentraSite.
- A registered group of users.

- A guest user.
- A consumer who is identified in an **application asset**.

To control who consumes assets, you:

- **Register users to consume assets.**
  CentraSite users with the proper permissions can register themselves or other users as consumers of specified assets. That is, users can request permission to access specified assets in the registry. The owners of the assets may approve or reject such requests.
- **Specify the registered consumers in the asset's Consumers profile.**
  After users, groups and/or applications are approved to consume an asset, you must specify those consumers in the asset's Consumers profile. The Consumers profile appears in the asset's detail page.

Because consumers are registered, CentraSite can easily track consumer-provider relationships. The purpose of tracking consumer-provider relationships is to identify:

- The artifacts in the registry that will be affected if an asset is not available or must be changed.
- The organizations that need to be informed in such situations.

You can track consumer-provider relationships in the following ways:

- **View pending registrations.**
  If you are the owner of an asset, and another user has made a request to register as a consumer of the asset, you can view the request (see ***Viewing Consumer Registration Requests)***.
- **View your registration requests.**
  If you have made a request to register as a consumer of an asset owned by another user, you can view the status of the request (see ***Viewing Consumer Registration Requests***).
- **Generate reports on consumer-provider relationships.**
  See *Working with Reports and Report Templates*.
- **Inspect the "Impact Analysis" view of the assets.**
  You can view associations between the registry objects to identify the impact when updating or deleting an asset in the catalog. For more information, see the section *Impact Analysis* in *Using the Asset Catalog*.

This section covers the following topic:

- Implementing Consumer Provisioning

Implementing Consumer Provisioning

To provide consumers with access to assets, you perform the following tasks:

- Creating the Consumer-Provider Relationship Policy
- Registering Users to Consume Assets
- Viewing Consumer Registration Requests

Creating the Consumer-Provider Relationship Policy

To enable users to register as consumers of assets, you must first create a policy that enables the asset owners to approve or reject the "Register as Consumer" requests. To create the policy, use this procedure.

To create the Consumer-Provider Relationship Policy

- Create a Design/Change-Time policy with the following specifications:

Field	Value
Name	`Consumer-Provider Relationship Policy`
Object Type	**Assets**
Event Type	**OnConsumerRegistration**
Actions	■ **Initiate Approval** ■ **Register Consumer** ■ **Set Consumer Permission**

If you need instructions for creating a Design/Change-Time policy, see *Working with Design/Change-Time Policies*. For information about setting the action parameters, see the *Built-In Design/Change-Time Actions Reference*.

When you register users to consume assets (as described in ***Registering Users to Consume Assets***), the policy is triggered and the "Register as Consumer" request is submitted to all members of the approval list specified in the Initiate Approval action. Then, the approvers can either approve or decline the request. If the approvers approve the request, the consumers will be registered as consumers, and appropriate permissions will be assigned to users and groups (permissions are not applicable to application consumers).

Registering Users to Consume Assets

If you have permissions to view assets in the catalog, and you belong to a role that includes the "Register as Consumer" permission, the **Register as Consumer** feature is enabled in the **Actions** menu when you browse or search the asset catalog in CentraSite Control. This feature opens a dialog that lets you request the right to be a consumer of one or more of the displayed assets. You can request the right for yourself, or for any user or group in any organization, or for any consumer application owned by any organization.

The request must be subsequently approved or rejected for each asset by at least one of the owners of the asset. This functionality is also available in the detail view of an asset by choosing the **Register as Consumer** menu entry in the **Actions** menu. This functionality is not available to guest users.

To register users to consume assets

1 Ensure that you have created a design-time policy named `Consumer-Provider Relationship Policy`, as described in *Creating the Consumer-Provider Relationship Policy*.

2 In CentraSite Control, go to **Asset Catalog > Browse**.

3 Select the check box next to each of the required assets and click **Actions > Register as Consumer**.

4 In the **User/Group** field, use the **Browse** button to display a list of all users and groups from all organizations, and then select the one you want.

5 If you want to specify additional users or groups, use the plus button beside the **User/Group** field to create a new **User/Group** input field, then use the **Browse** button to select the required user or group as in the previous step.

6 In the **Application** field, choose an application asset from the selection box. The selection box shows Applications assets from all organizations.

7 If you want to specify additional application assets, use the plus button beside the **Application** field to create a new **Application** input field, and choose another application asset.

8 When you have specified all required users, groups and applications, click **OK**.

Requests to register the users, groups and/or applications are sent to the owner(s) of the assets.

9 The owner of each asset can either accept or decline a "Register as Consumer" request as follows:

1. Go to **Home > My CentraSite > Consumer Registrations** and click **Pending Registrations**.
2. To start the approval process for the request, select the check box next to the request and click **Apply Registration Policies**. This triggers the OnConsumerRegistration event, which in turn activates the Consumer-Provider Relationship Policy.
3. To decline the request, select the check box next to the request and click **Decline**. The Consumer-Provider Relationship Policy is not activated.

10 After the users, groups and/or applications are approved to consume an asset, you must specify those consumers in the asset's Consumers profile. The Consumers profile appears in the asset's detail page.

Viewing Consumer Registration Requests

To view a summary of all "Register as Consumer" requests, go to **Home > My CentraSite > Consumer Registrations** and use the following links:

- **Pending Registrations**: If you are the owner of an asset, and another user has made a request to register as a consumer of the asset, you can view the request here. As the asset owner, you can accept or decline the request.
- **Registration Requests**: If you have made a request to register as a consumer of an asset owned by another user, you can view the status of the request here.

13 Deploying Virtualized Services and Consumer Applications

This section describes how to deploy virtualized services and consumer applications to webMethods Mediator.

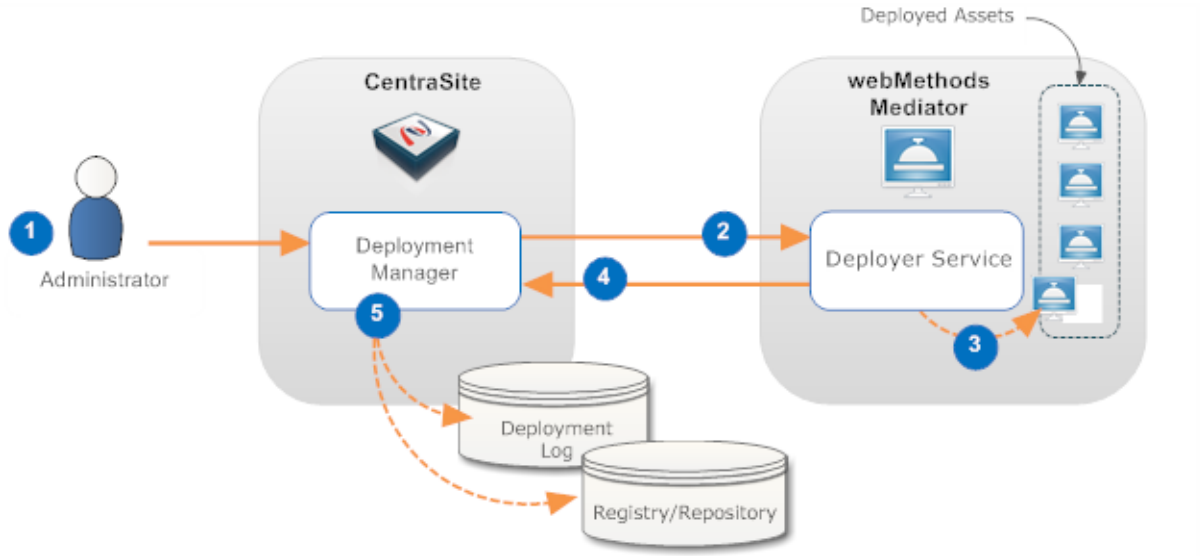
The Synchronous Deployment Model

Deployment refers to the process you use to send virtual services, virtual XML services, virtual REST services and consumer applications to the policy-enforcement points (PEPs) on which they are to be used for run-time governance. Instructions throughout the remainder of this guide use the term "virtualized service" when referring to all three virtual types in general.

Note: This section provides information about deploying virtualized services and consumer applications to webMethods Mediator. If you are using a different kind of PEP, refer to its documentation for deployment procedures and information.

As shown in the following diagram, the deployment process is initiated from CentraSite and is carried out by the "deployer" service on Mediator.

The Deployment Process



Step	Description
1	An administrator initiates the deployment by selecting the assets that are to be deployed and specifies to which Mediator they are to be deployed.
2	The Deployment Manager on CentraSite prepares the asset for deployment (the specific preparation steps depend on the type of asset being deployed) and invokes the deployer service on the Mediator. The prepared asset is submitted as input to this service.
3	The deployer service deploys the asset in the Mediator.

Step	Description
4	If the deployment is successful, the deployer service returns a success message and data that is pertinent to the deployed asset. If the deployment is unsuccessful, the deployer service returns a failure message.
5	The Deployment Manager on CentraSite logs information about the deployment in the Deployment log. If the deployer service returned specific data about the asset, the asset's metadata is updated as needed in the registry/repository.

For more information, see ***What Happens When You Deploy a Virtualized Service?***.

Who Can Deploy Virtualized Services and Consumer Applications?

To deploy virtualized services and consumer applications, you must belong to a role with the following permissions:

- Create Assets —OR— Manage Assets
- Manage Runtime Policies (required to configure virtualized services)
- Manage Lifecycle Models (required to change state of virtualized services)
- Manage Runtime Targets (required to deploy virtualized services)

For more information about roles and permissions, see the section *About Roles and Permissions* in *Users, Groups, Roles and Permissions*.

Note: Mediator exposes several Web service operations to allow CentraSite to manage deployed assets. This Web service is invoked by CentraSite any time a user deploys or undeploys a virtual service or consumer application to Mediator. A Mediator target's configuration details page includes the `User Name` and `Password` fields which identify an Integration Server user who is permitted to execute the Integration Server services associated with Mediator's deployer service. After installation, only members of the Integration Server "Administrators" group are allowed to invoke these services. However, administrators have the flexibility to allow their own users or groups to invoke them. Access to these services is controlled by an ACL, called `MediatorDeployer`. Initially, only the predefined "Administrators" group is assigned to this ACL. An Integration Server administrator can remove this group and add other groups or individual users. For example, you can create your own deployer group, (for example, "MyDeployers") and add Integration Server user IDs to this group. Then, the user must update the `MediatorDeployer` ACL by removing the "Administrators" group and adding the "MyDeployers" group. Now, on the target's configuration detail page, you can specify any user ID that belongs to "MyDeployers" group.

Conditions that Must be Satisfied for Effective Deployment of Virtualized Services

To deploy a virtualized service to a webMethods Mediator target, the following conditions must be met:

- Ensure that you have the "Manage Runtime Targets" permission. Only users with this permission can deploy a virtual service. CentraSite will not enable the deployment controls for any other users. For more information, see ***Who Can Deploy Virtualized Services and Consumer Applications?***.
- Ensure that the run-time policies for the virtualized service are active. This is indicated in the **Policies** profile on the virtualized service's detail page. If a policy is inactive, you must activate it as described in ***Activating a Run-Time Policy***.
- Ensure that the target to which the virtual service will be deployed has already been created, as described in the section *Run-Time Targets*.
- Ensure that the target's specified deployment URL is active and the user credentials of Integration Server are valid. To check this, go to the target's detail page and click the **Check Connection** button. If the connection is not active and valid, activate the deployment endpoint and modify the user credentials as required.
- If the virtualized service is under the control of an active lifecycle model (LCM), ensure that:
  - The virtualized service is in a "deployable" lifecycle state. If you are not certain of what the "deployable" lifecycle state is, consult your CentraSite administrator.
  - The virtualized service has a design-time policy that includes the Change Deployment Status action and it is set to Yes. This action specifies whether the service is eligible for deployment. For more information, see the description of the action in the section *Built-In Actions for Design/Change-Time Policies* in the *Built-In Design/Change-Time Actions Reference*.

If these conditions are not satisfied, all or part of the deployment user interface controls will be disabled when you view the virtual service.

What Happens When You Deploy a Virtualized Service?

When you deploy a virtualized service to one or more webMethods Mediator targets, keep the following points in mind.

Policy Validation

When you deploy a virtualized service, CentraSite automatically validates the service's run-time policy (or policies) to ensure that:

- Any action that appears in a single policy multiple times is allowed to appear multiple times.
- All action dependencies are properly met.

CentraSite will inform you of any violation, and you will need to correct the violations before deploying the service. For more information about dependencies and which actions can be included multiple times in a single policy, see the section *Run-Time Governance Reference* > *Built-In Run-Time Actions Reference for Virtual Services* .

What if You Need to Modify Deployed Assets?

If you need to modify a virtualized service or consumer application that is already deployed, you must modify it in CentraSite and then redeploy it to Mediator. Mediator does not monitor CentraSite for updates to deployed assets. If you make changes to a virtual service's processing steps, for example, you must manually redeploy the virtual service to put those changes into effect.

What if You Need to Modify the Run-Time Policy of a Virtualized Service?

You cannot make changes to a run-time policy while it is active. To make changes to a policy after it has been switched to the active state you must do one of the following:

1. Switch the policy to the Suspended state (to deactivate it), update the policy and then switch it back to the Productive state (to reactivate it).

   OR

2. Create a new version of the policy, make your changes to the new version of the policy and then switch the new version to the Productive state. Switching the new version of the policy to the Productive state will automatically Retire (and deactivate) the old version.

If you need to update a run-time policy that is already deployed with virtual services that are in production, always use the second method described above (i.e., create a new version of the policy). If you use the first method, which requires you to suspend the existing policy, your production services will be running without the policy while you are making your revisions to it.

Managing Endpoints of Virtualized Services

When you deploy a virtualized service to a Mediator target, CentraSite generates an XML document called a virtual service definition (VSD). The VSD defines the virtualized service for Mediator, and contains all the run-time policies and resources required to deploy the virtualized service to Mediator. In addition, CentraSite automatically updates the service's CentraSite endpoint to its Mediator endpoint. You can view the Mediator endpoint on the virtualized service's detail page in CentraSite. Because the endpoint information for virtualized services is generated and updated by CentraSite, you should not manually edit the endpoint information for virtualized services. In other words, unlike with native services, you should not manually add endpoints to a virtualized service. Instead, simply allow CentraSite to generate and manage the endpoints for the virtualized services that you deploy.

However, you can deploy multiple virtualized services for a single native service, in order to make the service available over multiple transports and/or security mechanisms. For details, see the section *Managing Endpoints* in *Implementation Concepts*.

What Happens When Deployment Fails?

If Mediator encounters a problem deploying or redeploying a virtualized service, it sets the service's Deployment Status to "Failed" and sends a message to CentraSite describing the problem. This failure is also logged to Mediator. In this case, it is up to the CentraSite or Mediator administrator to take corrective action and redeploy the service manually from CentraSite.

If the reason for the failure is that the Mediator instance is unavailable, and then you restart the Mediator instance, it loads all information about any previously deployed assets.

Deploying, Undeploying and Redeploying Virtualized Services

You can deploy, undeploy and redeploy virtualized services (of any type) to a webMethods Mediator target in any of the following ways:

- **From the virtualized service's detail page**

  You can deploy, undeploy and redeploy a virtualized service to one or more Mediator targets (see ***Deploying Virtualized Services from the Service's Detail Page***).

- **From the Operations > Deployment page**

  You can deploy, undeploy and redeploy *multiple* virtualized services to a Mediator target in a single step (see ***Deploying Virtualized Services from the Operations > Deployment Page***).

- **From the target's details page**

  You can deploy, undeploy and redeploy multiple virtualized services to a Mediator target in a single step (see ***Deploying Virtualized Services from the Target's Details Page***).

- **Using run-time commands**

  You can use the CentraSite Command facility to deploy, undeploy and redeploy virtualized services to a Mediator target individually or in bulk, and to synchronize consumers (see *Deploying Virtualized Services Using Run-Time Commands*.

- **Running a batch process**

  You can ran a batch process to deploy, undeploy and redeploy multiple virtualized services to a Mediator target in a single step (see *Deploying Virtualized Services Using a Batch Process*).

If You Migrate Virtualized Services from a Pre-8.2 Release

If you have virtualized services that were created prior to version 8.2, those virtualized services will continue to hold the deployment metadata generated by CentraSite 8.0. Although this metadata is not applicable in CentraSite 8.2, and will not affect deployment in 8.2, we strongly recommend that you perform the following steps:

1. Undeploy all virtualized services from CentraSite 8.0.
2. Upgrade to CentraSite 8.2.
3. Ensure that all target deployment URLs are configured correctly.
4. Deploy all virtualized services from CentraSite 8.2.

Note: Please be aware that the new synchronous deployment model does not support subscriptions and subscription services.

Deploying Virtualized Services from a Service's Detail Page

The following procedure describes how to deploy, undeploy and redeploy a single virtualized service to one or more Mediator targets, using the service's **Deployment** profile.

To deploy, undeploy or redeploy a virtualized service

1 In CentraSite Control, display the virtualized service detail page. For procedures, see the section *Viewing Details for an Asset* in *Using the Asset Catalog*.

2 Select the virtualized service's **Policies** profile and ensure that all of the service's run-time policies are Active. If not, activate them as described in *Activating a Run-Time Policy*.

3 Ensure that the virtualized service has a design-time policy that includes the Change Deployment Status action and it is set to Yes. This action specifies whether the service is eligible for deployment. For more information, see the description of this action in the *Built-In Design/Change-Time Reference*.

4 Make sure you switch the service to an active, ready-to-deploy state, as follows. (If you do not know which state to select, you will need to examine your organization's lifecycle model for Service objects or consult an administrator.)

1. In the **Actions** menu, select the **Change Lifecycle State** option.
2. Select the lifecycle state to which you want to switch the asset and click **OK**. (The list will contain only the states that you are permitted to assign to the service.)

   If the state change requires approval, CentraSite will initiate an approval workflow and your request for a state change will be submitted to the appropriate approvers. While the request is awaiting approval, the service will appear in a pending state.

5 After the lifecycle state has been successfully changed, select the virtualized service's **Deployment** profile.

The **Deployment** profile will display the following information.

Column	Description
Target	The target on which the service is deployed.
Target Type	The type of target on which the service is deployed (e.g., webMethods ESB (Integration Server) or Insight).
Description	Description of the target.
Deployment Status	The deployment status of the service (e.g., Deployed or Failed).
Date Deployed	The date/time that the deployment occurred.

6 Click the **Deploy** button, select the Mediator target(s) to which you want to deploy the virtualized service, and click **OK**.

7 The deployment process is carried out by a synchronous mechanism between the CentraSite and the Mediator:

1. CentraSite pushes the virtualized service that is ready for deployment to the webMethods Mediator target.
2. Instantly, the Mediator deploys the virtualized service that was received from CentraSite (along with its effective run-time policy), and notifies CentraSite when the deployment process is complete.

For more information, see *What Happens When You Deploy a Virtualized Service?*.

⚠ **Important:** If the status shown in the **Deployment Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to deploy the virtualized service successfully. If the deployment process failed, identify and correct the error and then try deploying the virtualized service again.

8 To undeploy the virtualized services, select the services' check boxes, and choose **Undeploy** from the Actions menu.

If the undeployment is successful, Mediator's deployer service returns a success message, and data that is pertinent to the undeployed virtual service. In addition, CentraSite's Deployment Manager logs information about the undeployment in the Deployment log. If the undeployment is unsuccessful, the deployer service returns a failure message.

Important: If the status shown in the **Deployment Status** column does not automatically switch to **Undeployed**, click the **Refresh** button to determine whether CentraSite was able to undeploy the virtualized services successfully. If the undeployment process failed, identify and correct the error and then try undeploying the virtualized services again.

9 To redeploy the virtualized service, select the services' check boxes, and choose **Redeploy** from the Actions menu.

Important: If the status shown in the **Deployment Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to redeploy the virtualized services successfully. If the redeployment process failed, identify and correct the error and then try redeploying the virtualized services again.

10 Examine the deployment log that is displayed by CentraSite Control and check for any errors that occurred during the deployment process.

Deploying Virtualized Services from the Operations > Deployment Page

The following procedure describes how to deploy, undeploy and redeploy multiple virtualized services to a Mediator target in a single step.

To deploy, undeploy or redeploy a virtualized service

1 In CentraSite Control, go to **Operations > Deployment**.

2 On the **Deployed Assets** tab, click **Deploy**.

3 In the **Select a Target and Services to be Deployed on the Selected Target** dialog box, perform a keyword or advanced search to display the virtualized services that are ready for deployment.

- If you want to perform a keyword search and you need procedures for this step, see **Keyword Search**.
- If you want to perform an advanced search and you need procedures for this step, see **Advanced Search**

4 Click **OK**.

5 The deployment process is carried out by a synchronous mechanism between the CentraSite and the Mediator:

1. CentraSite pushes the virtualized service that is ready for deployment to the webMethods Mediator target.
2. Instantly, the Mediator deploys the virtualized service that was received from CentraSite (along with its effective run-time policy), and notifies CentraSite when the deployment process is complete.

For more information, see *What Happens When You Deploy a Virtualized Service?*.

The **Deployed Assets** tab will display the following information.

Column	Description	
Pending Changes	Icons indicating the deployment status of the virtualized services.	
	Icon	**Description**
		Service is deployed to the target.
		Service is pending deployment to the target.
Deployment ID	The deployment ID of the virtualized service.	
Name	Name of the virtualized service.	
Status	The deployment status of the virtualized service (e.g., Deployed or Failed).	
Date/Time	The date/time that the deployment occurred.	
User ID	The user-ID of the Integration Server user to be used for the deployment operation.	
Type	Modification that is been performed on the deployed service.	
	Label	**Description**
	Processing Step Changes	Reflects any change that is performed in the virtualized service's **Processing Steps** profile. For example, modifying the HTTP authentication mode.
	Runtime Policy Changes	Reflects any change that is performed in the runtime policy associated to the virtualized service. For example, deactivating an associated runtime policy.
	WSDL Changes	Reflects any change that is performed in the virtualized service's asset file. For example, modifying an existing asset file or uploading a new asset file.

Important: If the status shown in the **Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to deploy the virtualized services successfully. If the deployment process failed, identify and correct the error and then try deploying the virtualized services again.

6 To undeploy the virtualized services, select the services' check boxes, and choose **Undeploy** from the Actions menu.

> **Important:** If the status shown in the **Status** column does not automatically switch to **Undeployed**, click the **Refresh** button to determine whether CentraSite was able to undeploy the virtualized services successfully. If the undeployment process failed, identify and correct the error and then try undeploying the virtualized services again.

7 To redeploy the virtualized service, select the services' check boxes, and choose **Redeploy** from the Actions menu.

> **Important:** If the status shown in the **Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to re-deploy the virtualized services successfully. If the redeployment process failed, identify and correct the error and then try redeploying the virtualized services again.

8 Examine the deployment log that is displayed by CentraSite Control and check for any errors that occurred during the deployment process.

Selecting a Target and Services to be Deployed on the Selected Target

The following procedure describes how to use CentraSite's Search feature to select target and services to be deployed on the selected target using keyword searches and advanced searches.

- Keyword Search
- Advanced Search

Keyword Search

The keyword search is an easy to use search facility in which you can specify arbitrary search patterns.

You can search for all virtualized services that contain one or more specified keywords (i.e., text strings) in the virtualized service's string attributes (virtualized service name, description, etc.).

Conventions for Keyword Searches

The conventions for keyword searches are as follows:

- A keyword search consists of 1-n search keywords. Multiple keywords are space separated. If multiple keywords are given, a logical disjunction (OR) is implied.
- A keyword is treated as partial text which can occur at the beginning of the searched strings. The "starts with" semantics are implied.

  Example: If the keyword is "AustralianPostCode", then the following matches are returned: "A sample VS for AustralianPostCode" as well as "AustralianPostCodeVService".
- If quotes (" ") exist around a phrase, then a search is performed on the exact phrase within the quotes. A space within a quoted phrase is considered as a space character and not as a logical

operation. To force the keyword search to treat the quote characters as a normal character, precede the quote character with a backslash (\). If you want to include the backslash character itself in the search, type two backslashes.

- You can mix and match any number of words and quoted phrases within the keyword field.
- The search is neither case nor accent sensitive, even within a quoted phrase. Example: A search for "AustralianPostCode" will return the same results as a search for "AUSTRALIANPOSTCODE" or "Australianpostcode".
- If you enter a string that contains an odd number of double-quote characters, then the last double-quote character is ignored when the search is performed.
- If the keyword search input field is empty when the search is executed, the search returns all available virtualized services.
- The keyword search can include wildcard characters.

Wildcard Characters

The available wildcard characters are:

Character	Usage
* or %	If you use the percent symbol ("%") or the asterisk ("*"), CentraSite replaces the wildcard symbol with as many characters as necessary to find a match. For example, an entry of "A%n" returns both "Amazon" and "American". If you enter "*al", then "CalcService", "Calendar" and "AustralianPostCode" all fit the search criteria.
? or _	If you use the question mark ("?") or the underscore ("_"), CentraSite replaces the wildcard symbol with a single character in order to find a match. Example: "AustralianPostCode?VService" matches any character for "?".

You can use a wildcard character at any point in the keyword text, and multiple times throughout the keyword text. If you enter a wildcard character in the middle of a string, for example "cat*dog", then at least one of the searched attributes must contain the string in order for the asset or supporting document to be included in the result set.

If a wildcard character between two words is surrounded by spaces, such as "word1 * word2", the wildcard will match one word.

Notes:

1. Certain non-alphanumeric characters that can appear in the name of a virtualized service are currently ignored by CentraSite's wildcard mechanism when you include them in a keyword search. In particular, the hyphen ("-") is ignored. Thus, if you have created the virtualized services "AustralianPostCodeVService-1" and "AustralianPostCodeVService_1", the wildcard search for "AustralianPostCodeVService?1" will find "AustralianPostCodeVService_1" but not "AustralianPostCodeVService-1".

2. The percent (%) character acts as a word delimiter when it appears in the text to be searched. Thus, for example, if the description field of a virtualized service contains the text "abc%def" (the characters a, b, c, %, d, e, f), this is treated by the search mechanism as two adjacent words "abc" and "def". A wildcard search such as "abc*def" looks for a single word beginning with "abc" and ending with "def", so the search will not find this asset.

Performing a Keyword Search

To search by keyword

1 In CentraSite Control, go to **Operations > Deployment**.

2 On the **Deployed Assets** tab, click **Deploy**. This opens up the **Select a Target and Services to be Deployed on the Selected Target** dialog.

3 In the text box, type the keyword(s) to search for. You can use one or more wildcards to specify the keywords.

If you leave the text box blank, or enter just a wildcard, the entire set of virtualized services is returned.

CentraSite returns the virtualized services that match the search criteria. The search looks for the keyword(s) in the virtualized service's name, type and description attributes.

Advanced Search

CentraSite's advanced search capabilities allow you to search for virtualized services on the basis of service types and targets.

Performing an Advanced Search

To search using service type and target

1 In CentraSite Control, go to **Operations > Deployment**.

2 On the **Deployed Assets** tab, click **Deploy**.

3 In the **Select a Target and Services to be Deployed on the Selected Target** dialog box, do the following:

1. In the field **Browse By**, select a virtualized service type. As a result, only virtualized services that are classified with this service type will be displayed.
   - If you do not specify a virtualized service type in the field **Browse by**, CentraSite Control displays a list of all virtualized services belonging to the CentraSite.
   - If you specify a virtualized service type in the field **Browse by**, CentraSite Control displays a list of all virtualized services that belong to the specified service type.

There are several generic entries in the drop-down list for the **Browse by** field. These are:

- **[All]**
  This lists all virtualized services that are available in a deployable state.
- **[Virtual Service]**
  This lists all virtual Web services that are available in a deployable state.
- **[Virtual REST Service]**
  This lists all virtual REST services that are available in a deployable state.
- **[Virtual XML Service]**
  This lists all virtual XML services that are available in a deployable state.

2. In the field **Target**, select a target for deploying the selected services.
3. Select the virtualized services that you want to deploy on the selected target.

4 Click **OK**.

Deploying Virtualized Services from the Target's Detail Page

The following procedure describes how to deploy, undeploy and redeploy multiple virtualized services to a Mediator target, using that target's details page.

Important: Ensure that the target's specified deployment URL is active and the user credentials of Integration Server are valid. To check this, go to the target's detail page and click the **Check Connection** button. If the connection is not active and valid, activate the deployment endpoint and modify the user credentials as required.

To deploy, undeploy and redeploy services to a target

1 On the **Target Details** page, select the **Services** profile.

2 Expand the plus button next to the **Name** column.

The **Services** profile will display the following information.

Column	Description
Name	All services that are deployed to the target (or are pending deployment or have failed deployment), as well as all design/change-time policies associated with each service. Clicking the hyperlinked name displays the service's detail page.
Description	Descriptions of the services.
Deployment Status	The deployment status of the services (e.g., Deployed or Failed).
Icons next to Name	Icons indicating the deployment status of the virtualized services.

Column	Description	
	Icon	**Description**
		Service is deployed to the target.
		Service deployment failed in the target.

3 Click the **Deploy** button. A pop-up displays all services that are eligible to be deployed to the target.

4 Select one or more services and click **OK**.

The deployment process is carried out by a synchronous mechanism between the CentraSite and the Mediator:

1. CentraSite pushes the virtualized service that is ready for deployment to the webMethods Mediator target.
2. Instantly, the Mediator deploys the virtualized service that was received from CentraSite (along with its effective run-time policy), and notifies CentraSite when the deployment process is complete.

For more information, see *What Happens When You Deploy a Virtualized Service?*.

5 To undeploy the virtualized services, select the services' check boxes, and choose **Undeploy** from the Actions menu.

Important: If the status shown in the **Status** column does not automatically switch to **Undeployed**, click the **Refresh** button to determine whether CentraSite was able to undeploy the virtualized services successfully. If the undeployment process failed, identify and correct the error and then try undeploying the virtualized services again.

6 To redeploy the virtualized service, select the services' check boxes, and choose **Redeploy** from the Actions menu.

Important: If the status shown in the **Status** column does not automatically switch to **Deployed**, click the **Refresh** button to determine whether CentraSite was able to re-deploy the virtualized services successfully. If the redeployment process failed, identify and correct the error and then try redeploying the virtualized services again.

Deploying Virtualized Services Using Run-Time Commands

You can perform the following deployment tasks by executing commands in the command line interface *CentraSiteCommand.cmd* (Windows) or *CentraSiteCommand.sh* (UNIX) of Command Central. The tool is located in `<CentraSiteInstallDir>`/utilities.

- Deploy or undeploy virtual services to a Mediator target.
- Synchronize consumers.
- To update and redeploy the virtual services' WSDLs without having to first undeploy them
- Perform a "bulk deploy", a "bulk undeploy", a "bulk redeploy", and a "bulk clean redeploy".

If you start the command line tool with no parameters, you receive a help text summarizing the required input parameters.

The parameters of the command are case-sensitive, so for example the parameter "-url" must be specified as shown and not as "-URL".

To deploy virtualized services to a Mediator target using a run-time command

- Use a command of the following format:

```
CentraSiteCommand deploy [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD> -virtualService <VIRTUAL-SERVICE> -target <TARGET>
```

The following table describes the complete set of input parameters that you can use with the `deploy` utility:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user`.
`-virtualService`	The name of the virtual service.
`-target`	The target to which the virtual service identified by the parameter `-virtualService` is to be deployed.

For example:

```
CentraSiteCommand deploy [-url "http://localhost:53307/CentraSite/CentraSite"] ↩
-user "Administrator" -password
"manage" -virtualService "VS1" -target "Target1"
```

To undeploy virtualized services to a Mediator target using a run-time command

- Use a command of the following format:

```
CentraSiteCommand undeploy [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD> -virtualService <VIRTUAL-SERVICE> -target <TARGET>
```

The following table describes the complete set of input parameters that you can use with the `undeploy` utility:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSitee) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user`.
`-virtualService`	The name of the virtual service.
`-target`	The target to which the virtual service identified by the parameter `-virtualService` is to be undeployed.

For example:

```
CentraSiteCommand  undeploy [-url "http://localhost:53307/CentraSite/CentraSite"] ↩
-user "Administrator" -password
"manage" -virtualService "VS1" -target "Target1"
```

To synchronize consumers using a run-time command

- Use a command of the following format:

```
CentraSiteCommand sync consumers [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD>  -target <TARGET>
```

The following table describes the complete set of input parameters that you can use with the `sync consumers` utility:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user`.
`-target`	The target on which to synchronize the consumers.

For example:

```
CentraSiteCommand  sync consumers [-url ↩
"http://localhost:53307/CentraSite/CentraSite"] -user "Administrator" -password
"manage" -target "Target1"
```

To update and redeploy the virtual services' WSDLs without having to first undeploy them

- Use a command of the following format:

```
CentraSiteCommand updatewsdl [-url <CENTRASITE-URL>] -user <USER-ID> ↩
-password <PASSWORD> -virtualService <VIRTUAL-SERVICE> -wsdlURI <WSDL-URI>
[-reuse] <TRUE | FALSE> ↩
[-wsdlAuthenticationUsername <WSDL-AUTHENTICATION_USERNAME>]
[-wsdlAuthenticationPassword <WSDL-AUTHENTICATION_PASSWORD>]
```

The following table describes the input parameters:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user`.
`-virtualService`	The name of the virtual service.
`-wsdlURI ↩`	Absolute path to the WSDL file location. All XSDs are expected to be present relative to the WSDL location. This argument can also be a remote URL location.

Input Parameter	Description
`-reuse`	Optional. Reuse the XSD parameters even if they are missing. Default is false.
`-wsdlAuthenticationUsername`	Optional. The username if the WSDL file needs authentication.
`-wsdlAuthenticationPassword`	Optional. The password if the WSDL file needs authentication.

To "bulk deploy", "bulk undeploy", "bulk redeploy" and "bulk clean redeploy" virtual services using a run-time command

- Use a command of the following format:

```
CentraSiteCommand bulk deploy [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD>  -target <TARGET>
```

```
CentraSiteCommand bulk undeploy [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD>  -target <TARGET>
```

```
CentraSiteCommand bulk redeploy [-url <CENTRASITE-URL>] -user <USER-ID> -password
<PASSWORD>  -target <TARGET>
```

```
CentraSiteCommand bulk cleanredeploy [-url <CENTRASITE-URL>] -user <USER-ID> ↩
-password
<PASSWORD>  -target <TARGET>
```

The following table describes the complete set of input parameters that you can use with these utilities:

Input Parameter	Description
`-url`	The fully qualified URL (http://localhost:53307/CentraSite/CentraSite) for the CentraSite registry/repository.
`-user`	The user ID of a user who has the "CentraSite Administrator" role.
`-password`	The password of the user identified by the parameter `-user`.
`-target`	The target to which the virtual service is to be deployed/undeployed/redeployed.

For example:

```
CentraSiteCommand bulk deploy [-url ↩
"http://localhost:53307/CentraSite/CentraSite"] -user "Administrator" -password
"manage" -target "Target1"
```

Deploying Virtualized Services Using a Batch Process

Use *Runtime.deployment.Deployer* when you do not have access to a browser or graphical user interface environment, and you want to perform deployment tasks. You can also use *Runtime.deployment.Deployer* when you want to automate deployment tasks through batch processes.

An automated deployment through a batch mode can be initiated by configuring the *Deployment-Configuration.properties* file located in the URL *http://<host>:53307/CentraSite/CentraSite/ino:dav/ino:dav/projects/CentraSite/configuration*.

Specifying a Deploy Batch Size

BatchSize is the maximum number of virtualized services to be pushed to a webMethods Mediator target before a syncpoint is taken. The default BatchSize is 50. To improve performance, you can set a BatchSize to define the maximum number of virtualized services to be pushed between two syncpoints using the property line:

```
com.softwareag.centrasite.runtime.deployment.DeployBatchSize=50
```

The BatchSize property can be set at any time. If a bulk deployment is already in progress, the current batch is sized according to the previous batch size. Subsequent batches use the new size. Suppose if the BatchSize is set to zero and changed while a deploy operation is already in progress, that operation loads the data as a single batch. Any subsequent deploy operations on the same CentraSite Control use the new BatchSize.

Specifying a Transaction Timeout

TransactionTimeout specifies the maximum time, in milliseconds, allowed for deployment operations (deployment, undeployment and consumer sync) that were pushed to a webMethods Mediator target to respond. Any such operations that do not respond before this timeout occurs are rolled back. The default TransactionTimeout is 6000 (ms). To improve performance, you can set a TransactionTimeout to define the maximum time for the deployment operations to respond using the property line:

```
com.softwareag.centrasite.runtime.deployment.TransactionTimeout=60000
```

For example, if a deployment operation attempts to set a transaction timeout of 360 seconds, and the TransactionTimeout setting is 300 seconds, the TransactionTimeout setting of 300 seconds is used. After the TransactionTimeout of 300 seconds the deployment operations roll back.

Note: If set to 0, the transaction will not time out.

Deploying, Undeploying and Redeploying Consumer Applications

You can deploy consumer application asset(s) to a policy enforcement point (PEP) such as webMethods Mediator in any of the following ways:

- **From the Operations > Deployment page**

  You can deploy multiple consumer applications to a Mediator target in a single step (see *Deploying Consumer Applications from the Operations > Deployment Page*).

- **Running a script file from a command line**

  You can run a script file to deploy multiple consumer applications to a Mediator target in a single step (see *Deploying Consumer Applications Using a Script File*).

Deploying Consumer Applications from the Operations > Deployment Page

The following procedure describes how to deploy multiple consumer application assets to a Mediator target in a single step.

To deploy consumer applications

1 In CentraSite Control, go to **Operations > Deployment**.

2 On the **Deploy Consumers** tab, click the **Synchronize** button.

3 In the **Select a Target and Consumer Applications to be Deployed** dialog box, perform a keyword or advanced search to display the consumer applications and the targets that are ready for deployment.

   - If you want to perform a keyword search and you need procedures for this step, see **Using Keyword Search**.
   - If you want to perform an advanced search and you need procedures for this step, see **Using Advanced Search**

4 Click **OK**.

5 The deployment process is carried out by a synchronous mechanism between the CentraSite and the Mediator:

   1. CentraSite invokes the Mediator's deployer service and pushes the consumer applications that are ready for deployment to the Mediator target.
   2. Instantly, Mediator deploys the consumer applications that were received from CentraSite and notifies CentraSite when the deployment process is complete.

   The **Deploy Consumers** tab will display the following information.

Column	Description	
Pending Changes	Icons indicating the deployment status of the consumer applications.	
	Icon	**Description**
		The consumer application is deployed to the target.
		The consumer application is pending deployment to the target.
Rule ID	The synchronization rule ID of the target (that is, webMethods Mediator or Insight).	
Target	The name of the target on which the consumer application is deployed.	
Last Sync Date	The date/time that the deployment occurred.	
Status	The deployment status of the consumer application (e.g., Deployed or Failed).	

Important: If the status shown in the **Status** column does not automatically switch to **Deployed** or **Failed**, click the **Refresh** button to determine whether CentraSite was able to deploy the virtualized services successfully. If the deployment process failed, identify and correct the error and then try deploying the virtualized services again.

Deploying Consumer Applications Using a Script File

The deployment operation of a consumer application invoked between CentraSite and the webMethods Mediator target using command line utility, includes a `main()` method, which allows it to be called from a Windows batch file or from a UNIX shell script.

- Creating a Script File to Invoke the Deployment Operation (for Windows)
- Creating a Script File to Invoke the Deployment Operation (for UNIX)

Creating a Script File to Invoke the Deployment Operation (for Windows)

Create a script file that looks as follows if CentraSite is running under Windows.

```
@echo off
set JAVAEXE=fullPathToJava.exe
set REDIST=CentraSiteHomeDirectory\redist
set BASEDIR=%~dp0
cd /d %REDIST%

REM build CLASSPATH with all files from jar directory
set LOCAL_CLASSPATH=
for %%I in (".\*.jar") do call "CentraSiteHomeDirectory\bin\cfg\lcp.cmd" %%I

%JAVAEXE% -cp %LOCAL_CLASSPATH% deployerClassName %*
cd /d %BASEDIR%
```

Where *deployerClassName* is the name of the deployer that you want to run.

Example

The following is an example of a script file that calls the Consumer Applications deployer:

```
@echo off
REM
REM Run Consumer Applications deployer
REM
set JAVAEXE=D:\software\java\jdk1.5.0_12\bin\java
set REDIST=C:\SoftwareAG\CentraSite\redist
set BASEDIR=%~dp0
cd /d %REDIST%

REM build CLASSPATH with all files from jar directory
set LOCAL_CLASSPATH=
for %%I in (".\*.jar") do call "C:\SoftwareAG\CentraSite\bin\cfg\lcp.cmd" %%I

%JAVAEXE% -cp %LOCAL_CLASSPATH% ↩
com.softwareag.centrasite.runtime.pep.util.VirtualServiceDeployer %*
cd /d %BASEDIR%
```

Creating a Script File to Invoke the Deployment Operation (for UNIX)

Create a script file that looks as follows if CentraSite is running under Unix.

```
set javaexe="fullPathToJava.exe"
set redist="CentraSiteHomeDirectory/redist"
set mainjar="CentraSiteRuntimePEP.jar"
set delim='\:'
cd "$redist"
set cl=""
foreach j ( 'ls *.jar' )
  if ($cl != "") set cl=${cl}${delim}
  set cl=${cl}${j}
end
setenv CLASSPATH ${mainjar}${delim}${cl}
$javaexe deployerClassName $*
```

Where *deployerClassName* is the name of the deployer that you want to run.

Example

The following is an example of a script file that calls the Consumer Applications deployer:

```
#!/bin/csh
#
# Run Consumer Applications deployer
#
set javaexe="/mydir/softwareag/cjp/v16/bin/java"
set redist="/mydir/softwareag/CentraSite/redist"
set mainjar="CentraSiteRuntimePEP.jar"
set delim='\:'
# build CLASSPATH with all files from jar directory
cd "$redist"
set cl=""
foreach j ( `ls *.jar` )
  if ($cl != "") set cl=${cl}${delim}
  set cl=${cl}${j}
end
setenv CLASSPATH ${mainjar}${delim}${cl}
$javaexe com.softwareag.centrasite.runtime.pep.util.VirtualServiceDeployer $*
```

Viewing the Deployment History Log

The Deployment History Log contains information about the virtualized services that CentraSite has pushed to the webMethods Mediator target for deployment.

To view the Deployment History Log, you must belong to a role that includes the "Manage Runtime Targets" permission. To see the list of predefined roles that include this permission, see *About Roles and Permissions* in the section *Users, Groups, Roles and Permissions*.

The following procedure describes how to view the deployment history log. To view this log, you must belong to a role that includes the "Manage Runtime Targets" permission.

- If you belong to one of the following roles, you can view all entries in the deployment history log.
  - CentraSite Administrator
  - Organization Administrator
  - Operations Administrator
- If you belong to the Organization Administrator role for an organization, you can view the deployment history log entries for virtualized services that were created by users in your organization.
- If you do not belong to either of these roles, but you have the "Manage Runtime Targets" permission, you can view the deployment history log entries for virtualized services that you created.

To view the Deployment History Log

1 In CentraSite Control, go to **Operations > Deployment**.

2 Click the **Deployment History** tab to view the list of all virtualized services that are sent to the webMethods Mediator target.

3 If you want to filter the list, type a partial string in the **Search** field. CentraSite applies the filter to the **Name** column. The **Search** field is a type-ahead field, so as soon as you enter any characters, the display will be updated to show only those virtualized services whose name contains the specified characters. The wildcard character "%" is supported.

The **Deployment History** tab provides the following information about each virtualized service.

Column	Description
Rule ID	The synchronization rule-ID that CentraSite automatically generates up on creation of the webMethods Mediator Target in CentraSite Control.
Name	The name assigned to the virtualized service.
Type	The type of the virtualized service (say, Web Service, XML Service or REST Service).
Version	The user-assigned version identifier for the virtualized service.
Target Name	The name of the target on which the virtualized service is deployed.
Action	The deployment action of the virtualized service on the webMethods Mediator Target (e.g., Deployed, Undeployed).
Status	The deployment status of the virtualized service on the webMethods Mediator Target (e.g., Success, Failed).
Date	The date/time that the virtualized service has deployed, redeployed or undeployed.

4 To view details of a particular deployment workflow, click the hyperlinked value in the **Name** column.

Deleting a Deployment Activity Log

When you delete an activity log, keep the following points in mind:

- To delete an activity log, you must belong to one of the following roles:
  - CentraSite Administrator
  - Organization Administrator
  - Operations Administrator
- Remember deleting an activity log via the **Deployment History** tab does not affect the deployment status of a virtualized service. However, deleting an activity log via the **Deployed Assets** tab will undeploy a virtualized service from the Mediator target and the status automatically switches to **Undeployed**.
- The **Deployment History** tab contains log entries for every deployment operation (such as Deployed, Undeployed etc.,) of a virtualized service. However, the **Deployed Assets** tab contains

log entry only for a Deployed operation of the virtualized service. Once you undeploy a virtualized service, the log entry will automatically get removed from the **Deployed Assets** tab.

- Be aware that you can never delete the last Deployed activity log of a virtualized service or consumer application via the **Deployment History** tab.

You can delete a deployment activity log in any of the following ways:

- Using the **Deployed Assets** tab on the Deployment page as described in *Performing a Delete Operation Using the Deployed Assets Tab*. This procedure enables you to delete the activity logs of virtualized services in the CentraSite Control.
- Using the **Deployment History** tab on the Deployment page as described in *Performing a Delete Operation Using the Deployment History Tab*. This procedure enables you to delete the activity logs of virtualized services and consumer applications in the CentraSite Control.

Performing a Delete Operation Using the Deployed Assets Tab

You delete the "deployed" and "redeployed" activity logs of virtualized services via the **Deployed Assets** tab.

You can delete multiple activity logs in a single step.

To delete one or more activity logs

1 In CentraSite Control, go to **Operations -> Deployment**.

2 On the **Deployed Assets** tab, mark the checkbox next to the name of each activity log that you want to delete. (You can select multiple logs.)

3 In the **Actions** menu, click **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

Each selected log is permanently removed from the CentraSite registry/repository. The activity logs in the virtualized service's **Deployment** profile and target's **Service** profile are not affected.

Performing a Delete Operation Using the Deployment History Tab

You delete the "deployed", "undeployed", "redeployed" and "failed" activity logs of virtualized services and consumer applications via the **Deployment History** tab.

You can delete multiple activity logs in a single step.

To delete one or more activity logs

1 In CentraSite Control, go to **Operations -> Deployment**.

2 On the **Deployment History** tab, mark the checkbox next to the name of each activity log that you want to delete. (You can select multiple logs.)

3 Choose **Delete**.

When you are prompted to confirm the delete operation, click **OK**.

Each selected log is permanently removed from the CentraSite registry/repository. The activity logs in the asset's **Deployment** profile and target's **Service** profile are not affected.

Securing Communications with CentraSite for Synchronous Deployment

An administrator can configure CentraSite to use either of the following kinds of authentication:

- HTTP Basic authentication
- HTTPS (Secure Sockets Layer (SSL)) authentication

Configuring CentraSite to use SSL authentication provides secure communications for the deployment.

This section explains how SSL works with CentraSite (which acts as the client) and Mediator (which acts as the server). This section also provides the information you need to configure both the client and server sides for SSL authentication.

Anatomy of a CentraSite SSL Connection

It is useful to conceptualize a CentraSite SSL connection in terms of a SSL client and a SSL server. The request for an SSL connection originates from a client.

During the SSL handshake process, the webMethods Mediator acting as the SSL server responds to the request for a connection by presenting its SSL credentials (an X.509 certificate) to the requesting CentraSite client. If those credentials are authenticated by the CentraSite client, either:

- An SSL connection is established and information can be exchanged between the CentraSite and webMethods Mediator.

—OR—

- The next phase of the authentication process occurs, and the webMethods Mediator requests the SSL credentials of the CentraSite. If the webMethods Mediator verifies those credentials (that is, the client's identity), an SSL connection is established and information exchange can take place.

CentraSite and SSL Connection Type

The types of SSL connection referred to above are termed *one-way* and *two-way* SSL authentication:

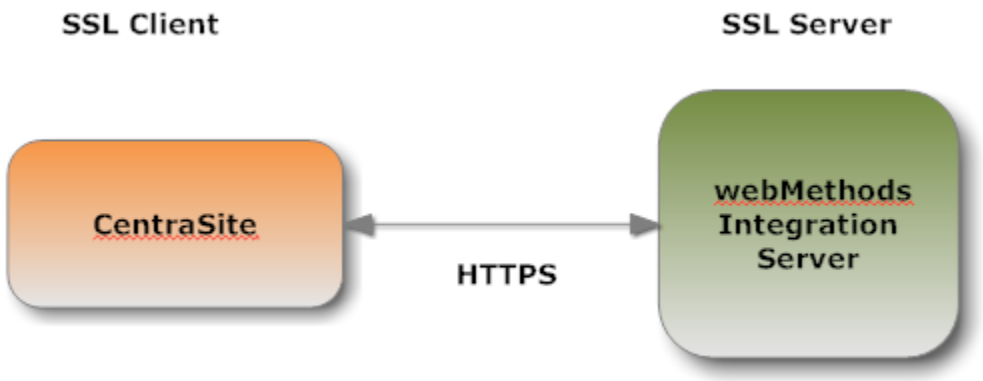
- In a *one-way* SSL connection, client authenticates the credentials of server in preparation for setting up a secure transaction. In most cases, the server knows nothing about the client's identity because verification of its credentials is not required. When desired, however the client can be authenticated by means such as basic username/password.

  This type of connection is typically one where CentraSite needs to verify the authenticity of the webMethods Mediator without itself needing to be authenticated. As a result, configurations on the CentraSite side are not actually required for this one-way connection.
- In a *two-way* SSL connection, both client and server must authenticate each other's credentials before an SSL connection is established and information can be exchanged.

  Unlike a one-way SSL connection, both CentraSite and the webMethods Mediator require access to each other's SSL certificates in order to authenticate each other, establish an SSL connection, and transmit information. Compared to a one-way connection, a two-way SSL connection provides a much higher level of security.

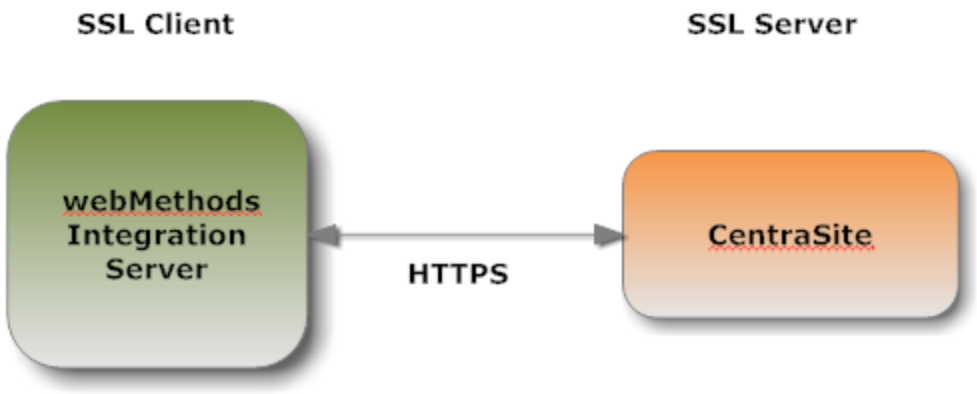
CentraSite as an SSL Client

When the CentraSite submits a HTTPS request to the webMethods Mediator, CentraSite is the SSL client and the webMethods Mediator with which it is communicating acts as the SSL server.



CentraSite as an SSL Server

When the webMethods Mediator submits a request to CentraSite via HTTPS, and a two-way SSL connection is established, the webMethods Mediator acts as the SSL client and the CentraSite acts as the SSL server.



Roadmap for Configuring SSL

The following table provides a high-level roadmap for configuring SSL on CentraSite.

Task	Activities	Notes
Create CentraSite keys and certificates	■ Generate a public key/private key pair. ■ Generate a certificate signing request (CSR) and send to the certificate authority (CA) for signing. ■ Receive validated certificate from the CA. ■ Import signed certificate into a keystore.	Required for one-way and two-way SSL connections. Refer to the documentation for Java *keytool* or your certificate management tool.
Create keystore and truststore for CentraSite	■ Create a keystore and import the signed certificate and private key. ■ Create a truststore and import the certificate of the signing CA. ■ Store the keystore and truststore in a secure CentraSite certificates directory. **Important:** If you use a Java *keytool* to create the keystore, you cannot import an existing private key. You can use other tools such as OpenSSL or Portecle.	Required for one-way and two-way SSL connections. Refer to the documentation for your certificate management tool.

Task	Activities	Notes
Obtain certificates of webMethods Mediator	Use the CentraSite truststore to save: ■ Signed certificate of the webMethods Mediator. ■ Signed certificate of the CA for the Mediator's SSL certificate.	Required for one-way and two-way SSL connections.

Creating CentraSite Keys and Certificates

Use a standard certificate management tool, such as OpenSSL or Portecle, to generate a private/public key pair for CentraSite. Then, place the public key in a certificate signing request (CSR).

After creating the CSR, send to the CA to sign the CSR. Request the certificate in DER format. If you receive a certificate in PEM format (or any format other than DER), you need to convert it to DER format.

The signing CA's certificate attests to the identity of the CA that signed the digital certificate for the CentraSite. The CA should send this certificate to you when it sends you the digital certificate for the CentraSite.

Once you receive your signed certificate from the CA, you need to import the certificate into a keystore. You will then have an SSL certificate and private key to use with CentraSite.

Note: The above process is described in general terms. The procedures may vary somewhat, depending upon the CA that you use.

Creating a Keystore and Truststore

Keystores and truststores are files that function as repositories for storage of keys and certificates necessary for SSL authentication, encryption/decryption, and digital signing/verification services. Keystores and truststores provide added layers of security and ease of administration, compared to maintaining the keys and certificates in separate files.

For information about creating keystores and truststores, importing keys and certificates into keystores and truststores, and other operations with these files, refer to the documentation for your certificate management tool.

For information about using CentraSite with keystores and truststores, see *Keystores and Truststores*.

Obtaining the Certificates and Keys of the webMethods Mediator

If your CentraSite will submit HTTPS requests to the webMethods Mediator, the CentraSite will be acting as a client and will receive certificates from this webMethods Mediator. In order for these transactions to work, your CentraSite must have copies of their public keys and signing CA certificates. For information on importing webMethods Mediator certificates to CentraSite and setting up client authentication, refer to the document *Administering webMethods Integration Server*.

Keystores and Truststores

CentraSite stores its private keys and SSL certificates in keystore files and the trusted roots for the certificates in truststore files. Keystores and truststores are secure files with industry-standard file formats.

Keystore File

CentraSite uses a special file called a *keystore* to store SSL certificates and keys.

A keystore file contains one or more pairs of a private key and signed certificate for its corresponding public key. The keystore should be strongly protected with a password, and stored (either on the file system or elsewhere) so that it is accessible only to administrators.

Keystore File Formats

The default, certificate file format for a CentraSite keystore is. JKS (Java keystore). Java keystore is a commonly used, standardized, certificate file format that provides a high degree of portability. PKCS#12 is another format you can use for a keystore. Other keystore types can be made available by:

- Loading additional security providers
- Setting the watt.security.keyStore.supportedTypes property.

HSM-Based Keystores

Under certain conditions, webMethods Mediator supports the use of keystore files stored on a Hardware Security Module (HSM). Integration Server supports HSM-based keystores for ports, but not for other components. You cannot use HSM-based keystores with remote server aliases, outbound certificates, an internal server port, WS-Security, or Integration Server public services.

Creating a Keystore

You can create and manage CentraSite keystores at the command line using keytool, a Java certificate editor.

You can also use other standard tools such as OpenSSL and Portecle.

Note: Software AG does not provide its own set of keystore utilities for creating or managing keystore files.

Truststore File

CentraSite uses a *truststore* to store its trusted root certificates, which are the public keys for the signing CAs. Although a truststore can contain the trusted roots for entire certificate chains, there is no requirement for the organization of certificates within a CentraSite truststore. It simply functions as a database containing all the public keys for CAs within a specified trusted directory.

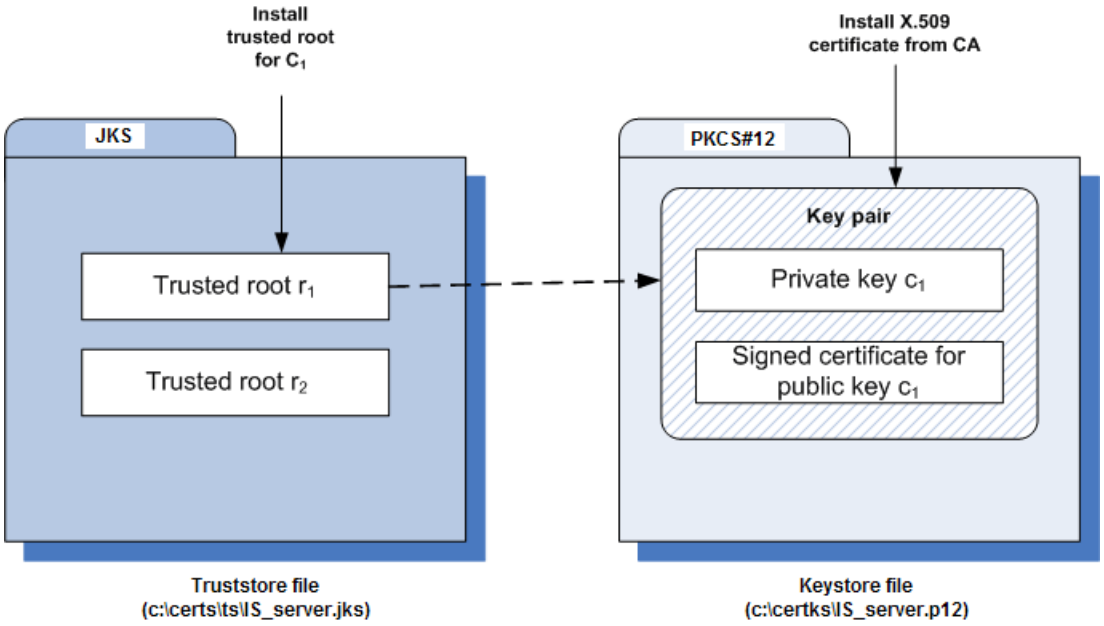
Truststore File Formats

CentraSite uses a *truststore* to store its trusted root certificates, which are the public keys for the signing CAs. Although a truststore can contain the trusted roots for entire certificate chains, there is no requirement for the organization of certificates within a CentraSite truststore. It simply functions as a database containing all the public keys for CAs within a specified trusted directory.

How CentraSite Uses a Keystore and Truststore

For a CentraSite service to be SSL authenticated, it must have a valid, authorized X.509 certificate installed in a keystore file *and* the private key and signing certificate for the certificate issuer (CA) installed in a truststore file. The following figure illustrates these requirements and the relationship between the two files.

Example Truststore File and Keystore File Showing Relationship



As shown in the above figure, the same truststore file can contain multiple trusted root certificates (public keys for the signing CAs). These trusted roots might be associated with numerous keystore files. A keystore file can contain the key pairs for multiple CentraSite services, and the entire certificate chain required for a service's authentication.

With a certificate chain, it is necessary to validate each subsequent signature in the list until a trusted CA certificate is reached. For CentraSite, you must include the entire chain of certificates in a keystore and truststore. Also, any root CA certificates in use by clients must be in a CentraSite truststore.

Protecting Keystore and Truststore Files

Keystore and truststore files exist within the file system of your operating system, and since these are critically important files, you want to maintain them in a secure directory path. If either of the these files cannot be located, CentraSite authentication will be disabled and no connection with the CentraSite can be made. It is recommended that only users serving as CentraSite administrators have access to these certificate files.

Configuring CentraSite to Use SSL

The configuration settings covered in this section deal with the CentraSite client side.

You can configure CentraSite client to use SSL in any of the following ways:

- Configure CentraSite Client to Use One-way SSL
- Configure CentraSite Client to Use Two-way SSL
- Using the CTP Server.xml File for SSL

Configure CentraSite Client to Use One-way SSL

You perform the following procedure to configure CentraSite for one-way SSL authentication:

To configure one-way SSL

1 Create at least one truststore *centrasitetruststore.jks*, in JKS format, in a desired location on the machine where CentraSite is running.

2 Import the webMethods Mediator's self-signed certificate *mediator.cer* into the above created truststore or JAVA cacerts.

When prompted for password, the default for truststores is "password".

```
C:\deploykeystores\new>keytool -export -alias mediator -keystore ↩
mediatorkeystore.jks -rfc -file mediator.cer
Enter keystore password:
Certificate stored in file <mediator.cer>

C:\deploykeystores\new>keytool -import -alias mediator -keystore ↩
centrasitetruststore.jks -file mediator.cer
Enter keystore password:
Re-enter new password:
Owner:
Issuer:
Serial number:
Valid from:
Certificate fingerprints:
             Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\deploykeystores\new>
```

If opting to import certificate in to Java cacerts, the Java runtime needs to trust the certificates of the webMethods Mediator (regardless of whether this is a Tomcat application or a standalone application) in order to establish the SSL connections. To do that, add the certificate to the trusted certificates of Java via the *keytool* utility that comes with Java. The following command will add the certificate located at a location (for example, c:\temp\server.crt) to the trusted certificates in the Java used by CentraSite:

```
keytool.exe -import -v -trustcacerts -alias test -file "C:\temp\server.crt"
-keystore "<JDKInstallDir>\jre\lib\security\cacerts"
```

When prompted for password, the default for Java is "changeit".

3 Open the *wrapper.conf* file located in <*CentraSiteInstallDir*>/profiles/CTP/configuration

4 Go to the section #`Java Additional Parameters`. Add the following property lines:

```
wrapper.java.additional.7=-Djavax.net.ssl.trustStore="C:/deploykeystores/new/centrasitetruststore.jks"
wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=password
```

5 Set the values as needed:

`wrapper.java.additional.7=-Djavax.net.ssl.trustStore=` represents the location of a truststore file (for example, *centrasitetruststore.jks*).

`wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=` represents the password for a truststore.

6 Save and close the file.

7 Now restart the CentraSite Tomcat. All communication via the webMethods Mediator to the database should now be using SSL.

Configure CentraSite Client to Use Two-way SSL

You perform the following procedure to configure CentraSite for two-way SSL authentication:

To configure two-way SSL

1 Using OpenSSL, create a self-signed certificate (*centrasite.cer*) with the following command:

```
openssl req -new -x509 -days 2000 -sha1 -newkey rsa:1024 -nodes
-keyout server.key -out server.crt -subj "/O=Company/OU=Unit/CN=localhost"
```

Whatever is specified in the CN section of the subject must match the hostname of the machine running the webMethods Mediator and is used to send requests to the Mediator.

2 Create at least one keystore *centrasitekeystore.jks*, in PKCS#12 or JKS format, containing a CentraSite key pair to use for SSL.

```
C:\deploykeystores\new>keytool -v -genkeypair -alias centrasite -keyalg RSA ↩
-validity 1000 -keystore centrasitekeystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?

Enter key password for <centrasite>
            <RETURN if same as keystore password>:
[Storing centrasitekeystore.jks]

C:\deploykeystores\new>
```

3 Create at least one truststore *centrasitetruststore.jks*, in JKS format, in a desired location on the machine where CentraSite is running.

4 Import the webMethods Mediator's self-signed certificate *mediator.cer* into the above created truststore or Java cacerts.

When prompted for password, the default for truststores is "password".

```
C:\deploykeystores\new>keytool -export -alias mediator -keystore ↩
mediatorkeystore.jks -rfc -file mediator.cer
Enter keystore password:
Certificate stored in file <mediator.cer>

C:\deploykeystores\new>keytool -import -alias mediator -keystore ↩
centrasitetruststore.jks -file mediator.cer
Enter keystore password:
Re-enter new password:
Owner:
Issuer:
Serial number:
Valid from:
Certificate fingerprints:
             Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\deploykeystores\new>
```

If opting to import certificate in to Java cacerts, the Java runtime needs to trust the certificates of the webMethods Mediator (regardless of whether this is a Tomcat application or a standalone application) in order to establish the SSL connections. To do that, add the certificate to the trusted certificates of Java via the *keytool* utility that comes with Java. The following command will add the certificate located at a location (for example, c:\temp\server.crt) to the trusted certificates in the Java used by CentraSite:

```
keytool.exe -import -v -trustcacerts -alias test -file "C:\temp\server.crt"
-keystore "<JDKInstallDir>\jre\lib\security\cacerts"
```

When prompted for password, the default for Java is "changeit".

5 Export the CentraSite's self-signed certificate *centrasite.cer* in to the webMethods Mediator's truststore.

6 Open the *wrapper.conf* file located in <*CentraSiteInstallDir*>/profiles/CTP/configuration

<*CentraSiteInstallDir*>/profiles/CTP/configuration

7 Go to the section #`Java Additional Parameters`. Add the following property lines:

```
wrapper.java.additional.5=-Djavax.net.ssl.keyStore="C:/deploykeystores/new/centrasitekeystore.jks"
wrapper.java.additional.6=-Djavax.net.ssl.keyStorePassword=password
wrapper.java.additional.7=-Djavax.net.ssl.trustStore="C:/deploykeystores/new/centrasitetruststore.jks"
wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=password
```

8 Set the values as needed:

`wrapper.java.additional.5=-Djavax.net.ssl.keyStore=` represents the location of a keystore file (say, *centrasitekeystore.jks*).

`wrapper.java.additional.6=-Djavax.net.ssl.keyStorePassword=` represents the password for a keystore.

`wrapper.java.additional.7=-Djavax.net.ssl.trustStore=` represents the location of a truststore file (say, *centrasitetruststore.jks*).

`wrapper.java.additional.8=-Djavax.net.ssl.trustStorePassword=` represents the password for a truststore.

9 Save and close the file.

10 Now restart the CentraSite Tomcat. All communication via the webMethods Mediator to the database should now be using SSL.

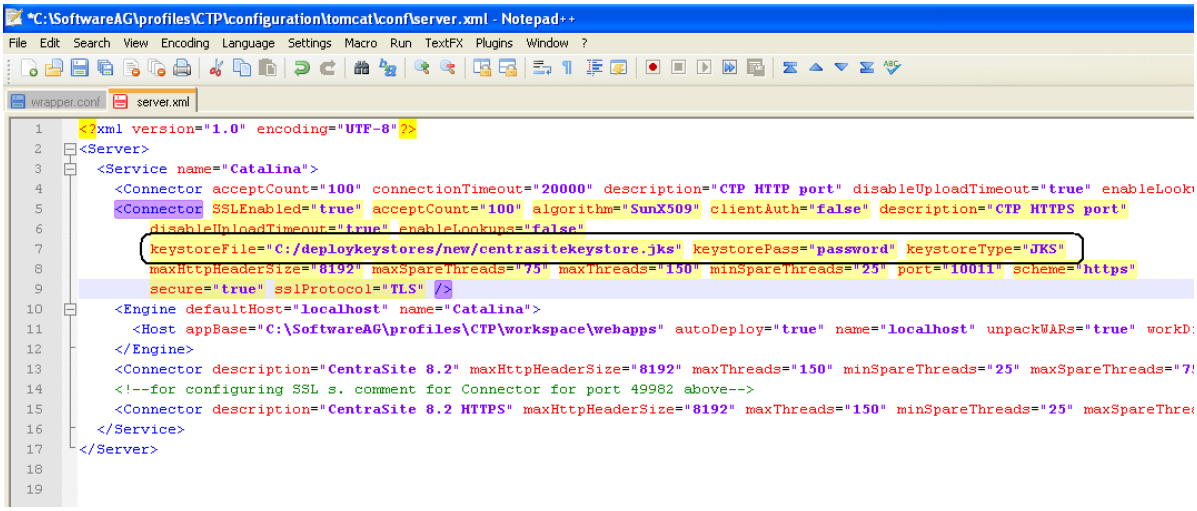
Using the CTP Server.xml File for SSL

To configure SSL using CTP server.xml file

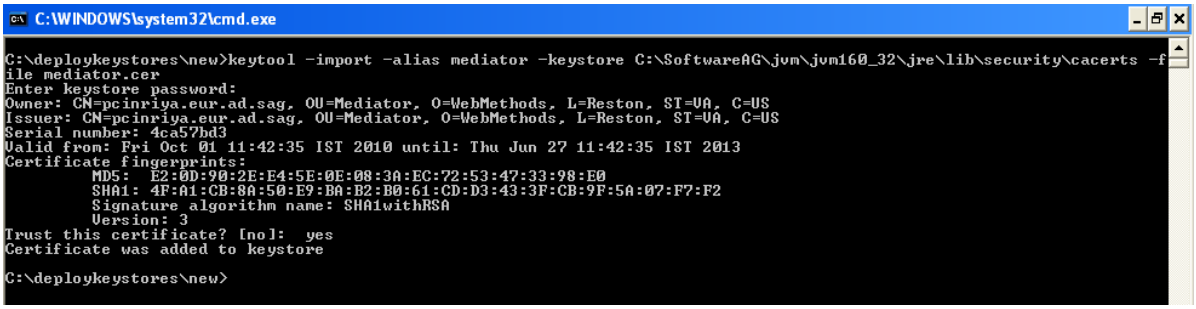
1 Open the *server.xml* file located in the following directory:

<*CentraSiteInstallDir*>/profiles/CTP/configuration/tomcat/conf

2 Enter the keystore information as specified below:



3 Add the mediator certificate (*mediator.cer*) into CentraSite JVM cacerts as below:



Configuring webMethods Integration Server to Use SSL

The configuration settings covered in this section deal with the webMethods Integration Server side.

You configure an Integration Server to use one of the following:

- Configure Integration Server to Use One-way SSL

- Configure Integration Server to Use Two-way SSL

Configure Integration Server to Use One-way SSL

You perform the following procedure to configure Integration Server for one-way SSL authentication:

To configure one-way SSL

1 Using OpenSSL, create a self-signed certificate (*mediator.cer*) with the following command:

```
openssl req -new -x509 -days 2000 -sha1 -newkey rsa:1024 -nodes
-keyout server.key -out server.crt -subj "/O=Company/OU=Unit/CN=localhost"
```

Whatever is specified in the CN section of the subject must match the hostname of the machine running the webMethods Mediator and is used to send requests to Mediator.

2 Create at least one keystore *mediatorkeystore.jks*, in PKCS#12 or JKS format, containing an Integration Server key pair to use for SSL and its corresponding key alias.

```
C:\deploykeystores\new>keytool -v -genkeypair -alias mediator -keyalg RSA ↩
-validity 1000 -keystore mediatorkeystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?

Enter key password for <mediator>
            <RETURN if same as keystore password>:
[Storing mediatorkeystore.jks]

C:\deploykeystores\new>
```

3 Export the webMethods Mediator's self-signed certificate *mediator.cer* into the CentraSite's truststore.

4 Configure an HTTPS port and specify the client authentication to **Username/Password**. The server prompts the client for a user ID and password.

5 On the **Ports** screen, click **Edit** to change the **Access Mode**. You may Set Access Mode to **Allow by Default** or **Reset to default access settings**.

For more information on configuring ports and client authentication, refer to the document *Administering webMethods Integration Server* in the documentation set for webMethods Integration Server.

6 Now restart the Integration Server.

Configure Integration Server to Use Two-way SSL

You perform the following procedure to configure Integration Server for one-way SSL authentication:

To configure two-way SSL

1 Using OpenSSL, create a self-signed certificate (*mediator.cer*) with the following command:

```
openssl req -new -x509 -days 2000 -sha1 -newkey rsa:1024 -nodes
-keyout server.key -out server.crt -subj "/O=Company/OU=Unit/CN=localhost"
```

Whatever is specified in the CN section of the subject must match the hostname of the machine running the webMethods Mediator and is used to send requests to the Mediator.

2 Create at least one keystore *mediatorkeystore.jks*, in PKCS#12 or JKS format, containing an Integration Server key pair to use for SSL.

```
C:\deploykeystores\new>keytool -v -genkeypair -alias mediator -keyalg RSA ↩
-validity 1000 -keystore mediatorkeystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?

Enter key password for <mediator>
            <RETURN if same as keystore password>:
[Storing mediatorkeystore.jks]

C:\deploykeystores\new>
```

3 Create at least one truststore *mediatortruststore.jks*, in JKS format, in a desired location on the machine where CentraSite is running.

4 Export the webMethods Mediator's self-signed certificate *mediator.cer* into the CentraSite's truststore.

5 Import the CentraSite's self-signed certificate *centrasite.cer* in to the mediator's truststore *mediatortruststore.jks*.

```
C:\deploykeystores\new>keytool -export -alias

centrasite -keystore centrasitekeystore.jks -rfc -file

centrasite.cer
Enter keystore password:
Certificate stored in file <centrasite.cer>

C:\deploykeystores\new>keytool -import -alias

mediator -keystore mediatortruststore.jks -file

centrasite.cer
Enter keystore password:
Re-enter new password:
Owner:
Issuer:
Serial number:
Valid from:
Certificate fingerprints:
             Trust this certificate? [no]:  yes
Certificate was added to keystore

C:\deploykeystores\new>
```

6 Create a keystore and truststore alias using the above created keystore (*mediatorkeystore.jks*) and truststore (*mediatortruststore.jks*) respectively. For more information on creating keystore and truststore aliases, refer to the document *Administering webMethods Integration Server* in the documentation set for webMethods Integration Server.

7 Configure an HTTPS port and specify the client authentication to any of the following:

- **Username/Password**. The server prompts the client for a user ID and password.
- **Request Client Certificates**. The server requests client certificates for all requests. If the client does not provide a certificate, the server prompts the client for a userid and password. If the client provides a certificate:
  - The server checks whether the certificate exactly matches a client certificate on file and is signed by a trusted authority. If so, the client is logged in as the user to which the certificate is mapped in Integration Server. If not, the client request fails, unless central user management is configured.
  - If central user management is configured, the server checks whether the certificate is mapped to a user in the central user database. If so, the server logs the client on as that user. If not, the client request fails.
- **Require Client Certificates**. The server requires client certificates for all requests. The server behaves as described for *Request Client Certificates*, except that the client must always provide a certificate.

8 On the **Ports** screen, click **Edit** to change the **Access Mode**. You may Set Access Mode to Allow by Default or Reset to default access settings.

9 If chosen client authentication as **Require Client Certificates** above, map the client certificate to any valid user in the Integration Server.

For more information on configuring ports and client authentication, refer to the document *Administering webMethods Integration Server* in the documentation set for webMethods Integration Server.

10 Now restart the Integration Server.

Configuring webMethods Mediator to Use SSL

Configure your instance of webMethods Mediator as described in the section *Configuring Mediator* in the document *Administering webMethods Mediator* in the documentation set for webMethods Mediator.