webMethods CloudStreams Development Help

Version 10.1

October 2017

WEBMETHODS

This document applies to webMethods CloudStreams Version 10.1 and to all subsequent releases.

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

Document ID: WST-OH-101-20171017

Table of Contents

About this Guide

Software AG CloudStreams provides a scalable approach for the development and governance of integration flows between "Software as a Service" (SaaS) applications and "on-premise" applications.

CloudStreams provides predefined, configurable CloudStreams Connectors that enable you to connect to popular SaaS cloud applications. Using the CloudStreams plug-ins and administrative options, you configure these Connectors to govern transactions, log payloads and monitor run-time performance. You can publish and view run-time performance metrics and events using the Software AG MashZone dashboards provided by CloudStreams.

Additionally, you can use the CloudStreams framework to create custom connectors to other SaaS applications.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.

Convention	Description
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 CloudStreams Governance Project

Governance Project Wizard

Use this wizard to create a CloudStreams Governance project in which you will create your virtual services, Connector Virtual Services, and their policies. You create CloudStreams Governance projects in your local file system, using the CloudStreams Development plug-in provided by Designer. Each project will contain the folders in which you will create virtual services, Connector Virtual Services, and their policies. You can create one or more projects.

To create a CloudStreams Governance project:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. The CloudStreams Governance view on the left side of the page lists all existing CloudStreams Governance projects.
3. Right-click an existing CloudStreams Governance project and click **New > CloudStreams Governance Project** (or click **File > New > Project > Software AG > CloudStreams > CloudStreams Governance Project** from the menu and click Next.

Complete the following fields in the New CloudStreams Governance Project wizard:

Project Name
A project name that is a valid resource name on your operating system. The name must not be null, cannot be an empty string, and the following invalid OS resource characters are not allowed:

- \\ (double backward slashes)
- / (forward slash)
- : (colon)
- * (asterisk)
- ? (question mark)
- " (double quote)
- < (Less Than symbol)
- > (Greater Than symbol)
- | (vertical bar)

Use Default Location
This option is selected by default. The default location is the Workspace root.

For example, if you are using C:\Workspaces\My_Workspace.. the default location would be C:\Workspaces\runtime-Eclipse Application, which is the Workspace root.

Location
Select a location in your local file system to store the new project.

Choose file system
Choose a file system, either “default” or “RSE”.

Publisher
Optional. Provide the name of the publisher of the project.

Description
Optional. Provide a description of the project.

The new CloudStreams Governance project is added to the existing projects in the CloudStreams Governance view, and it includes the default folders **Virtual Services**, **Connector Virtual Services** and **Policies**.

Note: Instead of creating a new CloudStreams Governance project, you can import an existing one. To do this, from the Designer menu click **File > Import > Software AG > CloudStreams Governance Project** and complete the dialog box that appears.

CloudStreams Servers Dialog Boxes

Use these dialog boxes to add and manage CloudStreams server targets. A CloudStreams server target specifies an instance of a CloudStreams server to which you will deploy your virtual services and connector virtual services.

CloudStreams Servers Dialog Box

Use this dialog box to add or remove CloudStreams servers to the target list, edit their configuration, import/export target instances, and test the server connections.

To open the CloudStreams Servers dialog box:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. Select **Window > Preferences** from the menu.
3. Expand the **Software AG** folder and click **CloudStreams Servers**.
4. In the CloudStreams Servers window, click **Add**.

Complete the following fields in the Add CloudStreams Server dialog box as follows and click **OK**:

Name
A name for the new target. Target names can contain alphanumeric characters and underscores (_) and hyphens (-).

Host
The server's host name (for example, `localhost`).

Port
The server's port number.

User
Optional. The Integration Server user who is permitted to deploy assets to this target. By default, only a member of the Integration Server's Administrator group is permitted to deploy assets to this target.

Password
Optional. The password of the Integration Server user who is permitted to deploy assets to this target. By default, the password of this user is `manage`.

Secure Connection
Indicates whether the session will be opened through HTTP or HTTPS. If you want to open an HTTPS session on the selected server using the Secure Socket Layer (SSL), select this check box. If you select this option, it is critical that you also specify the **IS Truststore Name** option for CloudStreams (in Integration Server Administrator go to **Solutions > CloudStreams > Administration > General**. Alternatively, if you want to open an HTTP session on the server, clear this check box.

Import
Use this button to import target instances from an archive file into the same Integration Server or to another instance of the server.

Export
Use this button to export target instances from the Integration Server to an archive file on the file system.

Add
Use this button to display the Add CloudStreams Servers dialog box, to add a server to the list.

Edit
Use this button to edit any parameter.

Remove
Use this button to remove the server from the list.

Test
Use this button to test the target's connection to the CloudStreams server. If the connection is not active and valid, activate the deployment endpoint and modify the user credentials as required.

Add CloudStreams Servers Dialog Box

To open the Add CloudStreams Servers dialog box: In the Designer menu, click **Window > Preferences > Software AG > CloudStreams Servers** and click the **Add** button.

Complete the following fields:

Name
A name for the new target. Target names can contain alphanumeric characters and underscores (_) and hyphens (-).

Host Name
The server's host name (for example, `localhost`).

Port
The server's port number.

User
Optional. The Integration Server user who is permitted to deploy assets to this target. By default, only a member of the Integration Server's Administrator group is permitted to deploy assets to this target.

Password
Optional. The password of the Integration Server user who is permitted to deploy assets to this target. By default, the password of this user is `manage`.

Secure Connection
Indicates whether the session will be opened through HTTP or HTTPS. If you want to open an HTTPS session on the selected server using the Secure Socket Layer (SSL), select this check box. If you select this option, it is critical that you also specify the **IS Truststore Name** option for CloudStreams (in Integration Server Administrator go to **Solutions > CloudStreams > Administration > General**. Alternatively, if you want to open an HTTP session on the server, clear this check box.

Connector Virtual Services (SOAP)

Use this editor to create custom SOAP-based connector virtual services. *Connector virtual services* handle requests in the outbound processing scenario. When an on-premise application sends a service request to a SaaS application, a connector virtual service handles the provider's responses and logs the requests/responses.

A connector virtual service runs on CloudStreams and acts as the consumer-facing proxy for a native service running in a SaaS application. A connector virtual service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing independence (of location, protocol and format) between the consuming application and the provider service.

CloudStreams provides a default connector virtual service for each metadata handler: the service WmCloudStreams.SoapVS (for the SOAP handler) and the service WmCloudStreams.RestVS (for the REST handler). The default services are located in the sample CloudStreams Governance project in the WmCloudStreams package. You cannot modify these default services.

Each default connector virtual service has a default policy (named Logging Policy), which logs all requests/responses to a database. You cannot modify the default policy. Alternatively, you can create additional connector virtual services with custom policies. If you create a connector virtual service with a custom policy, you can only include the

actions in the "Monitoring" or "Additional" action categories; you cannot include the "WS-SecurityPolicy 1.2" actions. For example, you might want to create a custom policy that monitors run-time performance, customizes how the service invocations are logged, validates response messages against an XML schema, or optimizes server traffic.

New Connector Virtual Service Wizard (SOAP)

To open the New Connector Virtual Service wizard:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, right-click the CloudStreams Governance project and click **New > Connector Virtual Service** (or expand the project, right-click the **Connector Virtual Services** folder and click **New Connector Virtual Service**).

Project
The CloudStreams Governance project in which you are creating the connector virtual service.

Name
Assign a name for the service. Unlike native services, the names of virtual services cannot contain spaces or special characters except _ (underscore) and - (hyphen). Consequently, if you adopt a convention that involves using the name of the service as part of the virtual service name, then the names of the services themselves must not contain characters that are invalid in virtual service names.

If you want to change the service name after it has been created, right-click the service name in the **Connector Virtual Services** folder and select **Rename**. The service must be undeployed before you can rename it (check the **Deployed Status** field in the Advanced page of the service's Properties view).

Version
The version is always set to 1.0.

Type
Select **SOAP**.

Description
Optional. A description for the service. This description appears when a user displays a list of connector virtual services in the user interface.

General Properties View (SOAP Connector Virtual Service)

To display the general properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open,

click **Window > Show View > Other > General > Properties**). Click the virtual service name and then select the "General" tab.

The **General** page in the Properties view displays the following properties:

Name
(Read-only field.) The service name. You can change the service name at any time by right-clicking the name in the **Virtual Services** folder and clicking **Rename**.

Service Type
(Read-only field.) **SOAP**.

Created/Last Modified
(Read-only field.) The service's creation/modification timestamps.

Target Namespace
(Read-only field.) The value is derived from the `targetNamespace` attributes of the WSDL's definition element.

Namespaces
Click the button next to this field to view other namespaces (such as `wsdl`, `xsd`, `soap`, etc.).

Version
The version is always set to 1.0.

WSDL URL
(Read-only field.) The URL of the service's WSDL.

Description
You can change the service description.

Advanced Properties View (SOAP Connector Virtual Service)

To display the advanced properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**). Click the virtual service name and then select the "Advanced" tab.

Name
(Read-only field.) The service name.

Type
(Read-only field.) The type of the service (SOAP)

Target Namespace
(Read-only field.) The value is derived from the `targetNamespace` attributes of the WSDL's definition element.

WSDL URL
Click this URL to display the contents of the service's abstract WSDL.

Namespaces
Click the button next to this field to view other namespaces (such as `wsdl`, `xsd`, `soap`, etc.).

Version
The version is always set to 1.0.

Description
(Read-only field.) The service description.

VSD
Click the button next to this field to view the service's "virtual service definition" (VSD). This button is disabled until you deploy the connector virtual service to a CloudStreams server.

When you deploy the connector virtual service to a CloudStreams server, CloudStreams generates an XML document called a *virtual service definition (VSD)*. The VSD defines the connector virtual service for CloudStreams, and contains all the resources required to deploy the connector virtual service to a CloudStreams server, including the policy that applies to the service. You cannot edit the VSD.

> **Note:** If multiple policies apply to the service, CloudStreams combines all those policies into a single policy known as the *effective policy*. The effective policy is a simple UNION of the run-time actions specified in all policies that apply to a service. To create the effective policy, CloudStreams evaluates the combined list of actions from all policies, using a set of internal rules known as Policy Conflict Resolution rules. For details, see the document *Administering webMethods CloudStreams*.

Deployed Status
(Read-only field.) Indicates whether the service is **Deployed**, **Undeployed** or **Not Deployed** (which is the initial status before you deploy the service).

Applicable Policies
Click the button next to this field to view a list of the active policies that apply to this service. Any inactive policy that applies to the service is not listed.

Endpoint
(Read-only field.) The endpoint is resolved when the connector service is configured and the user selects an enabled managed connection pool.

Entry Step (SOAP Connector Virtual Service)

To display the Entry Step page:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.

3. Click **Entry Step** and view the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Entry Step**.

Protocol
(Read-only field.) The protocol of the requests that the connector virtual service will accept. The value Local means that the service can only be called from Cloud Connector Services.

Routing Rule Step (SOAP Connector Virtual Service)

To display the Routing Rule page:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and view the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
Read-only field.) **Protocol**. You cannot change the protocol over which the connector virtual service will accept requests.

Routing Type
(Read-only field.) The value **Connection Pool** means that the requests are sent to the provider using the CloudStreams connection pool. For more information about connection pools, see the documentation specific to your CloudStreams connector (for example, *webMethods CloudStreams Provider for Salesforce.com Installation and User's Guide*).

Invoke IS Service Step (Inbound, SOAP Connector Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "In Sequence" step to pre-process the request message before it is submitted to the native service. You can create multiple "Invoke IS Service" steps.

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

You can use predefined or custom context variables in an IS service. For more information, see *Using Context Variables* in the document *Administering webMethods CloudStreams*.

To add the Invoke IS Service step:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Right-click **In Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the Entry Step. You cannot change the order of the steps.
3. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The IS Service to pre-process the request message before it is submitted to the native service.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `EnvelopeString`
- `MessageContext`: Integration Server will automatically place a MessageContext variable into the pipeline before executing the 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 (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 Serverpub.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*.

Invoke IS Service Step (Outbound, SOAP Connector Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "Out Sequence" step to pre-process the response message from the

native service before it is returned to the consuming application. You can create multiple "Invoke IS Service" steps.

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

You can use predefined or custom context variables in an IS service. For more information, see the section *Using Context Variables* in the document *Administering webMethods CloudStreams*.

To add the Invoke IS Service step (outbound):

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Right-click **Out Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the "Out Sequence" step.
3. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The IS Service to pre-process the response message before it is returned to the consuming application. Click **Browse** to select a file from your file system.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `EnvelopeString`
- `MessageContext`: CloudStreams will automatically place a MessageContext variable into the pipeline before executing the 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 (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 Serverpub.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*.

Error Messaging Step (SOAP Connector Virtual Service)

The default connector virtual services are configured to return the native service provider's service fault, if one is available. CloudStreams will send whatever content it received from the native service provider. You can optionally invoke IS services to pre-process or post-process the error messages.

To configure the Error Messaging step:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Expand the **Error Sequence** step in the editor.
3. Click **Error Messaging** and complete the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming conventions.

Type
(Read-only field.) Error Messaging.

Error Message
Select one or both of the following options:

- **Custom Failure Response Message**: Returns the following fault response to the consuming application:

  ```
  CloudStreams 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. CloudStreams 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 CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors).

    In this case, the `$ERROR_MESSAGE` variable will contain errors generated by CloudStreams.

The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.), which are described in the document *Administering webMethods CloudStreams*.

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

- The predefined CloudStreams context variables listed in the section *The Predefined Context Variables* in the document *Administering webMethods CloudStreams*.
- Custom context variables that you declare using the CloudStreams API (see the section *The API for Context Variables* in the document *Administering webMethods CloudStreams*).

- **Native Provider Fault**: When you select this option, CloudStreams sends the native service provider's service fault content, if one is available. CloudStreams will send whatever content it received from the native service provider.

  If you select this option, the **Custom Failure Response Message** is ignored when a fault is returned by the native service provider. (Faults returned by internal CloudStreams exceptions will still be handled by the **Custom Failure Response Message** option.)

Processing Method

Optionally select either of the following:

- **Pre-Processing**: Select this option if you want to invoke an IS service to manipulate the response message before the Error Sequence 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 about IS services, see "Invoke IS Service Step (Inbound, SOAP Connector Virtual Service)" on page 17.
- **Post-Processing**: Select this option if you want to invoke one or more IS services to manipulate the service fault after the Error Sequence 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.

Connector Virtual Services (REST)

Use this editor to create custom REST-based connector virtual services. *Connector virtual services* handle requests in the outbound processing scenario. When an on-premise application sends a service request to a SaaS application, a connector virtual service handles the provider's responses and logs the requests/responses.

A connector virtual service runs on CloudStreams and acts as the consumer-facing proxy for a native service running in a SaaS application. A connector virtual service provides a layer of abstraction between the service consumer and the service provider,

and promotes loose coupling by providing independence (of location, protocol and format) between the consuming application and the provider service.

CloudStreams provides a default connector virtual service for each metadata handler: the service WmCloudStreams.RestVS (for the REST handler) and the service WmCloudStreams.SoapVS (for the SOAP handler). The default services are located in the sample CloudStreams Governance project in the WmCloudStreams package. You cannot modify these default services.

Each default connector virtual service has a default policy (named Logging Policy), which logs all requests/responses to a database. You cannot modify the default policy. Alternatively, you can create additional connector virtual services with custom policies. If you create a connector virtual service with a custom policy, you can only include the actions in the "Monitoring" or "Additional" action categories; you cannot include the "WS-SecurityPolicy 1.2" actions. For example, you might want to create a custom policy that monitors run-time performance, customizes how the service invocations are logged, validates response messages against an XML schema, or optimizes server traffic.

New Connector Virtual Service Wizard (REST)

To open the New Connector Virtual Service wizard:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, right-click the CloudStreams Governance project and click **New > Connector Virtual Service** (or expand the project, right-click the **Connector Virtual Services** folder and click **New Connector Virtual Service**).

Complete the following fields in the New Connector Virtual Service wizard:

Project
Click **Browse** and select a CloudStreams Governance project in which to create the connector virtual service.

Name
Assign a name for the service. Unlike native services, the names of virtual services cannot contain spaces or special characters except _ (underscore) and – (hyphen). Consequently, if you adopt a convention that involves using the name of the service as part of the virtual service name, then the names of the services themselves must not contain characters that are invalid in virtual service names.

If you want to change the service name after it has been created, right-click the service name in the **Connector Virtual Services** folder and select **Rename**. The service must be undeployed before you can rename it (check the **Deployed Status** field in the Advanced page of the service's Properties view).

Version
The version is always set to 1.0.

Type
Select **REST**.

Description
Optional. A description for the service. This description appears when a user displays a list of connector virtual services in the user interface.

General Properties View (REST Connector Virtual Service)

To display the general properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**.)

The **General** page in the Properties view displays the following properties:

Name
(Read-only field.) The service name. You can change the service name at any time by right-clicking the name in the **Virtual Services** folder and clicking **Rename**.

Service Type
(Read-only field.) **REST**.

Created/Last Modified
(Read-only field.) The service's creation/modification timestamps.

Target Namespace
(Read-only field.) The value is derived from the `targetNamespace` attributes of the WSDL's definition element.

Namespace
Click the button next to this field to view other namespaces (such as `wsdl`, `xsd`, etc.).

Version
The version is always set to 1.0.

WSDL URL
(Read-only field.) The URL of the service's WSDL.

Description
You can change the service description.

Advanced Properties View (REST Connector Virtual Service)

To display the advanced properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.

2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**.)
3. Click the **Advanced** page in the Properties view, which displays the following properties.

Name
(Read-only field.) The service name.

Type
(Read-only field.) The type of the service (REST).

Target Namespace
(Read-only field.) The value taken from the `targetNamespace` attributes of the WSDL's definition element.

WSDL URL
Click this URL to display the contents of the service's abstract WSDL.

Namespace
Click the button next to this field to view other namespaces (such as `wsdl`, `xsd`, etc.).

Version
The version is always set to 1.0.

Description
(Read-only field.) The service description.

VSD
Click the button next to this field to view the service's "virtual service definition" (VSD). This button is disabled until you deploy the connector virtual service to a CloudStreams server.

When you deploy the connector virtual service to a CloudStreams server, CloudStreams generates an XML document called a *virtual service definition (VSD)*. The VSD defines the connector virtual service for CloudStreams, and contains all the resources required to deploy the connector virtual service to a CloudStreams server, including the policy that applies to the service. You cannot edit the VSD.

> **Note:** If multiple policies apply to the service, CloudStreams combines all those policies into a single policy known as the *effective policy*. The effective policy is a simple UNION of the run-time actions specified in all policies that apply to a service. To create the effective policy, CloudStreams evaluates the combined list of actions from all policies, using a set of internal rules known as Policy Conflict Resolution rules. For details, see the document *Administering webMethods CloudStreams*.

Applicable Policies
Click the button next to this field to view a list of the active policies that apply to this service. Any inactive policy that applies to the service is not listed.

Endpoint
Click the button next to this field to view the endpoint to which the virtual service is deployed (if applicable).

Deployed Status
(Read-only field.) Indicates whether the service is **Deployed**, **Undeployed** or **Not Deployed** (which is the initial status before you deploy the service).

Entry Step (REST Connector Virtual Service)

To display the Entry Step page:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Entry Step** and view the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Entry Step**.

Protocol
(Read-only field.) The protocol of the requests that the connector virtual service will accept. The value "Local" means that the service can be called only from Cloud Connector Services.

HTTP Methods
The HTTP methods that the virtual services should be allowed to perform on the native services.

It is important to specify all the HTTP methods that are supported for the virtual services. For example, if a virtual service is deployed to CloudStreams and only the GET method was selected here, then CloudStreams will only permit GET invocations. A POST request will be rejected with a return of statusCode 405 even if the native service happens to support POSTs. It is also important for the native service to return the correct Content-Type header with its responses.

At run time, CloudStreams determines the type of a request message based on the message's HTTP method and its Content-Type. For a list of valid HTTP method/Content-Type combinations, see the section *The Request Message's HTTP Methods and Content-Types for REST Services* in the document *Administering webMethods CloudStreams*.

Routing Rule Step (REST Connector Virtual Service)

To display the Routing Rule page:

1. Click the connector virtual service name in the CloudStreams Governance view.

2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and view the following fields in the Properties view.

Name
The step name, **Routing Rule**, which you may modify. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
Read-only field.) **Protocol**. You cannot change the protocol over which the connector virtual service will accept requests.

Routing Type
(Read-only field.) The value **Connection Pool** means that the requests are sent to the provider via the CloudStreams connection pool. For more information about connection pools, see the documentation specific to your CloudStreams connector (for example, *webMethods CloudStreams Provider for Salesforce.com Installation and User's Guide*).

Invoke IS Service Step (Inbound, REST Connector Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "In Sequence" step to pre-process the request message before it is submitted to the native service. You can create multiple "Invoke IS Service" steps.

An IS service must be running on the same Integration Server as CloudStreams. It can call out a C++ or Java or .NET function. It can also call other IS services.

You can use predefined or custom context variables in an IS service. For more information, see *Using Context Variables* in the document *Administering webMethods CloudStreams*.

To add the Invoke IS Service step:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Right-click **In Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the Entry Step. You cannot change the order of the steps.
3. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The IS Service to pre-process the request message before it is submitted to the native service.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `EnvelopeString`
- `MessageContext`: CloudStreams will automatically place a MessageContext variable into the pipeline before executing the 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 (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 Serverpub.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*.

Invoke IS Service Step (Outbound, REST Connector Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "Out Sequence" step to pre-process the response message from the native service before it is returned to the consuming application. You can create multiple "Invoke IS Service" steps.

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

You can use predefined or custom context variables in an IS service. For more information, see the section *Using Context Variables* in the document *Administering webMethods CloudStreams*.

To add the Invoke IS Service step (outbound):

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Right-click **Out Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the "Out Sequence" step.
3. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The IS Service to pre-process the response message before it is returned to the consuming application. Click **Browse** to select a file from your file system.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `EnvelopeString`
- `MessageContext`: CloudStreams will automatically place a MessageContext variable into the pipeline before executing the 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 (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 Serverpub.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*.

Error Messaging Step (REST Connector Virtual Service)

The default connector virtual services are configured to return the native service provider's service fault, if one is available. CloudStreams will send whatever content it received from the native service provider. You can optionally invoke IS services to pre-process or post-process the error messages.

To configure the Error Messaging step:

1. Click the connector virtual service name in the CloudStreams Governance view.
2. Expand the **Error Sequence** step in the editor.
3. Click **Error Messaging** and complete the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming conventions.

Type
(Read-only field.) Error Messaging.

Error Message
Select one or both of the following options:

- **Custom Failure Response Message**: Returns the following fault response to the consuming application:

  ```
  CloudStreams 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. CloudStreams 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 CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors).

    In this case, the `$ERROR_MESSAGE` variable will contain errors generated by CloudStreams.

  The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.), which are described in the document *Administering webMethods CloudStreams*.

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

  - The predefined CloudStreams context variables listed in the section *The Predefined Context Variables* in the document *Administering webMethods CloudStreams*.
  - Custom context variables that you declare using the CloudStreams API (see the section *The API for Context Variables* in the document *Administering webMethods CloudStreams*).

- **Native Provider Fault**: When you select this option, CloudStreams sends the native service provider's service fault content, if one is available. CloudStreams will send whatever content it received from the native service provider.

  If you select this option, the **Custom Failure Response Message** is ignored when a fault is returned by the native service provider. (Faults returned by internal

CloudStreams exceptions will still be handled by the **Custom Failure Response Message** option.)

Processing Method

Optionally select either of the following:

- **Pre-Processing**: Select this option if you want to invoke an IS service to manipulate the response message before the Error Sequence 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 about IS services, see "Invoke IS Service Step (Outbound, REST Connector Virtual Service)" on page 27.
- **Post-Processing**: Select this option if you want to invoke one or more IS services to manipulate the service fault after the Error Sequence 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.

Policies

Use this editor to create policies for virtual services and connector virtual services.

A *policy* provides run-time governance capabilities to a virtual service or a connector virtual service. A policy is a sequence of actions that are carried out by CloudStreams when a consumer requests a particular service through CloudStreams. The actions in a policy perform activities such as identifying/authenticating consumers, validating digital signatures and capturing performance measurements.

You should create a policy for each virtual service. You can apply a single policy to one or more virtual services. A policy for virtual services can include the following kinds of actions:

- **WS-SecurityPolicy 1.2 actions:**

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

  The authentication actions verify that the requests for virtual services contain a specified WS-SecurityPolicy element:

  - **Require SAML Token**: Requires that a WSS Security Assertion Markup Language (SAML) assertion token be present in the SOAP message header to validate service consumers.
  - **Require WSS Username Token**: Requires that a WSS username token and password be present in the SOAP message header to validate service consumers.
  - **Require X.509 Token**: Requires that a WSS X.509 token be present in the SOAP message header to validate service consumers.

The XML security actions provide confidentiality (through encryption) and integrity (through signatures) for request and response messages. CloudStreams includes the following XML security actions:

- **Require Signing**: Requires that a request's XML element (which is represented by an XPath expression) be signed.
- **Require Encryption**: Requires that a request's XML element (which is represented by an XPath expression) be encrypted.
- **Require SSL**: Requires that requests be sent via SSL client certificates and can be used with both SOAP and REST services.
- **Include Timestamps**: Requires that timestamps be included in the request header. CloudStreams checks the timestamp value against the current time to ensure that the request is not an old message. This serves to protect your system against attempts at message tampering, such as replay attacks.

- **Monitoring actions:**

  CloudStreams includes the following run-time monitoring actions:

  - **Monitor Service Performance**: This action monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when these performance conditions are violated.
  - **Monitor Service Level Agreement**: This action 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 especially for particular consumer(s). You can 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.
  - **Throttling Traffic Optimization**: Limits the number of service invocations 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:**

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

  - **Identify Consumer**: You use this action in conjunction with an authentication action ("Require WSS Username Token", "Require X.509 Token", or "Require HTTP Basic Authentication"). Alternatively, this action can be used alone to identify consumers only by host name or IP address.
  - **Require HTTP Basic Authentication**, which uses HTTP basic authentication to verify the user name and passwords of consumers 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. You use this action in conjunction

with an authentication action ("Require WSS Username Token", "Require SAML Token", or "Require HTTP Basic Authentication").

- **Log Invocation**, which logs request/response payloads.
- **Validate Schema**, which validates all XML request and/or response messages against an XML schema referenced in the WSDL.

Each default connector virtual service has a default policy, which you cannot modify. However, you can create additional connector virtual services with custom policies. If you create a connector virtual service with a custom policy, you can include only the monitoring, logging, and schema validation actions. (The virtual services, which receive the inbound requests, handle the security.) For example, you might want to create a custom policy that monitors run-time performance, customizes how the service invocations are logged, or optimizes server traffic.

When you create or modify a policy, you:

1. Specify the services to which the policy should apply. A policy can apply to one or more virtual services or to one or more connector virtual services, but not to both types of services.
2. Add the desired actions to the policy and configure their parameters.
3. Activate the policy when you are ready to put it into effect. When you deploy the virtual services or connector virtual services to which the policy is applied, the policy will also be deployed.

Create a New Policy Wizard

To open the New Policy wizard:

1. Open Designer and display the CloudStreams Development perspective.
2. In the CloudStreams Governance view, expand your CloudStreams Governance project folder, right-click the **Policies** folder and click **New Policy**.

Project
The CloudStreams Governance project in which you are creating the policy.

Name
Assign a name for the policy. A policy name can contain any character (including spaces).

> **Note:** If you want to change the policy name after it has been created, right-click the policy name in the **Policies** folder and select **Rename**.

Service Type
Specify whether the policy will be applied to a virtual service or to a connector virtual service.

Description
Optional. A description for the policy. This description appears when a user displays a list of policies in the user interface.

Criteria
Click "Next" to apply the policy to services that meet certain criteria.

In the next dialog box, specify the criteria as follows:

1. In the **Criteria** field, specify a value (**SOAP** or **REST**) for the **Service Type** attribute. That is, specify whether the policy should apply to all SOAP services or all REST services.
2. Click the plus button next to the **Criteria** field to add a new row, and choose another criteria attribute (**Name** or **Description**) and choose an operator and value for it. For example, specify whether the policy should apply to all SOAP services that equals, starts with or contains the prefix `Abc_`.
3. Repeat step 2 if desired to specify additional criteria.
4. If you specify multiple criteria, use the **Condition** field to connect the criteria by the AND or OR operator. The default operator is AND.

Note: If you do not specify any criteria, the policy will apply to all virtual services.

Caution: CloudStreams checks for policy conflicts when you deploy a virtual service. If the service has only one policy applied to it (e.g., the policy you are applying here), that policy is deployed to CloudStreams, and CloudStreams executes the policy's run-time actions in the order in which they are specified in the policy. However, if the service already has another policy applied to it, a policy conflict might occur. A policy conflict might have unintended consequences. CloudStreams will warn you of policy conflicts. For more information, see the section *What Happens When You Deploy a Service?* in the document *Administering webMethods CloudStreams*.

General Properties View (Policy)

To edit General properties, click the policy name in the CloudStreams Governance **view.**

Name
(Read-only field.) The policy name. You can change the name by right-clicking it in the **Policies** folder and clicking **Rename**.

Status
(Read-only field). Indicates whether the policy is **Active** or **Inactive**. To activate/deactivate the policy, right-click the policy name in the CloudStreams Governance view and click **Active** or **Inactive**. You will not be allowed to activate the policy unless all of its action parameters have been set.

Virtual Service Type
(Read-only field). Virtual Service or Connector Virtual Service.

Description
You can change the description.

Criteria
Click the button next to this field to change the criteria for the services to which the policy applies. (Alternatively, right-click the policy name in the **Policies** folder and click **Criteria**.)

In the Criteria dialog box that appears, specify the criteria as follows:

1. In the **Criteria** field, specify a value (**SOAP** or **REST**) for the **Service Type** attribute. That is, specify whether the policy should apply to all SOAP services or all REST services.
2. Click the plus button next to the **Criteria** field to add a new row, and choose another criteria attribute (**Name** or **Description**) and choose an operator and value for it. For example, specify whether the policy should apply to all SOAP services that equals, starts with or contains the prefix `Abc_`.
3. Repeat step 2 if desired to specify additional criteria.
4. If you specify multiple criteria, use the **Condition** field to connect the criteria by the AND or OR operator. The default operator is AND.

Action: Authorize User

Note: Dependency requirement: A policy that includes this action must also include one of the following actions: Require HTTP Basic Authentication, Require WSS Username Token or Require SAML Token.

This action authorizes consumers against a list of users and/or a list of groups registered in the Integration Server on which CloudStreams is running.

To set the Authorize User action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, click **Authorize User** in the **Applied Actions** list, and set the following action parameters.

User
Authorizes consumers against a list of users who are registered in the Integration Server instance on which CloudStreams is running.

Group
Authorizes consumers against a list of groups who are registered in the Integration Server instance on which CloudStreams is running.

Note: By default, both of these input parameters are selected. If you de-select one of these parameters, the fields showing the list of users (or groups) is not displayed.

Action: Identify Consumer

This action specifies the kind of consumer identifier (IP address, HTTP authorization token, etc.) that CloudStreams will use to identify consumer applications. You can select only one identifier. Alternatively, this action provides an option to allow anonymous users to send requests, without restriction.

To set the Identify Consumer action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Identify Consumer** in the **Applied Actions** list, and set the following action parameters.

Anonymous Usage Allowed
Specifies whether to allow anonymous users to send requests, without restriction.

IP Address
Right-click the action name and click **Add IP Address** to identify consumer applications based on their originating IP addresses.

Host Name
Right-click the action name and click **Add Host Name** to identify consumer applications based on a host name.

HTTP Token
Right-click the action name and click **Add HTTP Token** if you want to use HTTP Basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header. CloudStreams authorizes the credentials against the list of users registered in the Integration Server on which CloudStreams is running. This type of consumer authentication is referred to as "preemptive authentication". If you want to use "preemptive authentication", you should also include the action "Require HTTP Basic Authentication" in the policy.

If you choose to omit "Require HTTP Basic Authentication", the client will be presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as "non-preemptive authentication".

Note: If you select the value **HTTP Token**, do not include the "Authorize Against Registered Consumers" action in the policy. This is an invalid combination.

WSS Header Token
Right-click the action name and click **Add WSS Header Token** to validate user names and passwords that are transmitted in the SOAP message header in the WSS Username

Token. If you select this option, you should also include the action **Require WSS Username Token** in the policy.

XPATH Token
Right-click the action name and click **Add XPATH Token** to validate consumer applications based on an XML element (represented by an XPATH expression you specify in the **XPATH to Identify Token** field).

Consumer Certificate
Right-click the action name and click **Add Consumer Certificate** to identify consumer applications based on information in a WSS X.509 certificate. If you select this option, you should also include the **Require X.509 Token** or the **Require Signing** action in the policy.

User ID
Right-click the action name and click **Add User ID** if you are applying the Identify Consumer action to a connector virtual service. You must select the **User ID** identifier; no other identifier is valid. **User ID** identifies consumer applications based on a list of users who are registered in the Integration Server on which CloudStreams is running. (You need to apply the Identify Consumer action to a connector virtual service if you apply the following actions to the connector virtual service: "Monitor Service Level Agreement" or "Throttling Traffic Optimization" (if you select its **Limit Traffic for Applications** option)).

Action: Include Timestamps

Note: Dependency requirement: A policy that includes this action must also include all of the following actions: Require SSL, Require Signing and Require Encryption.

This action requires that timestamps be included in the request header. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST virtual services or connector virtual services.

CloudStreams checks the timestamp value against the current time to ensure that the request is not an old message, which serves to protect your system against attempts at message tampering, such as replay attacks.

CloudStreams rejects the request if either of the following things happen:

- CloudStreams receives a timestamp that exceeds the time defined by the timestamp element.
- A timestamp element is not included in the request.

There are no input parameters.

Action: Log Invocation

This action logs request/response payloads. You can log the payloads in the database and/or send the payloads in the form of email alerts. This action also logs other

information about the request/response, including the service name, operation name, the Integration Server user, a timestamp, the response time, and more.

To edit the Log Invocation action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Log Invocation** in the **Applied Actions** list, and set the following action parameters.

Log Generation Frequency
Specifies how frequently to log the payload.

- **Always**: Log all requests and/or responses.
- **On Success**: Log only the successful responses and/or requests.
- **On Failure**: Log only the failed requests and/or responses.

Log the Following Payloads
Specifies whether to log all request payloads, all response payloads, or both.

- **Request**: Log all request payloads.
- **Response**: Log all response payloads.

Send Data To
By default, this action logs the payloads to the CloudStreams Analytics database.

> **Note:** Ensure that you select the **Database Publishing** option in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Database**), as described in the section *Setting the Database Options for Publishing Run-Time Metrics and Events* in the document *Administering webMethods CloudStreams*.

Alert Email
Right-click the action name and click **Add Alert Email** to send the payloads in an email alert to the email address you specify in the **Email ID** field. You can select **Add Alert Email** multiple times to add multiple email addresses.

> **Note:** Ensure that you select the email options in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Email**), as described in the section *Setting the Email Options for Logging Payloads and Sending Performance Monitoring Alerts* in the document *Administering webMethods CloudStreams*.

Action: Monitor Service Performance

This action monitors a user-specified set of run-time performance conditions for a virtual service, and sends alerts to a specified destination when the performance conditions are violated.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), CloudStreams sends an alert as soon as the performance condition is violated, without

having to wait until the end of the metrics tracking interval. CloudStreams sends only one alert the first time the condition is violated during the interval. (It will send another alert the next time a condition is violated during a subsequent interval.) For information about intervals, see the section *The Intervals for Metric Publishing* in the document *Administering webMethods CloudStreams*.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), CloudStreams aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

By default, this action does not include metrics for failed invocations. To include metrics for failed invocations, set the `pg.PgMetricsFormatter.includeFaults` parameter to true in *IntegrationServer_directory* \packages\WmCloudStreams\config\resources\wst-config.properties.

To set the Monitor Service Performance action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Monitor Service Performance** in the **Applied Actions** list, and set the following action parameters.

Alert Interval
Number. The time period (in minutes) in which to monitor performance before sending an alert if a condition is violated.

Alert Frequency
String. Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count).

- **Every Time**: Issues an alert every time one of the specified conditions is violated.
- **Only Once**: Issues an alert only the first time one of the specified conditions is violated.

Alert Message
Optional. Specify a text message to include in the alert.

Send Data To
By default, this action logs the alerts to the CloudStreams Analytics database.

> **Note:** Ensure that you select the **Database Publishing** option in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Database**), as described in the section *Setting the Database Options for Publishing Run-Time Metrics and Events* in the document *Administering webMethods CloudStreams*.

Metrics Collection Level
The run-time performance metrics for a virtual service (which is invoked only in the inbound run-time scenario), are collected at the service level. That is, the metrics for all invocations of a single virtual services are aggregated together during your specified metrics publishing interval and then published.

In contrast, the metrics for a connector virtual service (which is invoked only in the outbound run-time scenario) can be collected at two different levels of metric collection:

- **Cloud Connector Service**: Remember that a single connector virtual service can be used by multiple cloud connector services. Select this option if you want to collect the metrics for the connector virtual service broken down by each separate cloud connector service that uses it. For example, if a connector virtual services is used by three cloud connector services, then this option will collect the metrics of that service separately, broken down by each of the three cloud connector services that use it.
- **Connector Virtual Service** (default): Select this option if you want to aggregate all the metrics for a single connector virtual service, even if it is used by multiple cloud connector services. For example, if a connector virtual service is used by three cloud connector services, then this option will collect the combined metrics for the connector virtual service by all three of the cloud connector services that use it.

Action Configuration

Right-click the action name and click **Add Action Configuration** to specify a condition to monitor. To do this, select a condition **Name** (the metric to monitor), an **Operator**, and a **Value** for the condition. You can select **Add Action Configuration** multiple times to add multiple conditions. Multiple conditions are connected by the AND operator.

Name: The metric to monitor, which can be:

- **Availability**: Indicates whether the service was available to the specified consumers in the current interval. A value of 100 indicates that the service was always available. If invocations fail due to policy violations, this parameter could still be as high as 100. That is, SOAP faults returned by the native provider or faults due to CloudStreams policy enforcements do not impact Availability. Only errors that CloudStreams interprets as a provider service being down will impact Availability.
- **Average Response**: The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment CloudStreams receives the request until the moment it returns the response to the caller.
- **Fault Count**: The number of faults returned in the current interval.
- **Maximum Response** : The maximum amount of time to respond to a request in the current interval.
- **Minimum Response**: The minimum amount of time to respond to a request in the current interval.
- **Successful Request Count**: The number of successful requests in the current interval.
- **Total Request Count**: The total number of requests (successful and unsuccessful) in the current interval.

Alert Email

Right-click the action name and click **Add Alert Email** if you want to send the monitoring alerts to an email address you specify in the **Email ID** field. You can select **Add Alert Email** multiple times to add multiple email addresses.

> **Note:** Ensure that you select the email options in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Email**), as described in the section *Setting the Email Options for Logging Payloads and Sending Performance Monitoring Alerts* in the document *Administering webMethods CloudStreams*.

Action: Monitor Service Level Agreement (SLA)

> **Note:** Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

This action is similar to the Monitor Service Performance action. Both actions can monitor the same set of run-time performance conditions for a virtual service, and then send alerts when the performance conditions are violated. This action is different because it enables you to monitor run-time performance for *one or more specified consumers*.

You can configure this action to define a *Service Level Agreement* (SLA), which is set of conditions that defines the level of performance that a consumer should expect from a service. You can use this action to identify whether a service's threshold rules are met or exceeded. For example, you might define an agreement with a particular consumer that sends an alert to the consumer if responses are not sent within a certain maximum response time. You can configure SLAs for each virtual service/consumer application combination.

For the counter-based metrics (Total Request Count, Success Count, Fault Count), CloudStreams sends an alert as soon as the performance condition is violated, without having to wait until the end of the metrics tracking interval. You can choose whether to send an alert only once during the interval, or every time the violation occurs during the interval. (CloudStreams will send another alert the next time a condition is violated during a subsequent interval.) For information about intervals, see the section *The Intervals for Metric Publishing* in the document *Administering webMethods CloudStreams*.

For the aggregated metrics (Average Response Time, Minimum Response Time, Maximum Response Time), CloudStreams aggregates the response times at the end of the interval, and then sends an alert if the performance condition is violated.

By default, this action does not include metrics for failed invocations. To include metrics for failed invocations, set the `pg.PgMetricsFormatter.includeFaults` parameter to true in *IntegrationServer_directory* \packages\WmCloudStreams\config\resources\wst-config.properties.

To set the Monitor Service Level Agreement (SLA) action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Monitor Service Level Agreement (SLA)** in the **Applied Actions** list, and set the following action parameters.

Alert Interval
Number. The time period (in minutes) in which to monitor performance before sending an alert if a condition is violated.

Alert Frequency
String. Specifies how frequently to issue alerts for the counter-based metrics (Total Request Count, Success Count, Fault Count).

- **Every Time**: Issues an alert every time one of the specified conditions is violated.
- **Only Once**: Issues an alert only the first time one of the specified conditions is violated.

Alert Message
Optional. Specify a text message to include in the alert.

Send Data To
By default, this action logs the alerts to the CloudStreams Analytics database.

> **Note:** Ensure that you select the **Database Publishing** option in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Database**), as described in the section *Setting the Database Options for Publishing Run-Time Metrics and Events* in the document *Administering webMethods CloudStreams*.

Metrics Collection Level
The run-time performance metrics for a virtual service (which is invoked only in the inbound run-time scenario), are collected at the service level. That is, the metrics for all invocations of a single virtual services are aggregated together during your specified metrics publishing interval and then published.

In contrast, the metrics for a connector virtual service (which is invoked only in the outbound run-time scenario) can be collected at two different levels of metric collection:

- **Cloud Connector Service**: Remember that a single connector virtual service can be used by multiple cloud connector services. Select this option if you want to collect the metrics for the connector virtual service broken down by each separate cloud connector service that uses it. For example, if a connector virtual services is used by three cloud connector services, then this option will collect the metrics of that service separately, broken down by each of the three cloud connector services that use it.
- **Connector Virtual Service** (default): Select this option if you want to aggregate all the metrics for a single connector virtual service, even if it is used by multiple cloud connector services. For example, if a connector virtual service is used by three cloud connector services, then this option will collect the combined metrics for the connector virtual service by all three of the cloud connector services that use it.

Action Configuration

Right-click the action name and click **Add Action Configuration** to specify a condition to monitor. To do this, select a condition **Name** (the metric to monitor), an **Operator**, and a **Value** for the condition. You can select **Add Action Configuration** multiple times to add multiple conditions. Multiple conditions are connected by the AND operator.

Name: The metric to monitor, which can be:

- **Availability**: Indicates whether the service was available to the specified consumers in the current interval. A value of 100 indicates that the service was always available. If invocations fail due to policy violations, this parameter could still be as high as 100. That is, SOAP faults returned by the native provider or faults due to CloudStreams policy enforcements do not impact Availability. Only errors that CloudStreams interprets as a provider service being down will impact Availability.
- **Average Response**: The average amount of time it took the service to complete all invocations in the current interval. Response time is measured from the moment CloudStreams receives the request until the moment it returns the response to the caller.
- **Fault Count**: The number of faults returned in the current interval.
- **Maximum Response** : The maximum amount of time to respond to a request in the current interval.
- **Minimum Response**: The minimum amount of time to respond to a request in the current interval.
- **Successful Request Count**: The number of successful requests in the current interval.
- **Total Request Count**: The total number of requests (successful and unsuccessful) in the current interval.

Alert for Applications

Right-click the action name and click **Add Alert for Applications** to specify the consumer application to which this Service Level Agreement will apply. You can select **Add Alert for Applications** multiple times to add multiple consumer applications.

Alert Email

Right-click the action name and click **Add Alert Email** if you want to send the monitoring alerts to an email address you specify in the **Email ID** field. You can select **Add Alert Email** multiple times to add multiple email addresses.

> **Note:** Ensure that you select the email options in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Email**), as described in the section *Setting the Email Options for Logging Payloads and Sending Performance Monitoring Alerts* in the document *Administering webMethods CloudStreams*.

Action: Require Encryption

This action requires that an XML element (which is represented by an XPath expression) be encrypted. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST virtual services or connector virtual services.

Prerequisites:

1. Configure Integration Server: Set up keystores and truststores in Integration Server (see the section *Securing Communications with the Server* in the document *webMethods Integration Server Administrator's Guide*).
2. Configure CloudStreams: In the Integration Server Administrator, navigate to **Solutions > CloudStreams > Administration > General** and complete the **IS Keystore Name**, **IS Truststore Name** and **Alias (signing)** fields, as described in the section *Setting the General Options* in the document *Administering webMethods CloudStreams*).

When this policy action is set for the virtual service, CloudStreams provides decryption of incoming requests and encryption of outgoing responses. CloudStreams can encrypt and decrypt only individual elements in the SOAP message body that are defined by the XPath expressions configured for the policy action. CloudStreams requires that requests contain the encrypted elements that match those in the XPath expression. You must encrypt the entire element, not just the data between the element tags. CloudStreams rejects requests if the element name is not encrypted.

Note: Do not encrypt the entire SOAP body because a SOAP request without an element will appear to CloudStreams to be malformed.

CloudStreams attempts to encrypt the response elements that match the XPath expressions with those defined for the policy. If the response does not have any elements that match the XPath expression, CloudStreams will not encrypt the response before sending. If the XPath expression resolves a portion of the response message, but CloudStreams cannot locate a certificate to encrypt the response, then CloudStreams sends a SOAP fault exception to the consumer and a Policy Violation event notification to CloudStreams.

How CloudStreams Encrypts Responses

The Require Encryption action encrypts the response back to the client by dynamically setting a public key alias at run time. CloudStreams determines the public key alias as follows:

1. If CloudStreams can access the X.509 certificate of the client (based on the incoming request signature), it will use "useReqSigCert" as the public key alias.

   OR

2. If the Identify Consumer action is present in the policy (and it successfully identifies a consumer application), then CloudStreams will look for a public key alias with that consumer name in the "IS Keystore Name" property. The "IS Keystore Name" property is specified in the Integration Server Administrator, under **Solutions >**

CloudStreams > Administration > General. This property should be set to an Integration Server keystore that CloudStreams will use.

For an Identify Consumer action that allows for anonymous usage, CloudStreams does not require a consumer name in order to send encrypted responses. In this case, CloudStreams can use one of the following to encrypt the response in the following order, depending on what is present in the security element:

a. A signing certificate.

b. Consumer name.

c. WSS username, SAML token or X.509 certificate.

d. HTTP authorized user.

OR

3. If CloudStreams can determine the current IS user from the request (that is, if an Integration Server WS-Stack determined that Subject is present), then the first principal in that subject is used.

   OR

4. If the above steps all fail, then CloudStreams will use either the WS-Security username token or the HTTP Basic-Auth user name value. There should be a public key entry with the same name as the identified username.

To set the Require Encryption action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Require Encryption** in the **Applied Actions** list, and set the following action parameters.

Element Required To Be Encrypted

An XPath expression that represents the XML element that is required to be encrypted.

Namespace Prefix

Optional. Right-click the action name and click **Add Namespace Prefix** if you want to specify the namespace prefix of the element required to be encrypted. Enter the namespace prefix in the following format:

`xmlns:<prefix-name>`

For example: `xmlns:soapenv`. For more information, see the XML Namespaces specifications at http://www.w3.org/TR/REC-xml-names/#ns-decl.

The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
            <sp:XPath
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">//soapenv:Body</sp
:XPath>
          </sp:SignedElements>
```

Action: Require HTTP Basic Authentication

This action uses HTTP Basic authentication to verify the consumer's authentication credentials contained in the request's Authorization header. CloudStreams authorizes the credentials against the list of users registered in the Integration Server on which CloudStreams is running. This type of consumer authentication is referred to as "preemptive authentication". If you want to perform "preemptive authentication", a policy that includes this action must also include the Identify Consumer action. This action supports WS-SecurityPolicy 1.2.

If the user/password value in the Authorization header cannot be authenticated as a valid Integration Server user (or if the Authorization header is not present in the request), a 500 SOAP fault is returned, and the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated. This type of consumer authentication is referred to as "non-preemptive authentication". If the client does not successfully respond to the challenge, a 401 "WWW-Authenticate: Basic" response is returned and the invocation is not routed to the policy engine. As a result, no events are recorded for that invocation, and its key performance indicator (KPI) data are not included in the performance metrics

If you choose to omit the "Require HTTP Basic Authentication" action (regardless of whether an Authorization header is present in the request or not), then:

- CloudStreams forwards the request to the native service, without attempting to authenticate the request.
- The native service returns a 401 "WWW-Authenticate: Basic" response, which CloudStreams will forward to the client; the client is presented with a security challenge. If the client successfully responds to the challenge, the user is authenticated.

In the case where a consumer is sending a request with both transport credentials (HTTP basic authentication) and message credentials (WSS username or X.509 token), the message credentials take precedence over the transport credentials when Integration Server is determining which credentials it should use for the session. For more information, see "Action: Require WSS Username" on page 48, and "Action: Require X.509 Token" on page 48. In addition, you must ensure that the service consumer that connects to the virtual service has an Integration Server user account.

To set the Require HTTP Basic Authentication action parameter

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Require HTTP Basic Authentication** in the **Applied Actions** list, and set the following action parameter.

Authenticate Credentials
Authorizes consumers against the list of users registered in the Integration Server on which CloudStreams is running. If you select this option, you must also include the Identify Consumer action in the policy.

Action: Require SAML Token

Requires that a WSS Security Assertion Markup Language (SAML) assertion token be present in the SOAP message header to validate service consumers. CloudStreams supports SAML 1.1 and 2.0 tokens. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST virtual services or connector virtual services.

Note: When a Require SAML Token action is generated, CloudStreams also implicitly selects the "timestamp" and "signing" assertions. You should not add the Require Timestamps and Require Signing actions to a virtual service if the Require SAML Token action is already applied.

To set the Require SAML Token action parameter

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Require SAML Token** in the **Applied Actions** list, and set the following action parameter.

SAML Version
Specifies the version of the WSS SAML Token to use.

- **SAML 1.1**: Default.
- **SAML 2.0**

Note: For important usage information, see the section *Require SAML Token* in the document *Administering webMethods CloudStreams*

Action: Require Signing

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

This action requires that an XML element (represented by an XPath expression) be signed. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST virtual services or connector virtual services.

Prerequisites:

1. Configure Integration Server: Set up keystores and truststores in Integration Server (see the section *Securing Communications with the Server* in the document *webMethods Integration Server Administrator's Guide*).
2. Configure CloudStreams: In the Integration Server Administrator, navigate to **Solutions > CloudStreams > Administration > General** and complete the **IS Keystore Name**, **IS Truststore Name** and **Alias (signing)** fields, as described in the section *Setting the General Options* in the document *Administering webMethods CloudStreams*). CloudStreams uses the signing alias specified in the **Alias (signing)** field to sign the response.

When this policy action is set for the virtual service, CloudStreams validates that the requests are properly signed, and provides signing for responses. CloudStreams provides support both for signing an entire SOAP message body or individual elements of the SOAP message body.

CloudStreams uses a digital signature element in the security header to verify that all elements matching the XPath expression were signed. If the request contains elements that were not signed or no signature is present, then CloudStreams rejects the request.

Note: You must map the public certificate of the key used to the sign the request to an Integration Server user. If the certificate is not mapped, CloudStreams returns a SOAP fault to the caller.

To set the Require Signing action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Require Signing** in the **Applied Actions** list, and set the following action parameters.

Element Required To Be Signed
An XPath expression that represents the XML element that is required to be signed.

Namespace Prefix
Optional. Right-click the action name and click **Add Namespace Prefix** if you want to specify the namespace prefix of the element required to be encrypted. Enter the namespace prefix in the following format:

`xmlns:<prefix-name>`

For example: `xmlns:soapenv`. For more information, see the XML Namespaces specifications at http://www.w3.org/TR/REC-xml-names/#ns-decl.

The generated XPath element in the policy should look similar to this:

```
<sp:SignedElements xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702">
            <sp:XPath
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">//soapenv:Body</sp
:XPath>
          </sp:SignedElements>
```

Action: Require SSL

This action ensures that requests are sent to the server using the HTTPS protocol (SSL). This action supports WS-SecurityPolicy 1.2 and can be used by both SOAP and REST services.

In addition, setting the **Client Certificate Required** parameter to True allows CloudStreams to verify the client sending the request.

Ensure that you specify an HTTPS port in the Integration Server Administrator (go to **CloudStreams > Administration > General**).

To set the Require SSL action parameter

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Require SSL** in the **Applied Actions** list, and set the following action parameter.

Client Certificate Required
Specifies whether client certificates are required for the purposes of:

- Verifying the signature of signed SOAP requests or decrypting encrypted SOAP requests.
- Signing SOAP responses or encrypting SOAP responses.

Values:

- **True**: Requires client certificates. If a valid client certificate is not presented, CloudStreams rejects the message. Ensure that the Integration Server HTTPS port is configured to request or require a client certificate.
- **False**: Default. Does not require client certificates.

Action: Require WSS Username

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

Requires that a WSS username token and password be present in the SOAP message header to validate service consumers. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST virtual services or connector virtual services.

CloudStreams rejects requests that do not include the username token and password of an Integration Server user. CloudStreams only supports clear text passwords with this kind of authentication.

In the case where a consumer is sending a request with both transport credentials (HTTP basic authentication) and message credentials (WSS username or X.509 token), the message credentials take precedence over the transport credentials when Integration Server is determining which credentials it should use for the session.

There are no input parameters.

Action: Require X.509 Token

Note: Dependency requirement: A policy that includes this action must also include the Identify Consumer action.

Requires that a WSS X.509 token be present in the SOAP message header to validate service consumers. This action supports WS-SecurityPolicy 1.2 and cannot be used with REST virtual services or connector virtual services.

In the case where a consumer is sending a request with both transport credentials (HTTP basic authentication) and message credentials (X.509 token or WSS username), the message credentials take precedence over the transport credentials when Integration Server is determining which credentials it should use for the session. In addition, you must ensure that the service consumer that connects to the virtual service has an Integration Server user account.

There are no input parameters.

Action: Throttling Traffic Optimization

> **Note:** Dependency requirement: A policy that includes this action must also include the Identify Consumer action if the **Limit Traffic for Applications** option is selected.

This action 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.

Reasons for limiting the service invocation traffic include:

- To avoid overloading the back-end services and their infrastructure.
- To limit specific consumers in terms of resource usage (that is, you can use the "Monitor Service Level Agreement" (SLA) action in addition to "Throttling Traffic Optimization").
- To shield vulnerable servers, services, and even specific operations.
- For service consumption metering (billable pay-per-use services).

To set the Throttling Traffic Optimization action parameters

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Throttling Traffic Optimization** in the **Applied Actions** list, and set the following action parameters.

Soft Limit
Optional. Specifies the maximum number of invocations allowed per **Interval** before issuing an alert. Reaching the soft limit will not affect further processing of requests (until the **Hard Limit** is reached).

> **Note:** The limit is reached when the total number of invocations coming from all the consumer applications (specified in the **Limit Traffic for Applications** field) reaches the limit.

Hard Limit
Required. Specifies the maximum number of invocations allowed per **Interval** before stopping the processing of further requests and issuing an alert. Typically, this limit should be higher than the soft limit.

> **Note:** The limit is reached when the total number of invocations coming from all the consumer applications (specified in the **Limit Traffic for Applications** field) reaches the limit.

Alert Interval
Specifies the amount of time for the soft limit and hard limit to be reached.

Alert Frequency
Specifies how frequently to issue alerts.

- **Every Time**: Issue an alert every time one of the specified conditions is violated.
- **Only Once**: Issue an alert only the first time one of the specified conditions is violated.

Alert Message For Soft Limit
Optional. Specify a text message to include in the soft limit alert.

Alert Message For Hard Limit
Optional. Specify a text message to include in the hard limit alert.

Send Data To
By default, this action logs the alerts to the CloudStreams Analytics database.

> **Note:** Ensure that you select the **Database Publishing** option in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Database**), as described the section *Setting the Database Options for Publishing Performance Metrics and Events* in the document *Administering webMethods CloudStreams*.

Metrics Collection Level
The run-time performance metrics for a virtual service (which is invoked only in the inbound run-time scenario), are collected at the service level. That is, the metrics for all invocations of a single virtual services are aggregated together during your specified metrics publishing interval and then published.

In contrast, the metrics for a connector virtual service (which is invoked only in the outbound run-time scenario) can be collected at two different levels of metric collection:

- **Cloud Connector Service**: Remember that a single connector virtual service can be used by multiple cloud connector services. Select this option if you want to collect the metrics for the connector virtual service broken down by each separate cloud connector service that uses it. For example, if a connector virtual services is used by three cloud connector services, then this option will collect the metrics of that service separately, broken down by each of the three cloud connector services that use it.
- **Connector Virtual Service** (default): Select this option if you want to aggregate all the metrics for a single connector virtual service, even if it is used by multiple cloud connector services. For example, if a connector virtual service is used by three cloud connector services, then this option will collect the combined metrics for the connector virtual service by all three of the cloud connector services that use it.

Limit Traffic For Applications
Specifies the consumer application(s) that this action applies to. To specify multiple consumer applications, use the plus button to add rows.

Alert Email
Right-click the action name and click **Add Alert Email** if you want to send the monitoring alerts to an email address you specify in the **Email ID** field. You can select **Add Alert Email** multiple times to add multiple email addresses.

Note: Ensure that you set the email options in Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Email**), as described in *Setting the Email Options for Logging Payloads and Sending Performance Monitoring Alerts* in the document *Administering webMethods CloudStreams*.

Action: Validate Schema

This action validates all XML request and/or response messages against an XML schema referenced in the WSDL. This action cannot be used with connector virtual services.

CloudStreams can enforce this policy action for messages sent between services. When this policy is set for the virtual service, CloudStreams validates XML request messages, response messages, or both, against the XML schema referenced in the WSDL. Ensure that you provide a schema.

To set the Validate Schema action parameter

1. In the CloudStreams Governance view, click the policy name.
2. In the policy editor on the right side of the page, double-click **Validate Schema** in the **Applied Actions** list, and set the following action parameter.

Validate SOAP Message(s)
Validates the request and/or response messages.

Note: CloudStreams does not remove `wsu:Id` attributes that may have been added to a request by a consumer as a result of security operations against request elements (signatures and encryptions). In this case, to avoid schema validation failures you would have to add a Transform sub-step to the virtual service's In Sequence step so that the requests are passed to an XSLT transformation file that removes the `wsu:Id`s.

Virtual Services (SOAP)

Use this editor to create and modify SOAP-based virtual services. *Virtual services* handle requests in the inbound processing scenario. When a SaaS application sends a service request, a virtual service performs security checks and other user-defined processing before sending the request to the on-premise application.

A virtual service runs on CloudStreams and acts as the consumer-facing proxy for a native service running in an on-premise application. A virtual service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing independence (of location, protocol and format) between the consuming application and the provider service. You should create a virtual service for each native service you want to expose to consumers.

When you create a virtual service, you can configure the following processing steps for the service:

- The **In Sequence step**, which you configure to manipulate the request messages. This step can include the following sub-steps:
  - The **Entry step** (provided by default), which specifies the protocol (HTTP or HTTPS) and SOAP format (1.1 or 1.2) of the requests that the virtual service will accept.
  - The **Transform step** (optional), which performs an XSLT message transformation on the request message before the virtual service submits it to the native service.
  - The **Invoke IS Service step** (optional), which pre-processes the request message before the virtual service submits it to the native service.
  - The **Routing Rule step** (provided by default), which specifies how the virtual service will route the requests to the native service endpoint. There are four ways to route HTTP or HTTPS requests:
    - "Straight Through" routing (to route requests directly to the native service endpoint).
    - "Context-Based" routing (to route specific types of messages to specific endpoints according to context-based routing rules).
    - "Content-Based" routing (to route specific types of messages to specific endpoints based on specific values that appear in the request message).
    - "Load Balancing" routing (to distribute requests among multiple endpoints).
- The service's **Out Sequence step**, which you configure to manipulate the response messages. This step can include the following sub-steps:
  - The **Transform step** (optional), which specifies how the response message from the native service provider is to be transformed before the virtual service returns it to the consuming application.
  - The **Invoke IS Service step** step (optional), which pre-processes the response message before the virtual service returns it to the consuming application.
- The service's **Error Sequence step** (provided by default). CloudStreams returns a default fault response to the consuming application, which you can customize with context variables. This fault response is used for faults returned by the native service provider as well as faults returned by internal CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors). In addition, you can:

- Choose whether or not to send the native service provider's service fault content, or just send the response message.
- Invoke IS services to pre-process or post-process the error messages.

New Virtual Service Wizard (SOAP)

To open the New Virtual Service wizard:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, right-click the CloudStreams Governance project and click **New > Virtual Service** (or expand the project, right-click the **Virtual Services** folder and click **New Virtual Service**).

Complete the following fields in the New Virtual Service wizard:

Project
The CloudStreams Governance project in which you are creating the virtual service.

Name
Assign a name for the service. Unlike native services, the names of virtual services cannot contain spaces or special characters except _ (underscore) and - (hyphen). Consequently, if you adopt a convention that involves using the name of the service as part of the virtual service name, then the names of the services themselves must not contain characters that are invalid in virtual service names.

Version
The version is always set to 1.0.

Type
Select **SOAP**.

WSDL
The WSDL that the virtual service uses. Select either **File** (and click **Browse** to select a WSDL) or select **URL** (and enter the URL of the WSDL). If you need to add a WSDL to the service later, you can leave the WSDL field blank and add the WSDL later. See "Attach WSDL Dialog Box (SOAP Virtual Service)" on page 69).

Description
Optional. A description for the virtual service. This description appears when a user displays a list of virtual services in the user interface.

General Properties (SOAP Virtual Service)

To display the general properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.

2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**). Click the virtual service name and then the "General" tab.

Name
(Read-only field.) The service name. You can change the service name at any time by right-clicking the name in the **Virtual Services** folder and clicking **Rename**.

Service Type
(Read-only field.) **SOAP**.

Created/Last Modified
(Read-only field.) The service's creation/modification timestamps.

Target Namespace
(Read-only field.) This value is derived from the `targetNamespace` attribute of the WSDL's definition element.

Namespaces
Click the button next to this field to view the virtual service's available namespaces (see "Virtual Service Namespaces Dialog Box (SOAP Virtual Service)" on page 55).

Version
The version is always set to 1.0.

WSDL URL
Go to the "Advanced" tab and then click this URL to display the contents of the service's abstract WSDL. If a WSDL file was not added, this will be empty. You can override the WSDL by attaching a new one (see "Attach WSDL Dialog Box (SOAP Virtual Service)" on page 69).

Description
You can change the service description in the "General" tab.

Advanced Properties (SOAP Virtual Service)

To display the advanced properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.

2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**). Click the virtual service name and then the "Advanced" tab.

Name
(Read-only field.) The service name. You can change the name of an undeployed service by right-clicking the name in the **Virtual Services** folder and clicking **Rename**.

Type
(Read-only field.) The type of the service (SOAP).

Target Namespace
(Read-only field.) The value is derived from the `targetNamespace` attributes of the WSDL's definition element.

WSDL URL
Click this URL to display the contents of the service's abstract WSDL. You can override the WSDL by attaching a new one (see "Attach WSDL Dialog Box (SOAP Virtual Service)" on page 69).

Namespaces
Click the button next to this field to view the virtual service's available namespaces (see "Virtual Service Namespaces Dialog Box (SOAP Virtual Service)" on page 55).

Version
The version is always set to 1.0.

Description
(Read-only field.) The service description.

VSD
Click the button next to this field to view the virtual service definition (VSD). For details, see "VSD Dialog Box (SOAP Virtual Service)" on page 55.

Deployed Status
(Read-only field.) Indicates whether the service is **Deployed**, **Undeployed** or **Not Deployed** (which is the initial status before you deploy the service).

Applicable Policies
Click the button next to this field to view a list of all active policies that apply to this service. Any inactive policy that applies to the service is not listed. For details, see "Applicable Policies Dialog Box (SOAP Virtual Service)" on page 56.

Endpoint
Click the button next to this field to view the endpoint to which the virtual service is deployed, if applicable (see "Endpoint Dialog Box (SOAP Virtual Service)" on page 56).

Virtual Service Namespaces Dialog Box (SOAP Virtual Service)

This dialog box displays the virtual service's available namespaces. The namespace prefixes (such as `wsdl`, `tns`, `xsd`, `soap`, and so on) and names are displayed.

VSD Dialog Box (SOAP Virtual Service)

This dialog box displays the virtual service definition (VSD) that CloudStreams generates when you deploy a virtual service to a CloudStreams server target.

CloudStreams generates an XML document called a virtual service definition (VSD). The VSD defines the virtual service for CloudStreams, and contains all the resources required to deploy the virtual service to CloudStreams, including the policy that applies to the service.

Note: If multiple policies apply to the service, CloudStreams combines all those policies into a single policy known as the *effective policy*. The effective policy is a simple UNION of the run-time actions specified in all policies that apply to a service. To create the effective policy, CloudStreams evaluates the combined list of actions from all policies, using a set of internal rules known as Policy Conflict Resolution rules. For details, see the topic *Policy Conflict Resolution Rules* in the document *Administering webMethods CloudStreams*.

Applicable Policies Dialog Box (SOAP Virtual Service)

This dialog box lists all active policies that apply to this service. Any inactive policy that applies to the service is not listed.

Note: If multiple policies apply to the service, CloudStreams combines all those policies into a single policy known as the *effective policy*. The effective policy is a simple UNION of the run-time actions specified in all policies that apply to a service. To create the effective policy, CloudStreams evaluates the combined list of actions from all policies, using a set of internal rules known as Policy Conflict Resolution rules. For details, see the topic *Policy Conflict Resolution Rules* in the document *Administering webMethods CloudStreams*.

Endpoint Dialog Box (SOAP Virtual Service)

This dialog box displays the virtual service's CloudStreams endpoint (not the native service endpoint).

Entry Step (SOAP Virtual Service)

The Entry Step specifies the protocol (HTTP or HTTPS) 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 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.

To configure the Entry Step:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.

3. Click **Entry Step** and complete the following fields in the **General** page in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming restrictions.

Type
(Read-only field.) **Entry Step**.

Protocol
The protocol (HTTP or HTTPS) over which the virtual service will accept requests. To specify HTTPS, select both **HTTP** and **SSL**.

Format
The SOAP format (SOAP 1.1 or SOAP 1.2) of the requests that the virtual service will accept.

Transform Step (SOAP Virtual Service)

Add the optional Transform step if you need to transform the request message into the format required by the native service, before CloudStreams submits the message to the native service.

No message transformation is required as long as the request message structure matches the request message structure that is required by the operation associated with the `soapAction`. 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 Transform step to transform the record submitted by the consuming application to the structure required by the Web service.

In this case, you would need to create two Transform steps:

- One in the "In Sequence" step, to transform the request messages into the format required by the native service, before CloudStreams sends the requests to the native services. To do this, you pass the message to an XSLT transformation file. (Additionally in this case, the transformation is required if the virtual service has a schema validation policy that validates the requests.)
- One in the "Out Sequence" step, to transform the native service's response messages into the format required by the consumer applications, before CloudStreams returns the responses to the consumer applications.

Transform Step (Inbound, SOAP Virtual Service)

To add the Transform step (inbound):

1. Click the virtual service name in the CloudStreams Governance view.
2. Right-click **In Sequence** and click **Transform**.

   The "Transform" step is added under the Entry Step. You cannot change the order of the steps.
3. Click **Transform** and complete the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming restrictions.

XSLT Service
The XSLT file to transform the request message before it is submitted to the native service. Click **Browse** to select a file from your file system and click **Save**.

The XSL file uploaded by the user should not contain the XML declaration in it (`xml version="1.0" encoding="UTF-8"`). This is because when the virtual service is deployed to CloudStreams, CloudStreams 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.

> **Note:** If you make changes to the XSLT file in the future, you must re-deploy the virtual service.

Invoke IS Service Step (Inbound, SOAP Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "In Sequence" step to pre-process the request message before it is submitted to the native service. You can create multiple "Invoke IS Service" steps.

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

You can use the following constructs in an IS service:

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

To add the Invoke IS Service step:

1. Click the virtual service name in the CloudStreams Governance view.
2. Right-click **In Sequence** and click **Invoke IS Service**.

The "Invoke IS Service" step is added under the Entry Step. You cannot change the order of the steps.

3. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming restrictions.

Service
The Integration Server service to pre-process the request message before it is submitted to the native service. Click **Browse** to select a file from your file system.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `MessageContext`: CloudStreams will automatically place a `MessageContext` variable into the pipeline before executing the 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 (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 Serverpub.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*.

Routing Rule Step (Straight Through Routing, SOAP Virtual Service)

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).

To configure the Routing Rule step for Straight Through routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **Straight Through**.

Default To
The URL of the service to which the request is to be routed.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows:

- **Optimization Method**: Select one of the following options:
  - **None**: The default.
  - **MTOM**: Indicates that CloudStreams 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 CloudStreams expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service.

  Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, CloudStreams 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 CloudStreams from a native provider will be forwarded to the caller using the same format it received.

  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's `Accept` header must be set to `multipart/related` (or the virtual service's "In Sequence" step should include an IS service callout that sets the `BUILDER_TYPE` context variable to `multipart/related`). This is necessary so CloudStreams knows how to parse the response properly.
- **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), CloudStreams will use the default value specified in Integration Server.
- **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), the default 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.

  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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Routing Rule Step (Context-Based Routing, SOAP Virtual Service)

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

A routing rule specifies where the 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. For example, you might use this capability to route requests from certain high-priority consumers to endpoints on a fast machine.

To configure the Routing Rule step for Context-Based routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **Context-Based**.

Rule Name
Assign a name to the rule.

Then, click the icon next to this field and complete the Configure Routing Rule dialog box as follows:

- Choose one of the following: **Time**, **IP Address** (IPv4 or IPv6 format), **Date**, **Consumer**, **Predefined Context Variable** or **Custom Context Variable** (see the section *Using Context Variables* in the document *Administering webMethods CloudStreams*).

  Note: If you select the value **Custom Context Variable**, you must write an IS service to get/set the custom context variable, and then specify the service in the **IS Service Name** field on the **Routing Rules** page. CloudStreams provides an API to get/set custom context variables. For more information, see the section *The API for Context Variables* in the document *Administering webMethods CloudStreams*. CloudStreams automatically declares any custom context variables you have specified; there is no need for you to declare them.

Route To
Enter the URL of the native service to route the request to, if the rule criteria are met.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows:

- **Optimization Method**: Select one of the following options:
  - **None**: The default.
  - **MTOM**: Indicates that CloudStreams 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 CloudStreams expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service.

  Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, CloudStreams 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 CloudStreams from a native provider will be forwarded to the caller using the same format it received.

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's `Accept` header must be set to `multipart/related` (or the virtual service's "In Sequence" step should include an IS service callout that sets the `BUILDER_TYPE` context variable to `multipart/related`). This is necessary so CloudStreams knows how to parse the response properly.

- **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), CloudStreams will use the default value specified in Integration Server.
- **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), the default 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.

  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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

Default To
Enter a native service endpoint to route the request to in case all routing rules evaluate to False.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box, as described for the **Route To** field above.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Routing Rule Step (Content-Based Routing, SOAP Virtual Service)

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

To configure the Routing Rule step for Content-Based routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **Content-Based**.

Rule Name
Assign a name to the rule.

XPath
Create an XPath expression as follows:

1. Click the icon next to this field to display the XPath Editor.
2. In the XPath Editor that appears, view the **Namespace** tab, which displays all predefined namespaces. If you want to add custom namespaces, click **Add Custom Namespace/prefix**, specify a name and value for the namespace, and click **OK**. To add additional rows, use the plus button at the end of the row to add them.

3. In the XPath Editor's **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 **Evaluator** tab, evaluate the XPath expression by specifying an argument in the **XPath Expression** field, and clicking **Evaluate**.

   The true/false result of the evaluation is displayed in the **Result** field.

To specify additional XPath expressions, use the plus button at the end of the row to add them.

> **Note:** Currently, you cannot define namespace URIs. So the XPath expressions should not include a namespace prefix.

Route To

Specify where to route the request if the rule criteria are met. Specify either the URL of the native service or a connection pool name.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows:

- **Optimization Method**: Select one of the following options:
  - **None**: The default.
  - **MTOM**: Indicates that CloudStreams 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 CloudStreams expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service.

  Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, CloudStreams 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 CloudStreams from a native provider will be forwarded to the caller using the same format it received.

  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's `Accept` header must be set to `multipart/related` (or the virtual service's "In Sequence" step should include an IS service callout that sets the `BUILDER_TYPE` context variable to `multipart/related`). This is necessary so CloudStreams knows how to parse the response properly.
- **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), CloudStreams will use the default value specified in Integration Server.
- **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), the default 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.

  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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

Default To
Enter a native service endpoint to route the request to in case all routing rules evaluate to False. Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box, as described for the **Route To** field above.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Routing Rule Step (Load Balancing Routing, SOAP Virtual Service)

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.

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 Interval** 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 the Routing Rule page for Load Balancing routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **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 **Endpoint** and select the URL of the Web service to route the request to.

To specify additional services, use the plus button next to the field to add rows.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows. These properties will apply to all the endpoints.

- **Optimization Method**: Select one of the following options:
  - **None**: The default.
  - **MTOM**: Indicates that CloudStreams 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 CloudStreams expects to receive a "SOAP with Attachment" (SwA) request, and will forward the attachment to the native service.

  Bridging between SwA and MTOM is not supported. If a consumer sends an SwA request, CloudStreams 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 CloudStreams from a native provider will be forwarded to the caller using the same format it received.

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's `Accept` header must be set to `multipart/related` (or the virtual service's "In Sequence" step should include an IS service callout that sets the `BUILDER_TYPE` context variable to `multipart/related`). This is necessary so CloudStreams knows how to parse the response properly.

- **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), CloudStreams will use the default value specified in Integration Server.
- **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), the default 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.

  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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

Suspend Interval
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.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Attach WSDL Dialog Box (SOAP Virtual Service)

Use this option to attach a WSDL to an existing virtual service. The new attached WSDL will override the old WSDL. In the dialog box that appears, specify one of the following options and click **OK**:

- **URL**: Specify the URL of the WSDL to attach.
- **File**: Click **Browse** to select a WSDL from your local file system.

Note: If the new WSDL contains referenced XSDs or WSDL, they will be copied if they are resolved by CloudStreams. But if the referenced XSDs or WSDL cannot be resolved, a dialog box will prompt you whether to import the unresolved XSDs or WSDL.

Transform Step (Outbound, SOAP Virtual Service)

Add this step if you need to invoke an XSLT transformation file to transform the native service's response messages into the format required by the consumer applications, before CloudStreams returns the responses to the consumer applications. For more information about when to use the Transform step, see "Transform Step (SOAP Virtual Service)" on page 57.

To add the Transform step (outbound):

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **Out Sequence** step in the editor.
3. Right-click **Out Sequence** and click **Transform**.

   The "Transform" step is added under the "Out Sequence" step.
4. Click **Transform** and complete the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming convention restrictions.

XSLT Service
The XSLT file to transform the response message before it is returned to the consuming application. Click **Browse** to select a file from your file system and click **Save**.

The XSL file uploaded by the user should not contain the XML declaration in it (`xml version="1.0" encoding="UTF-8"`). This is because when the virtual service is deployed to CloudStreams, CloudStreams 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.

Note: If you make changes to the XSLT file in the future, you must re-deploy the virtual service.

Invoke IS Service Step (Outbound, SOAP Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "Out Sequence" step to pre-process the response message from the native service before it is returned to the consuming application. You can create multiple "Invoke IS Service" steps.

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

You can use the following constructs in an IS service:

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

To add the Invoke IS Service step (outbound):

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **Out Sequence** step in the editor.
3. Right-click **Out Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the "Out Sequence" step.
4. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The Integration Server service to pre-process the request message before it is submitted to the native service. Click **Browse** to select a file from your file system.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `MessageContext`: CloudStreams will automatically place a `MessageContext` variable into the pipeline before executing the 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 (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 Serverpub.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*.

Error Messaging Step (SOAP Virtual Service)

CloudStreams returns a default fault response to the consuming application, which you can customize with context variables. This fault response is used for faults returned by the native service provider as well as faults returned by internal CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors). In addition, you can:

- Choose whether or not to send the native service provider's service fault content, or just send the response message.
- Invoke IS services to pre-process or post-process the error messages.

You use this step to configure error messaging for a single virtual service. If you want to configure global error messaging for *all* virtual services, set the Service Fault Configuration options, which are located in the Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Service Fault Configuration**).

To configure the Error Messaging step:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **Error Sequence** step in the editor.
3. Click **Error Messaging** and complete the following fields in the Properties view.

Error Message
Select one or both of the following options:

- **Custom Failure Response Message**: Returns the following fault response to the consuming application:

  ```
  CloudStreams 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. CloudStreams 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 CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors).

  In this case, the `$ERROR_MESSAGE` variable will contain errors generated by CloudStreams.

The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.), which are described in the document *Administering webMethods CloudStreams*.

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

- The predefined CloudStreams context variables listed in the section *The Predefined Context Variables* in the document *Administering webMethods CloudStreams*.
- Custom context variables that you declare using the CloudStreams API (see the section *The API for Context Variables* in the document *Administering webMethods CloudStreams*).

- **Native Provider Fault**: When you select this option, CloudStreams sends the native service provider's service fault content, if one is available. CloudStreams will send whatever content it received from the native service provider.

  If you select this option, the **Custom Failure Response Message** is ignored when a fault is returned by the native service provider. (Faults returned by internal CloudStreams exceptions will still be handled by the **Custom Failure Response Message** option.)

Processing Method

Optionally select either of the following:

- **Pre-Processing**: Select this option if you want to invoke an IS service to manipulate the response message before the Error Sequence 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 about IS services, see "Invoke IS Service Step (Inbound, SOAP Virtual Service)" on page 58.
- **Post-Processing**: Select this option if you want to invoke an IS service to manipulate the service fault after the Error Sequence 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.

Virtual Services (REST)

Use this editor to create and modify REST-based virtual services. *Virtual services* handle requests in the inbound processing scenario. When a SaaS application sends a service request, a virtual service performs security checks and other user-defined processing before sending the request to the on-premise application.

A virtual service runs on CloudStreams and acts as the consumer-facing proxy for a native service running in an on-premise application. A virtual service provides a layer of abstraction between the service consumer and the service provider, and promotes loose coupling by providing independence (of location, protocol and format) between the consuming application and the provider service. You should create a virtual service for each native service you want to expose to consumers.

When you create a virtual service, you can configure the following processing steps for the service:

- The **In Sequence step**, which you configure to manipulate the request messages. This step can include the following sub-steps:
  - The **Entry step** (provided by default), which specifies the protocol (HTTP or HTTPS) of the requests that the virtual service will accept. You also specify the REST resource that the service will access and the HTTP methods (GET, POST, PUT and DELETE) that the virtual service should be allowed to perform on the REST resource.
  - The **Transform step** (optional), which performs an XSLT message transformation on the request message before the virtual service submits it to the native service.
  - The **Invoke IS Service step** (optional), which pre-processes the request message before the virtual service submits it to the native service.
  - The **Routing Rule step** (provided by default), which specifies how the virtual service will route the requests to the native service endpoint. There are four ways to route HTTP or HTTPS requests:
    - "Straight Through" routing (to route requests directly to the native service endpoint).
    - "Context-Based" routing (to route specific types of messages to specific endpoints according to context-based routing rules).
    - "Content-Based" routing (to route specific types of messages to specific endpoints based on specific values that appear in the request message).
    - "Load Balancing" routing (to distribute requests among multiple endpoints).
- The service's **Out Sequence step**, which you configure to manipulate the response messages. This step can include the following sub-steps:

- The **Transform step** (optional), which specifies how the response message from the native service provider is to be transformed before the virtual service returns it to the consuming application.
- The **Invoke IS Service step** step (optional), which pre-processes the response message before the virtual service returns it to the consuming application.

- The service's **Error Sequence step** (provided by default). CloudStreams returns a default fault response to the consuming application, which you can customize with context variables. This fault response is used for faults returned by the native service provider as well as faults returned by internal CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors). In addition, you can:
  - Choose whether or not to send the native service provider's service fault content, or just send the response message.
  - Invoke IS services to pre-process or post-process the error messages.

New Virtual Service Wizard (REST)

To open the New Virtual Service wizard:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, right-click the CloudStreams Governance project and click **New > Virtual Service** (or expand the project, right-click the **Virtual Services** folder and click **New Virtual Service**).

Complete the following fields in the New Virtual Service wizard:

CloudStreams Governance Project
The CloudStreams Governance project in which you are creating the virtual service.

Name
Assign a name for the service. Unlike native services, the names of virtual services cannot contain spaces or special characters except _ (underscore) and - (hyphen). Consequently, if you adopt a convention that involves using the name of the service as part of the virtual service name, then the names of the services themselves must not contain characters that are invalid in virtual service names.

> **Note:** If you want to change the service name after it has been created, right-click the service name in the **Virtual Services** folder and select **Rename**.

Version
The version is always set to 1.0.

Type
Select **REST**.

Description
Optional. A description for the service. This description appears when a user displays a list of services in the user interface.

REST Resources Wizard

To open the REST Resources wizard:

1. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**). Click the virtual service name and then click the "Advanced" tab.
2. On the **Advanced** page in the Properties view, click the **Resource** button.

Complete the following fields:

Resource
Specify the name of the REST resource that the service will access.

> **Note:** To add a REST resource to the service after the service has been created, right-click the service name and select **Attach Resource** from the context menu.

HTTP Method
The HTTP methods that the virtual service should be allowed to perform on the REST resource (GET, POST, PUT and DELETE).

It is important to specify all the HTTP methods that are supported for the virtual service. For example, if the virtual service is deployed to CloudStreams and only the GET method was selected in the virtual service definition, then CloudStreams will only permit GET invocations. A POST request will be rejected with a return of statusCode 405 even if the native service happens to support POSTs. It is also important for the native service to return the correct "Content-Type" header with its responses.

At run time, CloudStreams determines the type of a request message based on the message's HTTP method and its Content-Type. For a list of valid HTTP method/ Content-Type combinations, see the section *The Request Message's HTTP Methods and Content-Types for REST Services* in the document *Administering webMethods CloudStreams*.

Content Type
Select **Application/xml** or **Application/json**.

Query String
Specify the search string as required. Any text typed in is not case sensitive. This field is used only for documentation purposes at this time. It is not included in the service's "virtual service definition" (VSD) that is deployed to CloudStreams, so this value has no impact at runtime.

Import From
Import the schemas that the virtual service will use. Choose one of the following options. To select multiple schemas, click the plus button to add a row.

- **URL**: Specify the URL of a schema and click the **Add** button. Optionally choose the following option::
  - **URL authentication**: If you have specified a URL and the site you want to access using the URL requires user authentication, select this option. This opens an Authentication sub-dialog in which you can enter a user name and password for authentication at the URL site.
- **File**: Click **Browse** to select a schema from your local file system.

General Properties (REST Virtual Service)

To display the general properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**). Select the virtual service name and then select the "General" tab.

The **General** page in the Properties view displays the following properties:

Name
(Read-only field.) The service name. You can change the service name at any time by right-clicking the name in the **Virtual Services** folder and clicking **Rename**.

Service Type
(Read-only field.) **REST**.

Created/Last Modified
(Read-only field.) The service's creation/modification timestamps.

Target Namespace
(Read-only field.) The value is derived from the `targetNamespace` attributes of the WSDL's definition element.

Namespace
Specify other namespaces (such as `wsdl`, `xsd`, and so on). See "Virtual Service Namespaces Dialog Box (REST Virtual Service)" on page 78.

Version
The version is always set to 1.0.

WSDL URL
(Read-only field.) The URL of the REST resource's abstract WSDL.

Description
You can change the service description.

Advanced Properties View (REST Virtual Service)

To display the advanced properties:

1. Open Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, expand your CloudStreams Governance project and click the virtual service name. (If the Properties view is not already open, click **Window > Show View > Other > General > Properties**). Select the virtual service name and then select the "Advanced" tab.

Name
(Read-only field.) The service name. You can change the name of an undeployed service by right-clicking the name in the **Virtual Services** folder and clicking **Rename**.

Type
(Read-only field.) The type of the service (REST).

Target Namespace
(Read-only field.) The value is derived from the `targetNamespace` attributes of the WSDL's definition element.

WSDL URL
(Read-only field.) The URL of the REST resource's abstract WSDL.

Namespaces
Specify other namespaces (such as `wsdl`, `xsd`, and so on). See "Virtual Service Namespaces Dialog Box (REST Virtual Service)" on page 78.

Version
The version is always set to 1.0.

Description
(Read-only field.) The service description.

VSD
Click the button next to this field to view the virtual service definition (VSD). For details, see "VSD Dialog Box (REST Virtual Service)" on page 78.

Applicable Policies
Click the button next to this field to view a list of all active policies that apply to this service. Any inactive policy that applies to the service is not listed. For details, see "Applicable Policies Dialog Box (REST Virtual Service)" on page 78.

Endpoint
Click the button next to this field to view the endpoint to which the virtual service is deployed, if applicable (see "Endpoint Dialog Box (REST Virtual Service)" on page 79).

Deployed Status
(Read-only field.) Indicates whether the service is **Deployed**, **Undeployed** or **Not Deployed** (which is the initial status before you deploy the service).

Resource
Click the button next to this field to view the REST resource that the service will access. To add a REST resource to the service, right-click the service name in the CloudStreams Governance view and click **Attach Resource**.

Virtual Service Namespaces Dialog Box (REST Virtual Service)

This dialog box displays the virtual service's available namespaces. The namespace prefixes (such as `wsdl`, `tns`, `xsd`, etc.) and names are displayed.

VSD Dialog Box (REST Virtual Service)

This dialog box displays the virtual service definition (VSD) that CloudStreams generates when you deploy a virtual service to a CloudStreams server target. You cannot edit the VSD.

When you deploy a virtual service to a CloudStreams server target, CloudStreams generates an XML document called a *virtual service definition (VSD)*. The VSD defines the virtual service for CloudStreams, and contains all the resources required to deploy the virtual service to CloudStreams, including the policy that applies to the service.

Note: If multiple policies apply to the service, CloudStreams combines all those policies into a single policy known as the *effective policy*. The effective policy is a simple UNION of the run-time actions specified in all policies that apply to a service. To create the effective policy, CloudStreams evaluates the combined list of actions from all policies, using a set of internal rules known as Policy Conflict Resolution rules. For details, see the topic *Policy Conflict Resolution Rules* in the document *Administering webMethods CloudStreams* available on the Software AG Documentation website.

Applicable Policies Dialog Box (REST Virtual Service)

This dialog box lists all active policies that apply to this service. Any inactive policy that applies to the service is not listed.

Note: If multiple policies apply to the service, CloudStreams combines all those policies into a single policy known as the *effective policy*. The effective policy is a simple UNION of the run-time actions specified in all policies that apply to a service. To create the effective policy, CloudStreams evaluates the combined list of actions from all policies, using a set of internal rules known as Policy Conflict Resolution rules. For details, see the topic *Policy Conflict Resolution Rules* in the document *Administering webMethods CloudStreams*.

Endpoint Dialog Box (REST Virtual Service)

This dialog box displays the virtual service's CloudStreams endpoint (not the native service endpoint).

Entry Step (REST Virtual Service)

To configure the Entry Step:

1. Click the Virtual Service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Entry Step** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Entry Step** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Entry Step**.

Protocol
The protocol (HTTP or HTTPS) over which the virtual service will accept requests. To specify HTTPS, select both **HTTP** and **SSL**.

The Entry 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.

HTTP Methods
The HTTP methods that are supported by the native service (GET, POST, PUT and DELETE).

It is important to specify all the HTTP methods that are supported for the virtual service. For example, if the virtual service is deployed to CloudStreams and only the GET method was selected in the virtual service definition, then CloudStreams will only permit GET invocations. A POST request will be rejected with a return of statusCode 405 even if the native service happens to support POSTs. It is also important for the native service to return the correct Content-Type header with its responses.

At run time, CloudStreams determines the type of a request message based on the message's HTTP method and its Content-Type. For a list of valid HTTP method/ Content-Type combinations, see the section *The Request Message's HTTP Methods and Content-Types for REST Services* in the document *Administering webMethods CloudStreams*.

Transform Step (REST Virtual Service)

Add the optional Transform step if you need to transform the request message into the format required by the native service, before CloudStreams submits the message to the native service.

As long as a consumer sends a REST request with the proper Content-Type, and as long as the HTTP method and endpoint URI are in the expected form, then CloudStreams can detect the correct service and operation; in this case, no message transformation is required. However, in some cases a virtual REST 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 Transform step to transform the record submitted by the consuming application to the structure required by the Web service.

In this case, you would need to create two Transform steps:

- One in the "In Sequence" step, to transform or pre-process the request messages into the format required by the native service, before CloudStreams sends the requests to the native services. To do this, you pass the message to an XSLT transformation file. (Additionally in this case, the transformation is required if the virtual service has a schema validation policy that validates the requests.)
- One in the "Out Sequence" step, to transform or pre-process the native service's response messages into the format required by the consumer applications, before CloudStreams returns the responses to the consumer applications.

Transform Step (Inbound, REST Virtual Service)

To add the Transform step (inbound):

1. Click the virtual service name in the CloudStreams Governance view.
2. Right-click **In Sequence** and click **Transform**.

   The "Transform" step is added under the Entry Step. You cannot change the order of the steps.
3. Click **Transform** and complete the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming convention restrictions.

XSLT Service
The XSLT file to transform the request message before it is submitted to the native service. Click **Browse** to select a file from your file system and click **Save**.

The XSL file uploaded by the user should not contain the XML declaration in it (`xml version="1.0" encoding="UTF-8"`). This is because when the virtual service is deployed to CloudStreams, CloudStreams 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.

Note: If you make changes to the XSLT file in the future, you must re-deploy the virtual service.

Invoke IS Service Step (Inbound, REST Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "In Sequence" step to pre-process the request message before it is submitted to the native service. You can create multiple "Invoke IS Service" steps.

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

You can use predefined or custom context variables in an IS service. For more information, see *Using Context Variables* in the document *Administering webMethods CloudStreams*.

To add the Invoke IS Service step:

1. Click the virtual service name in the CloudStreams Governance view.
2. Right-click **In Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the Entry Step. You cannot change the order of the steps.
3. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The IS Service to pre-process the request message before it is submitted to the native service.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `MessageContext`: CloudStreams will automatically place a MessageContext variable into the pipeline before executing the 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 (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 Serverpub.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*.

Routing Rule Step (Straight Through Routing, REST Virtual Service)

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).

To configure the Routing Rule step for Straight Through routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **Straight Through**.

Default To
The URL of the service to which to route the request.

Configure Endpoint Properties icon
This icon displays a dialog box that enables you 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 Properties**
  - **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), CloudStreams will use the default value specified in Integration Server.

- **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), the default 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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

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 **CUSTOM**. 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 **CUSTOM** option and then write an IS service to set the value of the context variable. For more information, see the section *Changing the HTTP Method of a REST Virtual Service* in the document *Administering webMethods CloudStreams*.

Use credentials from incoming request

Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials

Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously

Does not authenticate requests.

Use existing HTTP headers

Use the HTTP headers that are contained in the requests.

Customize HTTP headers

Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Routing Rule Step (Context-Based Routing, REST Virtual Service)

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

A routing rule specifies where the 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. For example, you might use this capability to route requests from certain high-priority consumers to endpoints on a fast machine.

To configure the Routing Rule step for Context-Based routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **Context-Based**.

Rule Name
Assign a name to the rule.

Then, click the icon next to this field and complete the Configure Routing Rule dialog box as follows:

- Choose one of the following: **Time**, **IP Address** (IPv4 or IPv6 format), **Date**, **Consumer**, **Predefined Context Variable** or **Custom Context Variable** (see the section *Using Context Variables* in the document *Administering webMethods CloudStreams*).

  Note: If you select the value **Custom Context Variable**, you must write an IS service to get/set the custom context variable, and then specify the service in the **IS Service Name** field on the **Routing Rules** page. CloudStreams provides an API to get/set custom context variables. For more information, see the section *The API for Context Variables* in the document *Administering webMethods CloudStreams*. CloudStreams automatically declares any custom context variables you have specified; there is no need for you to declare them.

Route To
Enter the URL of the native service to route the request to, if the rule criteria are met.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows:

- **HTTP Properties**
  - **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), CloudStreams will use the default value specified in Integration Server.
  - **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), the default 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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

HTTP Method
The HTTP method to pass to the rule.

Default To
A native service endpoint to route the request to in case all routing rules evaluate to False.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box, as described for the **Route To** field above.

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 **CUSTOM**. **CUSTOM** is a context variable that 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 **CUSTOM** option and then write an IS service to set the value of

the context variable. For more information, see the section *Changing the HTTP Method of a REST Virtual Service* in the document *Administering webMethods CloudStreams*.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Routing Rule Step (Content-Based Routing, REST Virtual Service)

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

To configure the Routing Rule step for Content-Based routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.
3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Protocol
(Read-only field.) **HTTP**.

Routing Type
Select **Content-Based**.

Rule Name
Assign a name to the rule.

XPath
The XPath field is applicable only while creating SOAP Virtual Services.

Route To
Specify where to route the request if the rule criteria are met. Specify either the URL of the native service or a connection pool name.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows:

- **HTTP Properties**
  - **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), CloudStreams will use the default value specified in Integration Server.
  - **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), the default 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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

HTTP Method
The HTTP method to pass to the rule.

Default To
A native service endpoint to route the request to in case all routing rules evaluate to False.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box, as described for the **Route To** field above.

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 **CUSTOM**. **CUSTOM** is a context variable that 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 **CUSTOM** option and then write an IS service to set the value of the context variable. For more information, see the section *Changing the HTTP Method of a REST Virtual Service* in the document *Administering webMethods CloudStreams*.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Routing Rule Step (Load Balancing Routing, REST Virtual Service)

If you have a native service that is hosted at two or more endpoints, you can use the Load Balancing protocol to distribute requests among the 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 Interval** 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 the Routing Rule page for Load Balancing routing:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **In Sequence** step in the editor.

3. Click **Routing Rule** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Routing Rule** to any other name. There are no naming convention restrictions.

Type
(Read-only field.) **Routing Rule**.

Format
(Read-only field.) **HTTP**.

Routing Type
Select **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 **Endpoint** and select the URL of the Web service to route the request to.

To specify additional services, use the plus button next to the field to add rows.

Then, click the icon next to this field and complete the Configure Endpoint Properties dialog box as follows. These properties will apply to all the endpoints.

- **HTTP Properties**
  - **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), CloudStreams will use the default value specified in Integration Server.
  - **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), the default 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.
  - **IS Keystore Alias**: The keystore alias of the instance of Integration Server on which CloudStreams is running. This value (along with the value of **Client Certificate Alias**) will be used for performing SSL client authentication.

Suspend Interval
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 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 **CUSTOM**. **CUSTOM** is a context variable that 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 **CUSTOM** option and then write an IS service to set the value of the context variable. For more information, see the section *Changing the HTTP Method of a REST Virtual Service* in the document *Administering webMethods CloudStreams*.

Use credentials from incoming request
Default. Authenticates requests based on the credentials specified in the HTTP header. CloudStreams passes the "Authorization" header present in the original client request to the native service.

Use specific credentials
Authenticates requests according to the values you specify in the **User**, **Password** and **Domain** fields.

Invoke service anonymously
Does not authenticate requests.

Use existing HTTP headers
Use the HTTP headers that are contained in the requests.

Customize HTTP headers
Use the HTTP headers that you specify in the **Name** and **Value** columns on the tab. If you need to specify multiple headers, use the plus button to add rows.

Transform Step (Outbound, REST Virtual Service)

Add this step if you need to invoke an XSLT transformation file to transform the native service's response messages into the format required by the consumer applications, before CloudStreams returns the responses to the consumer applications. For more information about when to use the Transform step, see "Transform Step (REST Virtual Service)" on page 80.

To add the Transform step (outbound):

1. Click the virtual service name in the CloudStreams Governance view.
2. Right-click **Out Sequence** and click **Transform**.

   The "Transform" step is added under the "Out Sequence" step. You cannot change the order of the steps.
3. Click **Transform** and complete the following fields in the Properties view.

Name
You can optionally change the step name to any other name. There are no naming convention restrictions.

XSLT Service
The XSLT file to transform the response message before it is returned to the consuming application. Click **Browse** to select a file from your file system and click **Save**.

The XSL file uploaded by the user should not contain the XML declaration in it (`xml version="1.0" encoding="UTF-8"`). This is because when the virtual service is deployed to CloudStreams, CloudStreams 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.

Note: If you make changes to the XSLT file in the future, you must re-deploy the virtual service.

Invoke IS Service Step (Outbound, REST Virtual Service)

An IS (Integration Server) service is a user-defined Integration Server flow service that you can invoke in the "Out Sequence" step to pre-process the response message from the native service before it is returned to the consuming application. You can create multiple "Invoke IS Service" steps.

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

You can use predefined or custom context variables in an IS service. For more information, see the section *Using Context Variables* in the document *Administering webMethods CloudStreams*.

To add the Invoke IS Service step (outbound):

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **Out Sequence** step in the editor.
3. Right-click **Out Sequence** and click **Invoke IS Service**.

   The "Invoke IS Service" step is added under the "Out Sequence" step. You cannot change the order of the steps.

4. Click **Invoke IS Service** and complete the following fields in the Properties view.

Name
You can optionally change the step name from **Invoke IS Service** to any other name. There are no naming convention restrictions.

Service
The IS Service to pre-process the response message before it is returned to the consuming application. Click **Browse** to select a file from your file system.

When you define the IS service, the **Pipeline In** section of the service should have the following input variables:

- `proxy.name`: This is the name of the virtual service.
- `SOAPEnvelope`: This is of the Java type org.apache.axiom.soap.SOAPEnvelope
- `MessageContext`: CloudStreams will automatically place a MessageContext variable into the pipeline before executing the 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 (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 Serverpub.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*.

Error Messaging Step (REST Virtual Service)

CloudStreams returns a default fault response to the consuming application, which you can customize with context variables. This fault response is used for faults returned by the native service provider as well as faults returned by internal CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors). In addition, you can:

- Choose whether or not to send the native service provider's service fault content, or just send the response message.
- Invoke IS services to pre-process or post-process the error messages.

You use this step to configure error messaging for a single virtual service. If you want to configure global error messaging for *all* virtual services, set the Service Fault Configuration options, which are located in the Integration Server Administrator (go to **Solutions > CloudStreams > Administration > Service Fault Configuration**).

To configure the Error Messaging step:

1. Click the virtual service name in the CloudStreams Governance view.
2. Expand the **Error Sequence** step in the editor.
3. Click **Error Messaging** and complete the following fields in the Properties view.

Error Message
Select one or both of the following options:

- **Custom Failure Response Message**: Returns the following fault response to the consuming application:

  ```
  CloudStreams 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. For REST service calls, the message is returned inside an `</Exception>` tag. If the response is XML, the message is returned inside `<Exception>'`*`custom message`*`' </Exception>`. If the response is JSON, it will be returned inside `{"Exception":"Invalid response"}`. CloudStreams 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 CloudStreams exceptions (policy violation errors, cloud connection errors and cloud connector service errors).

    In this case, the `$ERROR_MESSAGE` variable will contain errors generated by CloudStreams.

  The default fault response contains predefined fault handler variables (`$ERROR_MESSAGE`, `$OPERATION`, etc.), which are described in the document *Administering webMethods CloudStreams*.

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

  - The predefined CloudStreams context variables listed in the section *The Predefined Context Variables* in the document *Administering webMethods CloudStreams*.

- Custom context variables that you declare using the CloudStreams API (see the section *The API for Context Variables* in the document *Administering webMethods CloudStreams*).

- **Native Provider Fault**: When you select this option, CloudStreams sends the native service provider's service fault content, if one is available. CloudStreams will send whatever content it received from the native service provider.

  If you select this option, the **Custom Failure Response Message** is ignored when a fault is returned by the native service provider. (Faults returned by internal CloudStreams exceptions will still be handled by the **Custom Failure Response Message** option.)

Processing Method

Optionally select either of the following:

- **Pre-Processing**: Select this option if you want to invoke an IS service to manipulate the response message before the Error Sequence 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 about IS services, see "Invoke IS Service Step (Inbound, REST Virtual Service)" on page 81.

- **Post-Processing**: Select this option if you want to invoke one or more IS services to manipulate the service fault after the Error Sequence 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.

Deploy

You can deploy Virtual Services and Connector Virtual Services to CloudStreams server targets in two ways:

- Deploy all Virtual Services, and any custom Connector Virtual Services, that are contained in a particular CloudStreams Governance project.

  OR

- Deploy a single Virtual Service or custom Connector Virtual Service. Generally, this is useful for testing purposes.

Note: CloudStreams does not support sharing of Connector Virtual Services, Virtual Services, and Policies across nodes in a clustered setup. These artifacts should be manually deployed to a clustered node as needed.

Note: CloudStreams automatically deploys the default Connector Virtual Services WmCloudStreams.SoapVS and WmCloudStreams.RestVS; there is no need for you to deploy them.

When you execute the deployment operation, CloudStreams will immediately deploy the service(s) and the following items to the CloudStreams server target(s) that you specify:

- The policies of each service, as well as any other artifacts that you associated with the services when you created them.
- The VSD (virtual service definition) of each virtual service or connector virtual service.

  When you deploy a Virtual Service or a Connector Virtual Service, CloudStreams generates an XML document called a "virtual service definition (VSD)". The VSD defines the virtual service or connector virtual service for CloudStreams, and contains all the resources required to deploy the service to a CloudStreams server, including the policy that applies to the service. You cannot edit the VSD, but you can view it in the Advanced page in the Properties view of each service.

Before you deploy a virtual service, you should:

- Ensure that all policies of the services are Active; the value of the **Status** field in the Properties view of each policy should be **Active**. If it is not, right-click the policy name in the CloudStreams Governance view and click **Active**. You will not be allowed to activate a policy unless all of its action parameters have been set.
- Ensure that at least one CloudStreams server target has already been defined, as described in "Add CloudStreams Servers Dialog Box" on page 12.
- Ensure that the server's specified deployment URL is active and the user credentials of Integration Server are valid.
- Test the server connection from the Designer menu by clicking **Window > Preferences > Software AG > CloudStreams Servers** and using the **Test** button. If the connection is not active and valid, activate the deployment endpoint and modify the user credentials as required.

If CloudStreams encounters a problem deploying or redeploying a service, it displays a failure message, which is logged to the deployment log at the bottom of the page. In this case, it is up to the CloudStreams administrator to take corrective action and redeploy the service. If the reason for the failure is that the CloudStreams server is unavailable, and then you restart the CloudStreams server, it loads all information about any previously deployed services. For more information, see the topic *What Happens When You Deploy a Service?* in the document *Administering webMethods CloudStreams*.

To deploy all services in a CloudStreams Governance project:

When you deploy a CloudStreams Governance project, all virtual services in the project are deployed at once. If any custom connector virtual services are defined in the project, they are deployed too.

When you deploy a project, Integration Server automatically creates a package to support the project. The package name is the same as your project name. This package includes startup/shutdown services to manage the registration of the virtual service definitions (VSDs) when Integration Server restarts.

1. Open Software AG Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, right-click the project you want to deploy and click **Deploy**.
3. In the Deploy dialog box, choose one or more CloudStreams server targets to which to deploy the services and click **OK**. The services are immediately deployed.

   If only one CloudStreams server target is available to select, this dialog box will not appear.

   The services are immediately deployed to the following location:

   IntegrationServer_directory \packages*<PackageName><ProjectName* \config\proxies *VirtualService*

To deploy a single virtual service or connector virtual service:

1. Open Software AG Designer and display the CloudStreams Development perspective by clicking **Window > Open Perspective > Other > CloudStreams Development**.
2. In the CloudStreams Governance view, right-click the service name you want to deploy and click **Deploy**.
3. In the Deploy dialog box, choose one or more CloudStreams server targets to which to deploy the service and click **OK**. The service is immediately deployed.

   If only one CloudStreams server target is available to select, this dialog box will not appear.

   The service is immediately deployed to the following location:

   IntegrationServer_directory \packages*<PackageName><ProjectName* \config\proxies *VirtualService*

2 CloudStreams Provider Project

You must create a CloudStreams Provider project in which you will define the connector. You can perform several management tasks for Provider projects, such as publish, unpublish, import, export, delete, and more.

Creating a Provider Project

You create a CloudStreams Provider project in which you will define the connector. You can perform several management tasks for Provider projects, such as export, import, publish, unpublish, and delete.

To create a Provider Project

1. In **Software AG Designer**, click **Window > Perspective > Open Perspective > Other > CloudStreams Development** to open the CloudStreams Development perspective.
2. Click **File > New > CloudStreams Provider Project** from the main menu.
3. In the **Create New CloudStreams Provider Project** wizard, assign a name to the new project in the **Project name** field using any combination of letters, numbers, and the underscore character. The project name must not be null and cannot be an empty string.

   Note: CloudStreams connector development is a connected development where all artifacts are created on Integration Server.

   CloudStreams Provider project names are validated according to the same rules that apply to Integration Server package names. Keep the following guidelines in mind when naming new packages:

   - Start all package names with an uppercase letter and capitalize the first letter of subsequent words (for example, PurchaseOrder).
   - Keep package names short. Use abbreviations instead of full names. For example, instead of ProcessPurchaseOrder, use ProcessPO.
   - Ensure that the package name describes the functionality and purpose of the services it contains.
   - Control characters and special characters like periods (.), including:
   - Avoid creating package names with random capitalization (for example, cOOLPkgTest).
   - Avoid using articles (for example, “a,” “an,” and “the”) in the package name. For example, instead of TestTheService, use TestService.
   - Avoid using the prefix “Wm”. Integration Server and Software AG Designer use the “Wm” prefix for predefined packages that contain services, IS document types, and other files.
   - Avoid using control characters and special characters like periods (.) in a package name. The watt.server.illegalNSChars setting in the server.cnf file (which is

located in the *IntegrationServer_directory* \instances*instance_name* \config directory) defines all the characters that you cannot use when naming packages. Additionally, the operating system on which you run the Integration Server might have specific requirements that limit package names.

4. The CloudStreams Target Runtime displays the default API level supported by the server. The project will be built on the default API level.
5. Select the type of connector in the **Connector Type** field. The type of connector depends on the type of SaaS provider with which you want to communicate. If you communicate with a REST-based provider, you create connections using a **REST** connector and if you communicate with a SOAP-based provider, you create connections using a **SOAP** connector.
6. Click **Next** to add Connector information in the **Connector Details** screen or click **Finish** to create the provider. The provider package is created at <your_integration_server_installaiton_location>\instances\default\packages.

CloudStreams Connector

A CloudStreams connector contains connections and services that you use to integrate with a software as a service (SaaS) provider.

Although CloudStreams provides predefined connectors for some SaaS applications (for example, Salesforce.com), you might want to create connectors for other SaaS applications. You use the CloudStreams Development plug-in to create connectors, which you can publish to webMethods Integration Server. The CloudStreams connector is installed on webMethods Integration Server as a package.

The type of a CloudStreams connector depends on the type of SaaS provider you communicate with. If you communicate with a SOAP-based cloud application provider, you create cloud connections using a CloudStreams SOAP connector. If you communicate with a REST-based provider, you create cloud connections using a CloudStreams REST connector. You use Integration Server Administrator to load, manage, and use the CloudStreams connectors.

You can create one or more connections for a CloudStreams connector at design time to use in integrations. A connector service uses a connection to connect to a SaaS provider at run time. You must create and enable a connection before you can create connector services. The number of connections you create depends on your integration needs.

A connector service is an webMethods Integration Server service used to integrate an on-premise application with a SaaS application. At run time, the connector service constructs and maps a SOAP or REST request from the service's input pipeline to an appropriate message builder. When the application provider sends a response, the connector service processes and maps the response and populates the output pipeline. If you are using a REST-based provider, you must create at least one connector service for each REST resource.

Creating a Connector

To create a new connector, right-click the Provider project and click **New Rest Connector** to create a REST connector or **New SOAP Connector** to create a SOAP connector. Complete the following fields in the **Connector Details** screen:

Name
Enter a valid name for the connector. Only alphanumeric characters and "_" are valid for the connector name.

ID
The ID will uniquely identify the connector in CloudStreams. Only alphanumeric characters, "_" and "." are valid for the connector ID. Further, the connector ID should not have leading integers. For example, com.softwareag.7 is an invalid ID, but com.softwareag7 is a valid ID.

Group
The **Group** field is optional. It is the group under which you would like to categorize this connector. If you specify a group, the connector will be added to that group, else the connector will be added to the group with the same name as the provider. To specify a group for the connector, type a group name in this field or select an existing group name from the drop-down list.

> **Note:** The group will also be visible in the Integration Server Administrator's Connector List (under **Solutions > CloudStreams > Providers**) if at least one connector is published.

File/URL (SOAP only)
This field appears only while creating a SOAP connector. Specify the SaaS provider's WSDL that CloudStreams will use to generate the document types by selecting either:

- **File**. Specify the WSDL file name, or click **Browse** and select the WSDL from your local file system.
- **URL**. Specify the URL of the WSDL.

Import Swagger specification file (REST only)
Select this option if you want to import a Swagger document. Swagger is an open specification for defining REST APIs. A Swagger document is the REST API equivalent of a WSDL document for a SOAP-based web service and specifies the list of resources that are available in the REST API. The Swagger document also specifies the list of parameters for the resource, including the name and type of the parameters, whether the parameters are required or optional, and information about acceptable values for those parameters. Additionally, the Swagger document can include JSON Schema that describes the structure of the request and response bodies. Swagger documents must be in either JSON format with a .json file extension, or YAML format with a .yaml or .yml file extension.

Click **Browse**, select the Swagger file, and then click **Finish**. The Provider Project, the Connector, and all Resources and dependent assets will be created.

Endpoint (REST only)
This field appears only while creating a REST connector. Type the login endpoint URL to initiate communication with the SaaS provider. For example, for Twitter, the end point is https://api.twitter.com/1. To get the end point, go through the SaaS connector documentation available on the internet. For example, for Twitter, go to https://dev.twitter.com/rest/reference.

Authentication (REST only)
This field appears only while creating a REST connector. Every back end provides its own authentication mechanism. Get the authentication details from the back end documentation. Select that authentication in the drop down. For example, for the Twitter connector, the authentication is OAuth V1.0a, which you can get from the Twitter documentation available at https://dev.twitter.com/.

Note: For information on end point URL and supported authentication schemes, see the relevant SaaS provider documentation.

Click **Next** to select the connection configuration groups or click **Finish** to create the Connector. This step is optional. You can choose to select one or more of the available groups.

Configuration groups allow you to define configurations needed for the HTTP protocol and also define the HTTP request header contents. The **Connection** configuration group is selected by default and allows you to define the required connection configurations to initiate communication with the SaaS provider.

The **Connection Configuration** screen allows you to select the configuration groups. Configuration groups allow you to define configurations needed for the HTTP **Protocol** that the connection will use to connect to the SaaS provider and also define the HTTP **Request Header** contents that will be used while making a request to a SaaS provider. The **Connection** configuration group is selected by default.

Click **Finish**. The new connector is added to your Provider project.

Note: If you are creating a SOAP connector, the WSDL is parsed and all artifacts such as operations and document types are created in the provider package.

Double-click the Provider Project to view the **Overview - Project Information** screen, which describes the project information, for example, the project name, the target runtime, project version, and the publisher of the project. The **Target Runtime** displays the default API level supported by the server. The project is built on the default API level. The **Connectors** section provides information on all the connectors created. The screen also displays the **Project Dependencies** and the **Start/Shutdown Services** information.

Connector Overview-General Information

The **Overview-General Information** screen displays the general information you defined in the Connector wizard, for example, the **ID**, **Name**, **Version**, **Group**, **Vendor**, and **Description**. You can modify relevant fields in this screen.

Click the **Connection** tab at the bottom of the screen to view the **Connection** screen and define the required configurations to initiate communication with the SaaS provider.

Click the **Connector XML** tab for an XML view of the connector, resource, parameter, header, and other details.

Click the **Manifest** tab to view the fields, which contain attributes whose values are captured from your connector definition.

Editing Connection Configuration and Authentication Groups

In this screen, you can define the required configurations to initiate communication with the SaaS provider. In case you want to change the default values for the properties, you can edit them using the respective screens. You can override the connection property default value and can edit the name of the connection property on the **Connection** screen.

On the **Authentication** screen, you can select different authentication types, override the property default value and can also edit the name of the property. You can also enter default values or assign values from Headers, Connection Parameters, Services, and Parameters to the connection and authentication properties.

Note: If you change the **Authentication Type**, all modifications and related references done to the previous authentication type will be lost.

If a provider requires a Login/Logout sequence, click the  icon to view the **Configuration Group** dialog box. In the **Configuration Group** dialog box, select the **Include Login/Logout sequence** option. The **Login/Logout Sequence** tab will appear in the **Connection** configuration screen.

Note: Currently, Login/Logout sequence is supported only for REST connectors.

The connection properties are described below:

Name	The available fields are...
Connection	■ **Server URL**: The native provider endpoint target for the connection configuration. The default configuration field provided with the connection factory is `cn.providerURL`.

Name	The available fields are...
	■ **Min Pool Connections**: The minimum number of socket connections to reserve for a connection configuration alias. ■ **Max Pool Connections**: The maximum number of socket connections to reserve for a connection configuration alias. ■ **Connection TimeOut**: The number of milliseconds a connection attempt will wait before giving up. (0 will wait indefinitely). ■ **Socket Read Timeout**: The number of milliseconds in which the the client must read a response message from the server. (0 will wait indefinitely). ■ **Use Stale Checking**: If true, the connection factory performs additional processing to test the socket to see if it is still functional each time it is used. ■ **Connection Retry Count**: How many times should the connection factory attempt to execute a failed invocation. ■ **Retry On Response Failure**: If true, the retry mechanism will be used for failed responses even if the request was sent successfully. ■ **Use TCP NoDelay**: If true, do not use Nagles algorithm as a socket optimization technique. ■ **Socket Linger**: Determines how quickly a socket should close. ■ **Socket Buffer Size**: The size of the read and write socket buffers, in bytes. ■ **Socket Reuse Address**: If true, the socket will be reused even if it is in TIME_WAIT due to a previous socket closure. ■ **Hostname Verifier**: Fully qualified class name that implements the Apache HC org.apache.http.conn.ssl.X509HostnameVerifier interface. Guards against "man-in-the-middle" attacks. ■ **Proxy Server Alias**: The alias to a web proxy server configuration in Integration Server. ■ **Trust Store Alias**: Alias for the Integration Server trust store configuration. ■ **Session Token**: Session token for a stateful session.

Name	The available fields are...
Protocol	■ **Element Character Set**: The encoding to use for the HTTP request line, headers, etc. ■ **HTTP Content Character Set**: The encoding to use for the request message. ■ **HTTP Protocol Version**: The HTTP version (HTTP/0.9, HTTP/1.0 or HTTP/1.1. The default value for the connection factory is HTTP/1.1. ■ **User Agent**: The value to the connection configuration will send for the User-Agent request header. ■ **Use Expect Continue**: If true, use the Expect/Continue HTTP/1.1 handshake and send the Expect request header. ■ **Wait For Continue Time**: The number of milliseconds that the connection factory's client connection should wait for a "100 Continue" response from the server. ■ **Strict Transfer Encoding**: If true, the connection factory connection raise an exception if the "Transfer-Encoding" header is invalid. ■ **Use Chunking**: If true, use HTTP/1.1 chunking, using a chunk size that matches the socket buffer size. ■ **Follow Server Redirects**: If true, follow server redirects. ■ **Server Redirect Maximum Tries**: Maximum number of times to follow a server redirect.
Request Headers	■ **Request Header Names**: An array of request header names to include for this connection configuration. The value should be a comma-delimited list of header names; for example `Content-Type,SOAPAction`. ■ **Request Header Values**: An array of request header values to include for this connection configuration. The value should be comma-delimited list of values in the same order as the header names; for example, `text/xml,login`.
custom	Click the  icon on the top-right corner of the **Connection** screen to add a custom group. Click **Add** to add a property.

To configure the connection authentication, select the **Authentication** tab:

1. Select the **Authentication Type** from the drop-down list. If you change an Authentication type, all modifications and related references done to the previous

Authentication type will be lost. By default, it shows the authentication you had selected while creating the connector.

2. Select **Show Advanced properties** to view more properties for the selected authentication type.

If the Authentication Type is...	The available properties are...
OAuth V1.0a	■ **Consumer ID**: The 'Consumer Key' issued by the Service Provider and used by the consumer to identify itself to the Service Provider. ■ **Consumer Secret**: A secret used by the Consumer to establish ownership of the 'Consumer Key'. ■ **Access Token**: A value used by the Consumer to gain access to the Protected Resources on behalf of the User, instead of using the User's Service Provider credentials. ■ **Access Token Secret**: A secret used by the Consumer to establish ownership of a given 'Access Token'.
OAuth v2.0	■ **Consumer ID**: A 'client identifier' issued to the client to identify itself to the authorization server. ■ **Consumer Secret**: A secret matching to the 'client identifier'. ■ **Access Token**: A token used by the client to make authenticated requests on behalf of the resource owner. ■ **Instance URL**: Optional field, used to specify a run-time host, if applicable. This may be required in some backends like Salesforce. ■ **Refresh Access Token**: Option to refresh the 'Access Token'. OAuth 2.0 access tokens typically have a very short lifetime. When an access token expires, the OAuth profile does not automatically refresh the expired access token. Select this option if you want an expired access token to be refreshed automatically. If you select this option, you must also specify the relevant refresh parameters. The access token is refreshed whenever the session expires. Session expiration is handled according to the setting of the Session Management property in your connection. Note that if Session Management is set to "none", then you must manually modify the access token in the OAuth alias. (The Refresh Access Token option will not be applicable in this case). Default is 'false'.

If the Authentication Type is...	The available properties are...

- **Refresh Token**: A token used by the client to obtain a new access token without having to involve the resource owner.
- **Refresh URL**: The provider specific URL to refresh an 'Access Token'. This is required when 'Refresh Access Token' is enabled (configured to 'true') and ‘Refresh URL Request’ is configured to 'URL Query String' or 'Body Query String'.
- **Refresh URL Request**: Options for sending the parameters in the 'Access Token' refresh request. The options are 'URL Query String', 'Body Query String', and 'Custom ESB Service'. Default is 'Body Query String'.
- URL Query String: The refresh request parameters, for example, refresh_token, grant_type, and so on, and their values are sent as query strings in the URL of the POST request.

Example:

```
www.examplebackend.com/o/
oauth2/token?grant_type=refresh_token&client_id=
842428530070-pubfebfgfqkgj6t54m4ns6&client_secret=
4adQT95cAtUxWINbDxGP9SJ4&refresh_token=
1%2Fn072P4BXpuNObjCLUtiZTc4fMH6YersmxBIv8QN3bhw
```

- Body Query String: The refresh request parameters, for example, refresh_token, grant_type, and so on, and their values are sent as query strings in the body of the POST request.

Example:

```
POST /o/oauth2/token HTTP/1.1
Host: accounts.backend.com
Content-length: 163
content-type: application/x-www-form-urlencoded
client_secret4adQT95cAtUxWINbDxGP9SJ4&grant_type=
refresh_token&refresh_token=1%2Fn072P4BXpuNObj
CLUtiZTc4fMH6YersmxBIv8QN3bhw&
client_id=407408718192
```

- Custom ESB Service: If the backend requires the refresh request in a custom format, for example, requests which need more parameters than the ones specified by OAuth v2.0, or the backend uses some custom way of organizing parameters, or expects some other HTTP method request (other than POST), use the ‘Custom ESB Service’ option.

Refresh Custom ESB Service: User implemented service for refreshing the 'Access Token'. This is required when the

If the Authentication Type is...	The available properties are...
	'Custom ESB Service' option is selected as the 'Refresh URL Request'. This service must strictly conform to the specification: `- wm.cloudstreams.service.common.lookup.specs:oauthTokenRefreshServiceSpec` **Authorization Header Prefix**: The prefix to be used with the 'Access Token' in the Authorization header. Options are 'Bearer' and 'OAuth'. Default is 'Bearer'.
Credentials	■ **Username**: The username credentials for the current connection configuration. ■ **Password**: The password credentials for the current connection configuration ■ **Authorization Type**: The string identifying the authentication protocol scheme to use for the connection configuration. ■ **Preemptive Auth**: If true, basic auth credentials will be included when a request is sent. (It will not wait for a 401 response challenge.) ■ **Domain Name**: The domain/security realm for the current connection configuration. ■ **Keystore Alias**: Alias for the Integration Server key store configuration. ■ **Client Key Alias**: Alias to reference a key inside a key store file.
AWS V4 Signature	■ **Access Key**: This is a username. It is an alphanumeric text string that uniquely identifies the user who owns the account. No two accounts can have the same AWS Access Key. ■ **Secret Key**: This key plays the role of a password. It is called secret because it is assumed to be known only by the owner. When you type the secret key, it is displayed as asterisk or dots. ■ **Region**: An area-specific value.
AWS V2 Signature	■ **Access Key**: This is a username. It is an alphanumeric text string that uniquely identifies the user who owns the account. No two accounts can have the same AWS Access Key.

If the Authentication Type is...	The available properties are...
	■ **Secret Key**: This key plays the role of a password. It is called secret because it is assumed to be known only by the owner. When you type the secret key, it is displayed as asterisk or dots. ■ **Region**: An area-specific value. ■ **Signing Algorithm**: Explicitly specify the signing algorithm, for example, HMAC-SHA1 Signatures) used to sign the message.
AWS S3 Signature	■ **Access Key**: This is a username. It is an alphanumeric text string that uniquely identifies the user who owns the account. No two accounts can have the same AWS Access Key. ■ **Secret Key**: This key plays the role of a password. It is called secret because it is assumed to be known only by the owner. When you type the secret key, it is displayed as asterisk or dots. ■ **Region**: An area-specific value.

3. If a provider requires a **Login Sequence**, configure one as follows:

   a. Click the  icon on the top-right corner of the **Connection** screen. The **Configuration Group** dialog box appears. In the **Configuration Group** dialog box, select the **Include Login/Logout Sequence** option and click **Apply**. The **Login/Logout Sequence** tab appears.

   b. Configure a **Login Sequence** by completing the following fields:

Field	Description
Method	Select the Method. The login resource will be generated for the selected HTTP method.
Endpoint	Displays the login endpoint URL to initiate communication with the SaaS provider and cannot be edited.
Resource Path	Type the path that you want to use for the resource. Each REST resource derives its path from the namespace of the REST resource.

c. Create a Request for the Login resource by completing the following fields:

Field	Description
Select Request Document	Click **Browse** and select a Document Reference, for example **myConnector_v1.customDoctypes:docTypeRef_LoginInput**.
Content Type	Specify the Content Type of the Document Reference, for example, **application/xml**.

d. Create a Response for the Login resource by completing the following fields. Only one response is currently allowed.

Field	Description
Code	Specify a code for the Response, such as 201 or 400. Response codes can be comma separated or a range.
Select Response Document	Click **Browse** and select a Document Reference, for example **myConnector_v1.customDoctypes:docTypeRef_LoginOutput**.
Content Type	Specify the Content Type of the Document Reference for example, **application/xml**.

e. To assign values to a leaf element, right-click on the leaf element and select **Assign From**. For more information, see "Assigning Override Values" on page 119.

4. Configure the **Logout Sequence** in a similar manner.

> **Note:** While enabling or disabling the connection in the Integration Server Administrator console, the configured login or logout sequences will be run respectively and the values will be assigned, if applicable.

Configuring References

The response that you receive from the SaaS provider has to be incorporated in a Document Type. Document Types (DocTypes) are placeholders for receiving information from the back end. If you have already created a DocType, you can add the DocTypes to the Resource Group namespace.

To add DocTypes to the Resource Group namespace

1. Right-click on the service and select **Configure References**.
2. In the **Configure References - Documents and Services** dialog box > **Documents** tab, click **Add**... to select an already created DocType from a package.
3. You can also add a service created earlier from the **Services** tab, and use the service in the connector.

> **Note:** You need to create DocTypes and Services in the **Service Development** Perspective.

Creating and Renaming Resource Groups

Once you have created a connector, you can create any number of Resource Groups under it. A default Resource Group is also created as part of the initial wizard. The default Resource Group name is derived from the Connector name. For example, if the Connector name is MyConnector, then the display name of the Resource Group will be MyConnector_ResourceGroup and the name is myconnector_resourcegroup.

To create a new resource group

1. In Software AG Designer, click **Window > Perspective > Open Perspective > Other > CloudStreams Development** to open the CloudStreams Development perspective.
2. Right-click on the connector and select **New Resource Group...**.
3. In the **Create New Resource Group** dialog box, provide a name and description of the new resource group.
4. Click **Apply**. The new resource group appears under the connector.

To rename the default resource group

1. In Software AG Designer, click **Window > Perspective > Open Perspective > Other > CloudStreams Development** to open the CloudStreams Development perspective.
2. Right-click on the connector and select **Rename**.
3. In the **Rename Resource Group** dialog box, provide a new Resource Group name. By default, it displays the default Resource Group name. The **References** field displays the existing references for the Resource Group.
4. Select **Update References** to refactor the references. Click **Apply** to create a new namespace and keep the earlier assets in the newly created namespace. If you do not select this option and click **Apply**, only the display name of the Resource Group will be changed.

> **Note:** The **References** folder stores the services and the DocTypes used by a Resource Group.

REST Connector

REST (Representational State Transfer) is an architectural style that requires web applications to support the HTTP GET, POST, PUT, and DELETE methods and to use a consistent, application-independent interface.

Keep the following points in mind when creating a REST resource:

- The fully-qualified name of the resource must be unique across the entire webMethods Integration Server namespace.
- You can select the HTTP methods for which you want to create a service.
- The format of the request URI depends on the location of the resource.

Creating a REST Resource

Once you have created a **Resource Group**, any number of resources can be created under this **Resource Group**. This screen allows you to create a REST resource.

To create a REST resource

1. In the CloudStreams Development perspective, right-click on the **Resource Group** and then click **New Resource**.

   The **Create New Resource** dialog box appears.

2. The **Endpoint** field displays the login endpoint URL to initiate communication with the SaaS provider and cannot be edited.
3. In the **Name** field, type a name for the REST resource using any combination of letters, numbers, and/or the underscore character.
4. In the **Path** field, type the path that you want to use for the resource. Each REST resource derives its path from the namespace of the REST resource. For example, if the REST resource is named myREST.myRESTResource, the path is "/myREST.myRESTResource".
5. Select the **Methods**. Resources will be generated for the selected HTTP methods. Select **All** to create resources for all the HTTP methods.
6. Click **Apply**.

   Software AG Designer creates the resource folder.

7. On the **General** pane, the **Resolve references in the request/response body** option is applicable only if you have already defined abstract objects. If selected, it means that the resource is of COMPLEX type and while creating the connector service, references will be resolved with the abstract object definition. Select **Hide resource from end users** if you do not want to display the resource to the connector user.

Creating a Resource Request Parameter

To add a resource request parameter

1. On the Resource **Request** pane, click the **Parameter** tab and then click **Add...** to add a resource parameter. In the **Create Resource Parameter** dialog box, enter the parameter name and the value needed to access the resource. In the **Style** field, select the parameterization style of the request. Currently, four parameter styles are supported: URI_CONTEXT, QUERYSTRING_PARAM, CFG_PARAM, and FORM_ENCODED_PARAM. Refer the following section on **Parameter Styles**. Select **Required**, if you want the parameter to be made mandatory for users. Select **Editable**, if you want the connector user to edit the value.

   CloudStreams REST connector services allow you to set parameters which become part of the outgoing request.

   REST services rely on HTTP methods like GET, POST to make request to a SaaS provider, so the parameters are closely tied to these HTTP methods, where they are sent as part of these HTTP method requests.

   The parameters are typically part of the HTTP URI which may be of the following format:

   [scheme:][//authority][path][?query]

   CloudStreams supports the following parameter styles:

Type	URI Scope
URI_CONTEXT	path
QUERYSTRING_PARAM	query
CFG_PARAM	authority
FORM_ENCODED_PARAM	body

   URI_CONTEXT parameters are passed as the path component of a REST resource URI, and the parameter names correspond to the URI path variable names specified in the {} annotation. For example, a sample request might look like this:

   ```
   services/async/25.0/job/{jobId}
   ```

   In the above sample request, the URI path variable name `jobId` is specified as a parameter to the job resource. The annotation {} is set to the variable name `jobId`. At service run time, the variable is substituted with its runtime value to form the dynamic path.

   A sample resource definition with URI_CONTEXT parameter:

```
<resource name="Job" method="POST" path="services/async/25.0/job/{jobId}">
  <parameters>
     <parameter name="jobId" isRequired="true" style="URI_CONTEXT"/>
  </parameters>
</resource>
```

QUERYSTRING_PARAM parameters are passed as the query component of a REST resource invocation request.

The following example demonstrates a typical HTTP GET request with parameters that form a query string of the resource URI.

```
GET  /status?key1=value1&key2=value2 HTTP/1.1
Host: www.softwareag.com
Content-Length: 0
```

Notice that the parameters are added to the path after a "?", and specified as ampersand (&) separated list of key-value pairs, with the corresponding Content-Length set accordingly. The key and values passed as parameters are URL-encoded by CloudStreams.

A sample resource definition with QUERYSTRING_PARAM parameter:

```
<resource name="GetStatus" method="GET" path="/status">
  <parameters>
     <parameter name="key1" isRequired="true" style="QUERYSTRING_PARAM"/>
     <parameter name="key2" isRequired="false" style="QUERYSTRING_PARAM"/>
  </parameters>
</resource>
```

CFG_PARAM parameters signify an internal contract between the connector tier and the connecter-specific authentication scheme along with the virtual runtime layer.

The allowed set of parameter name, which can be specified as CFG_PARAM depends on the authentication scheme used for the connector definition.

Example

`aws.bucketName` is a parameter which is used by the Amazon Authentication Scheme version 3 for specifying the dynamic host based on the parameter value.

While executing a resource with such parameter, the service endpoint is prefixed with the appropriate bucket name.

```
<resource name="GetBucket" method="GET" path="/">
  <parameters>
     <parameter name="aws.bucketName" isRequired="true" style="CFG_PARAM"/>
   </parameters>
</resource>
```

If you create a parameter with style **FORM_ENCODED_PARAM** or change an existing parameter style to **FORM_ENCODED_PARAM**, all assignments and the document type in the request body will be removed and these changes cannot be restored.

During run time, all the parameters with style **FORM_ENCODED_PARAM** are passed in the body of a resource request.

A sample resource definition with FORM_ENCODED_PARAM parameter:

```
<resource method="POST" path="/rest/v1/customobjects.json" type="SIMPLE"
name="postCUSTOMOBJECTNAMEJSON" displayName="Sync Custom Objects">
            <description>Inserts, updates, or upserts custom
```

```
   object records to the target instance.</description>
            <parameters>
                <parameter name="customObjectName" isRequired="true"
    style="FORM_ENCODED_PARAM" dataType="String"/>
            </parameters>
            <request name="stream" messageFormatterType=
   "application/x-www-form-urlencoded" builderType=
   "vnd.sag.cloudstreams+binary"/>
            <responses>
                <response name="post200" code="200"
       messageFormatterType="application/octet+idataoref"
    builderType="application/json" documentRef=
    "swaggertest1.swaggertest1service.doctypes:
    ResponseOfCustomObject"
excludeRoot="true" dimension="0"/>
            </responses>
        </resource>
```

2. Click **Apply**.

Creating a Resource Request Header

To add a resource request header

1. On the Resource **Request** pane, click the **Header** tab and then click **Add...** to create a resource header. In the **Create Resource Header** dialog box, enter the parameter name and the value needed to access the resource. You can also assign a value from a **Connection Parameter**, a **Service**, or from **Parameters** by clicking the **Assign From** drop-down arrow. Select **Add resource header in all the resources** if you want to add the resource header in all the connector resources. Select **Enable editing** if you want the connector user to modify the value. Select **Required** if you want the resource header to be made mandatory for connector users.
2. Click **Apply**.

Adding a Resource Request Body

To add a resource request body

1. On the **Request** pane, click the **Body** tab and then right-click to add a new **Document Reference**. If the body is an array, select **Document Reference List**. Click **Browse**, select a **Document Reference**, and then click **Next**.
2. In the **Configure** section, click **Add Parent** if you want to add a root element to the body. The root element of the payload or document type will have the specified new root element. By default, the **Add Parent** option is cleared. See "Handling JSON representations of REST resources" on page 115 for more information.
3. Click **Finish**.

> **Note:** To assign values to a leaf element, right-click the leaf element and select **Assign From**. For more information, see "Assigning Override Values" on page 119

Creating a Resource Response Header

To add a resource response header

1. On the **Response** pane, click the **Header** tab and then click **Add...** to create a resource header. In the **Create Resource Header** dialog box, enter the header name and the value needed to access the resource. You can also assign a value to a **Connection Parameter** by clicking the **Assign To** drop-down arrow. Select **Enable editing** if you want the connector user to modify the value. Select **Required** if you want the resource header to be made mandatory for connector users.

   Note: If there are any references, the reference name for all the references in the DocType will be displayed in the request/response body.

2. Click **Apply**.

Adding a Resource Response Body

To add a resource response body

1. On the **Response** pane, click the **Body** tab and then right-click to add a new **Document Reference**. If the body is an array, select **Document Reference List**. Click **Browse**, select a **Document Reference**, add the **Response Code**, and then click **Next**.
2. In the **Configure** section, click **Add Parent** if you want to add a root element to the body. The root element of the payload or document type will have the specified new root element. By default, the **Add Parent** option is cleared. See "Handling JSON representations of REST resources" on page 115 for more information.
3. Click **Finish**.

Handling JSON representations of REST resources

If you are creating resources which have JavaScript Object Notation (JSON) data representation and you want to have your outgoing requests in JSON data format and response received in JSON format, you must set the root element correctly. Ensure that you create your IS document representations in accordance with the structure of the data.

JSON data format typically has data organized in key:value pairs enclosed within { } brackets. A JSON data snippet may not have a root element, so if you want to send requests using an IS document (whose name represents a data root), you need to exclude that root element because you may want only the child key and value elements to be sent. Similarly, in response, you may want to exclude or consider the root element based on the representation of the resource from the SaaS provider. To handle such cases, consider the **Add Parent** option.

To add a root element

1. From the **Request** or **Response Body** panel, right-click and select a document type.
2. Click **Next** to configure the request or response body.

   The request or response body dialog box appears. By default, the name of the doctype is displayed.
3. In the **Configure** section, click **Add Parent** to add a root element to the body of the request or response. By default, the **Add Parent** option is cleared.

   The root element of the payload or document type will have the specified new root element. For requests, if the **Add Parent** option is not selected, outgoing requests will have key:value pairs without the root element, for example, data - {"first": "John", "last": "Doe"}. For responses, it means that the root is excluded in the response.

   Note: If you add the parent, only then the name is added as the root element in the body and the root element is added, else the request/response name is not added as the root element.

If the JSON response has just key:value pairs and no top level root element, you can have a document type with custom root, having the expected key:value pairs. In the resource response, **Add Parent** should be unchecked. For example, {"first": "John", "last": "Doe", "age": 20, "sex": "M", "salary": 50000, "registered": true }. In this example, the parent is not required.

If the JSON response has key:value pairs with a top level parent root present, you can select the **Add Parent** option in response as your document matches the response. For example, { "error": { "errors": [{ "domain": "local", "reason": "credError", "message": "Invalid Credentials" }], "code": "401", "message": "Invalid Credentials" }}. This example already has a root - error, so your IS document type can be mapped if its root name is error too. Within it, the child elements appear as in the response data. It is important to create appropriate IS documents and set the correct property value for the responses to be mapped correctly.

SOAP Connector

The SOAP Connector wizard generates a WSDL-based connector. In the connector details screen, specify the file or URL path for the WSDL. The WSDL will be parsed and all artifacts such as operations and document types will be created in the provider package.

See the following sections while creating a SOAP connector:

- "Creating a Provider Project" on page 98
- "Creating a Connector" on page 100
- "Editing Connection Configuration and Authentication Groups" on page 102

Creating a SOAP Request Parameter

To add the SOAP Request Parameter

1. On the **Request** pane, click the **Parameter** tab and then click **Add...** to add a parameter. Enter the parameter name and the value needed to access the operation. Select **Required** if you want the parameter to be made mandatory for users. Select **Editable** if you want the connector user to edit the value.
2. Click **Apply**

   The parameter appears in the table. Right-click and select **Assign From** to assign values to the parameter. See "Assigning Override Values" on page 119 for more information.

SOAP Request Header

To view the SOAP Request Header

1. On the **Request** pane, click the **Header** tab. The Header section displays the SOAP Request Header Document Types for the selected SOAP operation.
2. You can assign a value to a leaf element from a **Connection Parameter**, a **Service**, from **Parameters**, or assign a **Literal** by clicking the **Assign From** drop-down arrow and selecting the required option. See "Assigning Override Values" on page 119 for more information.

SOAP Request Body

To view the SOAP Request Body

1. On the **Request** pane, click the **Body** tab. The Body section displays the SOAP Request Body Document Type for the selected SOAP operation.
2. You can assign a value to a leaf element from a **Connection Parameter**, a **Service**, from **Parameters**, or assign a **Literal** by clicking the **Assign From** drop-down arrow and selecting the required option. See "Assigning Override Values" on page 119 for more information.

SOAP Response Header

To view the SOAP Response Header

1. On the **Response** pane, click the **Header** tab to view the response headers for the selected SOAP operation.

Note: You cannot add a Response Header. Headers will appear in this section only if there are response headers defined in the WSDL for the selected operation.

SOAP Response Body

To view the SOAP Response Body

1. On the **Response** pane, click the **Body** tab to view the response body for the selected SOAP operation.

Note: You cannot add a Response Body. A response body will appear only if there is a response body defined in the WSDL for the selected operation.

Advanced Configurations

CloudStreams allows you to modify the behaviour about how properties take values at run time. You can assign override values from Header, Connection Parameter, Service, Parameters, and Literal to Connection group and Authentication group properties. You can also assign override values to resource request parameters, resource request headers, and from response headers. When you assign an override value, you can either hard code a specific value, or assign a default value to a variable, that is, a value that is only assigned if the variable is null at run time, or assign a value of another variable by referencing the variable. See "Assigning Override Values" on page 119 for more information.

CloudStreams supports two abstract object types: BusinessObject and SchemaObject. A Business Object represents business entities such as an invoice or a transaction. Some SaaS providers expose these business entities, and for those back ends, you need to create abstract objects so that you can dynamically create the signatures.

So for back ends to represent business entities correctly and also to reduce the effort needed to get information for each entity, you can abstract the back end entity by defining appropriate abstract object types. You need to create a Document Type (DocType) that will be used to create the abstract object type and to get information about those objects, you must also implement two look up services. The look up services are like dynamic calls to the back end to list and describe the back end entities.

CloudStreams currently supports the data types String and Record. CloudStreams also provides a parameter formatter which enables you to format input data into a custom format as desired. Alternatively, in cases where the default parameter formatter is not sufficient to achieve the desired parameter behavior, you can write a custom implementation for the parameter formatter.

Assigning Override Values

CloudStreams allows you to modify the behaviour about how properties take values at run time. You can assign override values from Header, Connection Parameter, Service, Parameters, and Literal to Connection group and Authentication group properties. You can also assign override values to request parameters, request headers, and from response headers. When you assign a value, you can either hard code a specific value or assign a default value to a variable, that is, a value that is only assigned if the variable is null at run time, or assign a value of another variable by referencing the variable.

Note: You should assign values only if you want to change the default behaviour about how the properties take the values at run time. Further, when you assign values to properties in the **Connection** group, the properties appear as **Global Variables** under the **Resource Group**.

CloudStreams allows variable sources for target types. The following table describes the allowed source and target assignments:

- Y - Allowed
- X - Not Allowed

Source	Header	Connection Parameters	Service	Parameter	Literal (User Typed Value)
Target					
Header	X	Y	Y	Y	Y
Connection Parameters	Y, only if there is a global header.	Y	Y	Y	Y
XPath	Y	Y	Y	Y	Y
Parameter	X	Y	Y	X	X

The following are the allowed types and their descriptions:

- **Header** - Typically defined in REST Connector to represent a HTTP Header.

- **Connection Parameter** - Represents any Connection parameter, qualified with the group prefix, for example, cr.username represents the “username” field value defined under the “Credentials” group.
- **Service** - Represents a service, which implements the wm.cloudstreams.service.common.lookup.specs:mapServiceSpec
- **Parameter** - Interaction (Resource) parameter. It represents a run-time value of system or user specified parameter, within an interaction scope. Currently, system parameters can only be used as a SOURCE. A system parameter starts with the “core” prefix, for example, core.serviceName means the functional area/service under which the interactions (resource) are defined.
- **Literal** (User inserted value) - A constant value.
- **XPATH** - Represents a field path in an Integration Server document.

The following table describes the allowed assignment types and their descriptions:

<table>
<tr><th>Type Name</th><th>Description</th><th>Details</th></tr>
<tr><td>HEADER</td><td>HTTP Header</td><td>Typically defined in a REST Connector to represent an HTTP Header.</td></tr>
<tr><td>CONNECTION PARAMETER</td><td>Connection Parameter</td><td>Represents any Connection parameter, which should be qualified with the group prefix.
For example, cr.username represents the field "username" value defined under the "credentials" group.
<table>
<tr><th>Group Name</th><th>Group Prefix</th><th>Is a Required Group?</th></tr>
<tr><td>Connection</td><td>cn</td><td>Yes</td></tr>
<tr><td>Protocol</td><td>pr</td><td>No</td></tr>
<tr><td>Credential</td><td>cr</td><td>No</td></tr>
<tr><td>OAuth</td><td>oa</td><td>No</td></tr>
<tr><td>Request Header</td><td>rh</td><td>No</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Type Name</th><th>Description</th><th>Details</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Group Name</th><th>Group Prefix</th><th>Is a Required Group?</th></tr>
<tr><td>Amazon Web Services
(aws_v2, aws_s3, aws_v4)</td><td>aws</td><td>No</td></tr>
<tr><td>Custom</td><td>cx</td><td>No</td></tr>
</table>
</td></tr>
<tr><td>XPATH</td><td>Signature Field Value</td><td>Represents a field path in an Integration Server document.</td></tr>
<tr><td>SERVICE</td><td>Integration Server Service</td><td>Represents a service, which implements the wm.cloudstreams.service.common.lookup.
specs:mapServiceSpec</td></tr>
<tr><td>PARAMETER</td><td>Interaction (Operation/ Resource) Parameter</td><td>Represents a run-time value of system or user specified parameter, within an interaction scope. A system parameter starts with a prefix "core". Currently, a system parameter can only be used as a SOURCE. The following table lists the system parameters:
<table>
<tr><th>Parameter Name</th><th>Description</th><th>Type</th></tr>
<tr><td>core.connectorID</td><td>Unique connector identifier.</td><td>String</td></tr>
<tr><td>core.serviceName</td><td>Functional Area/ Service under which the interactions (operation/ resource)</td><td>String</td></tr>
</table>
</td></tr>
</table>

Type Name	Description	Details

Parameter Name	Description	Type
	are defined.	
core.connectionAlias	Connection Alias used by the connector service.	String
core.virtualServiceName	Virtual Service Name used by the connector service.	String
core.interactionName	Interaction (operation/ resource) name of the connector service.	String
core.businessObject	Business Object name (if any) used by the connector service	String
core.objectFieldList	Business Object Field Names (if any) used	String[ ]

<table>
<tr><th>Type Name</th><th>Description</th><th>Details</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Parameter Name</th><th>Description</th><th>Type</th></tr>
<tr><td></td><td>by the connector service</td><td></td></tr>
</table>
Any new user-defined parameter within the connector definition or using the connector service (during design time), can be directly referred to by its name, without any prefix.
</td></tr>
<tr><td>LITERAL</td><td>User inserted value</td><td>Represents any text value.</td></tr>
</table>

Note: Assign actions are enabled only if there are arguments. For example, you will not be able to define arguments to pass to a service if there are no arguments in the selected service.

To assign an override value to a property/parameter

1. Right-click on the row that has the property/parameter to which you want to assign a value. The assign icon denotes that a value can be assigned to the property.
2. Select **Assign From** and then select the source.
3. Select either the **Header**, **Connection Parameter**, **Service**, **Parameters...**, or **Literal** option.
4. In the displayed **Assign** window, select the Header, Connection Parameter, Service, Parameters, or Literal. Select the **Enable Editing** option if you want the connector user to change the assigned value. For Services, you can enter the input, delimiter, or any other arguments.

   After the value is assigned, the value appears in the **Override Value** column and the properties appear under the **Global Variables** folder. The icon shows that the value has been assigned. You can right-click on the icon and select **Unassign** to remove the value. See "Assigning Values from a Service" on page 123 for information on how to assign override values from a service.

Assigning Values from a Service

CloudStreams allows you to modify the behaviour about how properties take values at run time. You can assign values from a service to Connection group and Authentication group properties and also to resource request parameters and resource request headers.

Note: You should assign values only if you want to change the default behaviour about how the properties take the values at run time. Further, when you assign values to properties in the **Connection** group, the properties appear as

Global Variables under the **Resource Group**. See "Assigning Override Values" on page 119 for more information.

Note: Assign actions are enabled only if there are arguments. For example, you will not be able to define arguments to pass to a service if there are no arguments in the selected service.

To assign a value from a service

1. Right-click on the row that has the assign icon  .
2. Select **Assign From** and then select **Service**.
3. In the **Assign Service** dialog box, select the service listed in the **Available services** pane. You can select the **Enable Editing** option if you want the connector user to change the assigned value.
4. Click **Next**.
5. Enter the argument values. To enter the values for **input** or **delimiter**, right-click on the **Assign** column and select **Connection Parameter** or **Parameters.....** Click **Add** to add any other arguments in the **Service Argument** dialog box. After the argument values are assigned, the values appear in the **Assign Service** dialog box.
6. Click **Finish**.
7. The values appear in the **Override Value** column and the properties appear under the **Global Variables** folder. The icon  appears, showing that the values have been assigned. You can right-click on the icon and select **Unassign** to remove the values.

Abstract Object Definition

CloudStreams supports two abstract object types: BusinessObject and SchemaObject. A Business Object represents business entities such as an invoice or a transaction. Some SaaS providers expose these business entities, and for those back ends, you need to create abstract objects so that you can dynamically create the signatures.

So for back ends to represent business entities correctly and also to reduce the effort needed to get information for each entity, you can abstract the back end entity by defining appropriate abstract object types. You need to create a Document Type (DocType) that will be used to create the abstract object type and to get information about those objects, you must also implement two look up services. The look up services are like dynamic calls to the back end to list and describe the back end entities.

At run time, the abstract object type you have defined will be filled up by the two lookup services. The **boList** lookup service will allow you to list the back end supported business entities and the **boDescribe** lookup service will retrieve the fields corresponding to each business entity.

If Schema Objects are already defined in the schema file provided by the SaaS provider, document types for all the objects that are present in schema file are automatically

created. For Schema Objects, you need to define the abstract objects and implement the **soList** and **soDescribe** look up services.

To define an abstract object

1. Create a DocType that has an abstract field, for example, *any. The DocType has a node of Object or ObjectList. The DocType can be referenced from another DocType that could be used in the body of the Request/Response of a resource.
2. Right-click on a **Resource Group** or **Operation Group/Services** (for SOAP) and select **Define Object Type...**.
3. In the **Document Type** field, click **Browse** and select the DocType that defines the structure of the abstract object. This is the DocType created in Step 1.

   Note: If you select **Derive objects automatically**, abstract objects will be derived automatically from the schema if a schema file exists.

4. For REST, if you have selected **Derive objects automatically**, click **Next** to view the **Select Concrete Object** dialog box, which displays the list of concrete objects. Select a concrete object that is the default implementation for the abstract object. The abstract object created is a Schema Object.

   For SOAP, the **Operation List** dialog box appears. The **Operation List** dialog box displays the SOAP operations related to the selected document type in Step 3. If you select an operation in the list, that operation will be marked as a complex operation.
5. For REST, if you have not selected **Derive objects automatically**, click **Next** to see the **Assign Services** dialog box. The **Assign Services** dialog box displays the list of look up services that implement the **boList** lookup type. Select the look up services that you implemented for the abstract object. If you select the **Generate objects listing the service** option, stubs for the services will be automatically generated. You can also choose existing services listed in the **Assign Services** dialog box.

   For SOAP, the **Operation List** dialog box appears. The **Operation List** dialog box displays the SOAP operations related to the selected document type in Step 3. If you select an operation in the list, that operation will be marked as a complex operation. Click **Next** to see the **Assign Services** dialog box. The **Assign Services** dialog box displays the list of look up services that implement the **boList** lookup type. Select the look up services that you implemented for the abstract object. If you select the **Generate objects listing the service** option, stubs for the services will be automatically generated. You can also choose existing services listed in the **Assign Services** dialog box.
6. Click **Next** and select the look up service that implement the **boDescribe** lookup type. If you had selected the **Generate objects listing the service** option, skip to Step 7.
7. Click **Finish**. The services appear in the **Defined Objects** pane. For every object defined, the services will be accordingly listed. For a Schema Object, the concrete object details will appear.

8. From the **Defined Objects** pane, you can select **Define Object...** to define an abstract object type, or select **Edit Object...** to modify an existing abstract object, or select **Remove** to delete an existing abstract object.

Parameter Data Types and Formatters

Parameter Data Types

CloudStreams currently supports the data types String and Record.

String data type represents a simple parameter which is an individual string key-value parameter. This is the most commonly used data type and the default type, if not specified explicitly.

Record data type handles complex parameters involving a collection of related key-value pairs. These parameters can have indices's for one or more parameter entry. Such parameters can ideally be represented with an IS document type, and configured as a "Record" data type.

Example

Consider a complex parameter structure as follows:

```
+ Filter[]
    - Name
    + Value[]
```

This needs to be represented as a query string like:

```
Filter.1.Name=instance-
type&Filter.1.Value.1=m1.small&Filter.1.Value.2=t1.micro
```

Such a parameter can be defined by creating an IS document type, and referring it as a "Record" data type parameter within the resource definition.

CloudStreams provides the capability to represent and transform such complex parameters into the desired format using its predefined formatter implementation. The resource's parameter definition can be enhanced to refer to a formatter, as follows:

```
<parameter name=”Filter” dataType=”Record”
documentRef=”documents.paramType:FilterList”
style=”QUERYSTRING_PARAM”>
    <formatter service=”wm.cloudstreams.service.util.formatters:paramFormatter”
type=”paramFormatter”/>
</parameter>
```

At runtime, the resource PATH would look like:

```
GET /?Filter.1.Name=instance-type&Filter.1.Value.1=m1.small&Filter.1.Value.2=t1.micro
```

For more details on formatters and customizing the formatter behavior, see "Parameter Formatters".

Parameter Formatters

CloudStreams provides a parameter formatter which enables you to format input data into a custom format as desired.

Alternatively, in cases where the default parameter formatter is not sufficient to achieve the desired parameter behavior, you can write a custom implementation for the parameter formatter.

The input will typically be data present in a complex structure, for example, an IS document type.

The output will be data organized in a desired output format, for example, a URL query string. The format in which the output data is generated depends on the implementation of the formatter.

Typically, you will use the QUERYSTRING_PARAM or the FORM_ENCODED_PARAM parameter types to format the input data into a desired output format, for example, a URL query string in case of QUERYSTRING_PARAM or in case of FORM_ENCODED_PARAM, a formatted query string in the Request body.

In order to form the query string, you can write a custom formatter service by implementing the bundled IS specification wm.cloudstreams.service.common.lookup.specs:formatterSpec.

At run time CloudStreams will give control to the formatter service to format the data appropriately and to return the formatted string data. Then, CloudStreams will encode the data during a post-processing step, and the final query string will be formed.

Note: Record type parameter formatter is currently not supported.

Using the Default Parameter Formatter

For convenience, CloudStreams provides a default formatter implementation that handles the most common formatter usage:

wm.cloudstreams.service.util.formatters:paramFormatter

This formatter allows you to specify a complex IS document type, and to implement the desired output. The input will typically be data in a complex structure, for example, an IS document type. The output will be a data organized in a desired output format, for example, a URL query string. The format in which the output data is generated depends on the implementation of the formatter. Typically, the formatter is used in conjunction with the QUERYSTRING_PARAM parameter type.

Example of using the default parameter formatter

Consider a SaaS back-end expecting the following parameter:

```
BlockDeviceMapping.n.DeviceName
BlockDeviceMapping.n.VirtualName
BlockDeviceMapping.n.Ebs.NoDevice
BlockDeviceMapping.n.Ebs.VolumeSize
```

where `n` represents a number.

1. Create IS document types for the above parameter definition. The IS document structure would look like this:

```
BlockDeviceMapping[]
  + DeviceName
```

```
  + VirtualName
  + Ebs
      + NoDevice
      + VolumeSize
```

2. Set the `dataType` of the parameter to `Record`.
3. Set the document reference (`documentRef`) to the newly created IS document type in step 1.
4. Specify a formatter element within the parameter element, along with the service name (*service* attribute) and type (*type* attribute).

In case the implementation contains supported arguments, specify any customization by adding the appropriate argument name and value. The parameter definition for a resource using the above, within the Connector Descriptor, would look like this:

```
<parameter name="BlockDeviceMapping" dataType="Record"
documentRef="document.paramType:BlockDeviceList" style="QUERYSTRING_PARAM">
  <formatter service = "wm.cloudstreams.service.util.formatters:paramFormatter"
type = "paramFormatter">
    <arg name="startIndex" value="1" />
    <arg name="format" value="." />
  </formatter>
</parameter>
```

Note: The default value for the type attribute within the scope of a parameter definition is "paramFormatter". This type refers to the bundled IS Specification:

```
wm.cloudstreams.service.common.lookup.specs:formatterSpec
```

The generated representation of the above parameter would look like this:

```
BlockDeviceMapping.1.DeviceName=/dev/sdj&;BlockDeviceMapping
.1.Ebs.NoDevice=true&;BlockDeviceMapping
.2.DeviceName=/dev/sdh&;BlockDeviceMapping
.2.Ebs.VolumeSize=300&;BlockDeviceMapping
.3.DeviceName=/dev/sdc&;BlockDeviceMapping
.3.VirtualName=ephemeral1
```

The default formatter can be configured with the following supported arguments:

Argument	Default Value	Description
`startIndex`	0	Determines the index values of the keys e.g. should the n present in BlockDeviceMapping.n.DeviceName start with 0 or 1 or some number of your choice.
`format`	. (period)	Specifies how the document entries be separated. For example, BlockDeviceMapping.3.DeviceName is separated by a period (.).

Implementing a Custom Parameter Formatter

In cases where the default parameter formatter is not sufficient to achieve the desired parameter behavior, you can write a custom implementation for the parameter formatter.

Implement a custom formatter as follows:

1. Write an IS service implementation, adhering to the bundled IS specification wm.cloudstreams.service.common.lookup.specs:formatterSpec.
2. The fully-qualified service namespace should be referred with the formatter service attribute under the parameter definition. Ensure that the formatter type attribute is set to "paramFormatter".
3. Optionally, any service arguments can be specified within the formatter element definition.

Example

Consider a custom service implementation:

```
my.custom.formatter:sample
```

The parameter definition for a resource using the above, within the Connector Descriptor, would look like this:

```
<parameter name="SampleParam" dataType="Record"
documentRef="my.custom.document:sample" style="QUERYSTRING_PARAM">
  <formatter service = "my.custom.formatter:sample" type = "paramFormatter">
    <arg name="separator" value=";" />
  </formatter>
</parameter>
```

Importing Packages

You can import an exported development package to another server either as a zip file or from a workspace.

To import a package

1. Right-click on the server or the provider package and select **Import**.
2. In the **Import** dialog box, select **Archive File** and specify the zip file name in the **Select an archive file** field if you want to import an archive file. Select **Import an existing project from workspace** and specify the project in the **Select the project** field if you want to import an existing project from a workspace.

> **Note:** You cannot choose a different name while importing a package. The provider is created with the same name as the zip file. While importing from a workspace, you must specify a valid source workspace location. The provider content is retrieved and the project is created in the current workspace. If a project with the same name already exists in the current workspace, an error message appears. If the project does not exist in the

current workspace but exists in the server, the provider will be retrieved from the server.

3. Click **Finish**.

   The provider and all the artifacts are created.

Exporting Packages

You can export a development package (which is still under development) or a published package (which can be installed on webMethods Integration Server). The exported development package can be imported to another server.

To export a package

1. Right-click on the provider package and select **Export**.
2. In the **Export** dialog box, select **Development Package** and specify the file name in the **To archive file** field if the package is a development package. Select **Publishable Package** if the package is already published.

   Note: You can select the connectors to export from the dialog box pane.

3. Click **Finish**.

   Note: You can click **Cancel** to stop the export process.

   The selected connector and all its assets are exported to the specified archive file.

Publishing and Unpublishing Providers

You can publish a provider to target servers.

To publish a provider

1. Right-click on the provider package and select **Publish**.
2. In the **Publish Provider** dialog box, select one or more connectors you want to publish from the provider package and click **Next**.
3. In the **List of Servers** dialog box, select the **Current or Development Server** if you want to publish the provider package on your local server. Select **Target Server(s)** to publish the connectors on to other servers.
4. Click **Finish**.

   The connectors are published to the selected servers: *IntegrationServer_directory* \packages\<*PackageName*> and the status is displayed in the log at the bottom of the screen.

Unpublish a Provider

You can unpublish an existing provider. Right-click on the provider package and select **Unpublish...**. The selected provider will be unregistered from CloudStreams.