API Gateway

Version 7.6.2

14 July 2020

Policy Developer Guide

DRAFT

This documentation describes the following Axway software:

Axway API Gateway 7.6.2

Contents

Preface

This guide describes the main API Gateway features (for example, policies, filters, and configuration options), and how to configure them using the Policy Studio graphical tool.

Who should read this guide

This guide is intended for policy developers.

Familiarity with Axway products is recommended.

How to use this guide

This guide should be used alongside the *API Gateway Policy Developer Filter Reference* and the other guides in the API Gateway documentation set.

Before you begin developing policies, review this guide thoroughly. The following is a brief description of the contents of each section:

Get started on page 23 – Describes how to get started with policy development in Policy Studio.

Manage policies on page 65 – Describes how to configure policies manually, including global policies and policy assemblies.

Web services on page 75 – Describes how to register and secure web services, and how to work with WSDL documents and XML schemas.

Messaging on page 151 – Describes how to configure messaging services.

General configuration on page 169 – Describes general configuration settings available in Policy Studio.

Manage deployments on page 208 – Describes how to deploy API Gateway configurations.

Environment configuration on page 221 – Describes how to work with API Gateway resources and libraries.

API Gateway instances on page 264 – Describes how to configure API Gateway instances.

External connections on page 359 – Describes how to configure external connections.

Extend and customize API Gateway on page 420 – Describes advanced Policy Studio features that enable you to extend API Gateway to suit your environment.

Common settings on page 430 – Describes how to configure settings that are common to many API Gateway components.

Related documentation

The AMPLIFY API Management solution enables you to create, publish, promote, and manage Application Programming Interfaces (APIs) in a secure and scalable environment. For more information, see the *AMPLIFY API Management Getting Started Guide*.

The following reference documents are also available on the Axway Documentation portal at https://docs.axway.com:

- *Supported Platforms*

  Lists the different operating systems, databases, browsers, and thick client platforms supported by each Axway product.

- *Interoperability Matrix*

  Provides product version and interoperability information for Axway products.

Support services

The Axway Global Support team provides worldwide 24 x 7 support for customers with active support agreements.

Email support@axway.com or visit Axway Support at https://support.axway.com.

See "Get help with API Gateway" in the *API Gateway Administrator Guide* for the information that you should be prepared to provide when you contact Axway Support.

Training services

Axway offers training across the globe, including on-site instructor-led classes and self-paced online learning. For details, go to: http://www.axway.com/support-services/training

Accessibility

Axway strives to create accessible products and documentation for users.

This documentation provides the following accessibility features:

Screen reader support

- Alternative text is provided for images whenever necessary.
- The PDF documents are tagged to provide a logical reading order.

Support for high contrast and accessible use of colors

- The documentation can be used in high-contrast mode.
- There is sufficient contrast between the text and the background color.
- The graphics have the right level of contrast and take into account the way color-blind people perceive colors.

Updates and revisions

This guide includes the following documentation changes.

Changes in version 7.6.2

- Updated the topic on configuring a JMS service to describe a new field that enables you to specify the maximum number of JMS sessions. For more information, see *Configure a JMS service* on page 153.
- Updated the topic on configuring a directory scanner to describe enhancements to the file processing settings. For more information, see *Configure a directory scanner* on page 345.
- Restructured the API Gateway Analytics information and added links to the new *API Gateway Analytics User Guide*.

Changes in version 7.6.1

- Added information on specifying multiple input schemas when creating data maps. For details, see *Manage data maps* on page 108.

Changes in version 7.6.0

- Removed references to the following:
  - API Gateway server-side support for Windows
  - API Gateway Appliance
  - API Gateway Analytics
- Added a new topic on how configure limits to the rates at which requests pass through API Gateway. For details, see *Configure rate limiting* on page 306.

Get started 1

This section describes how to get started with policy development in API Gateway.

Policy development with Policy Studio

Overview

This topic explains some of the main components and concepts used in API Gateway policy development, and shows examples of how they are displayed in the API Gateway management tools such as Policy Studio and API Tester.

For example, these include concepts such as filters, policies, message attributes, and listeners. For details on the core API Gateway features and architecture, see the *API Gateway Concepts Guide*. For example, this includes concepts such as API Gateway instances, groups, Node Manager, Admin Node Manager, and so on.

Policy Studio projects

A Policy Studio project is a design-time store of API Gateway configuration on a local file system, which can be deployed to a running API Gateway instance on a server. Policy Studio projects enable you to use API Gateway as a design-time repository for policies as well as a runtime engine for executing those policies.

For example, policy developers can create specific API Gateway configuration projects on their file system, edit the API Gateway policies and other configuration items, and save their changes back to the file system. They can also then deploy their local Policy Studio project configuration to a running API Gateway instance to test those changes. For more details, see *Manage Policy Studio projects* on page 169.

API Gateway instances and groups

You can use Policy Studio to configure and deploy API Gateway instances and groups. An API Gateway instance is an API Gateway capable of running on a host. You can configure various features at the instance level as listeners (for example, HTTP(S) interfaces, file transfer services, JMS services, remote hosts, and so on).

An API Gateway group is a collection of one or more API Gateway instances that share the same configuration. For more details, see the *API Gateway Concepts Guide*.

Filters

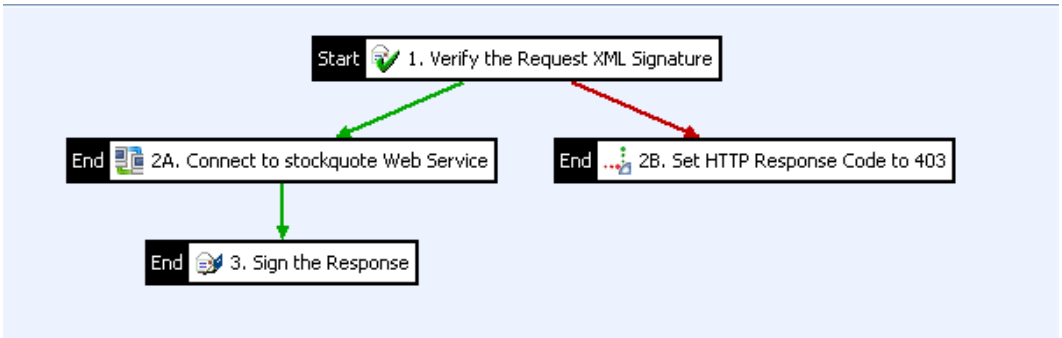
A filter is an executable rule that performs a specific type of processing on a message. For example, the **Message Size** filter rejects messages that are greater or less than a specified size. There are many categories of message filters available with the API Gateway, including authentication, authorization, content filtering, signing, and conversion. In the Policy Studio, a filter is displayed as a block of business logic that forms part of an execution flow known as a policy. The next section shows some example filters.

Policies

A policy is a network of message filters in which each filter is a modular unit that processes a message. A message can traverse different paths through the policy, depending on which filters succeed or fail. For example, this enables you to configure policies that route messages that pass a **Schema Validation** filter to a back-end system, and route messages that pass a different **Schema Validation** filter to a different system. A policy can also contain other policies, which enables you to build modular reusable policies.

In the Policy Studio, the policy is displayed as a path through a set of filters. Each filter can have only one **Success Path** and one **Failure Path**. You can use these success and failure paths to create sophisticated rules. For example, if the incoming data matches schema A, scan for attachments and route to service A, otherwise route to service B. You can configure the colors used to display success paths and failure paths in the Policy Studio **Preferences** menu. You can also specify to **Show Link Labels** (**S** or **F**).

The following shows an example policy that performs XML signature verification:



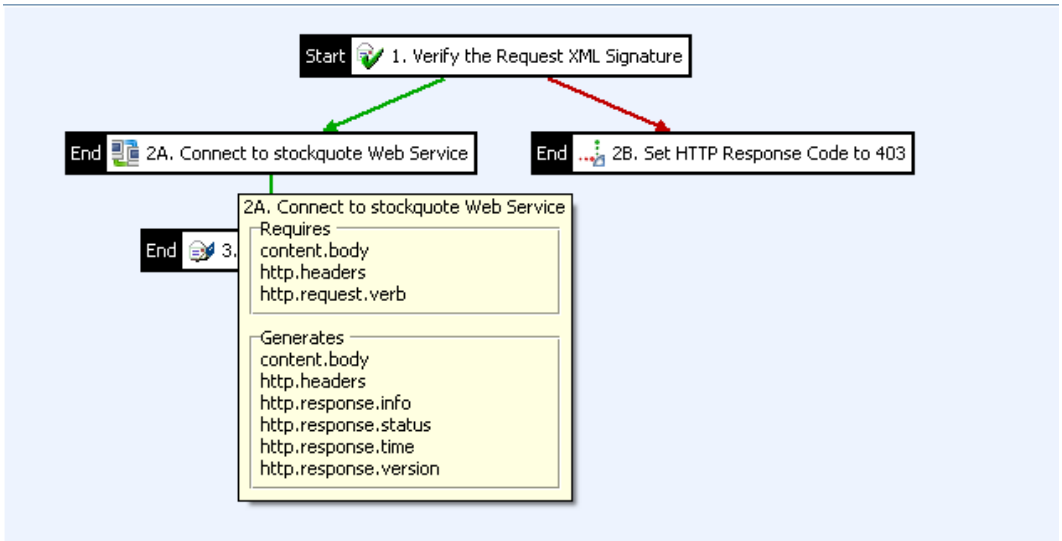
A policy must have a **Start** filter (in this case, **Verify the Request XML Signature**). Filters labeled **End** stop the execution of the policy (for example, the filter execution fails). A filter labeled **Start/End** indicates that the policy execution starts there, and that the policy stops executing if this filter fails. A policy with a single filter labeled **Start/End** is also valid.

Message attributes

Each filter requires input data and produces output data. This data is stored in message attributes, and you can access their values in API Gateway configuration using a selector syntax (for example, `${attribute.name}`).

You can also use specific filters to create your own message attributes, and to set their values. The full list of message attributes flowing through a policy is displayed when you right-click the Policy Studio canvas, and select **Show All Attributes**. You can also hover your mouse over a filter to see its inputs and outputs. The **Trace** filter enables you to trace message attribute values at runtime.

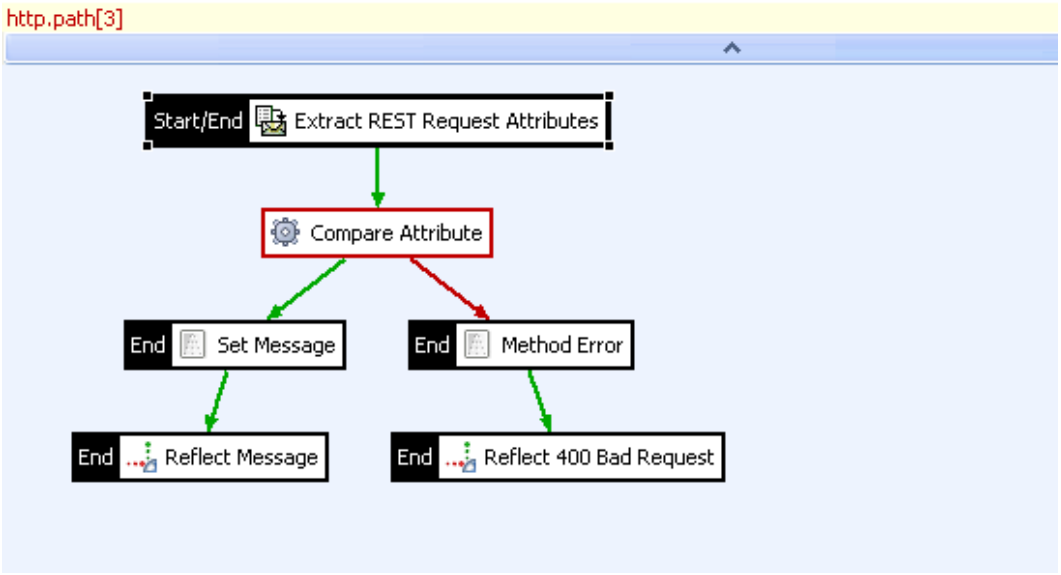
The message attribute *white board* refers to the list of attributes that are available to a particular filter at runtime of the API Gateway during the processing of requests and responses. The following example shows the attributes displayed when hovering over a **Connect to URL** filter at design time in Policy Studio:



If a filter requires an attribute as input that has not been generated in the previous execution steps, the filter is displayed in a different color in the Policy Studio (default is red). You can configure the color used to display missing attributes in the Policy Studio **Preferences** menu. Alternatively, you can also view all required attributes by right-clicking the canvas, and selecting **Show All Attributes**.

A missing attribute might indicate a problem that you need to investigate (for example, in the chaining of filters or policies, or that the policy cannot run on its own). This is often the case for reusable filters, such as those displayed in the previous example.

At the policy level, you also can click the horizontal bar at the top of the Policy Studio canvas to show the list of all attributes required as input to the entire policy. If any attributes are generated by the policy, you can click a corresponding bar at the bottom to see a list of generated attributes. The following example shows a required attribute:



Selectors

A selector is a special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata (for example, in message attributes, a Key Property Store (KPS), or environment variables). For example, the following selector returns the value of a message attribute:

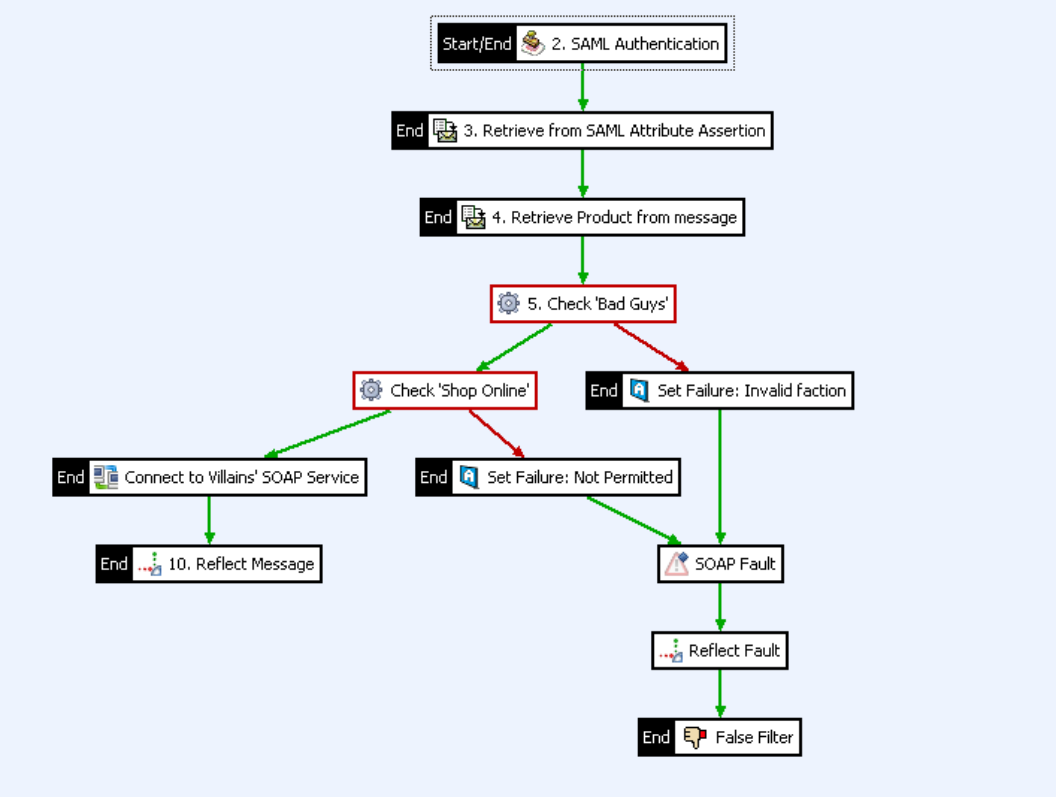
```
${http.request.clientaddr}
```

Selectors are powerful a feature for System Integrators (SIs) and Independent Software Vendors (ISVs) when extending the API Gateway to integrate with other systems. For more details on selectors, see *Select configuration values at runtime* on page 421.

Faults and errors

When a transaction fails, you can use a fault to return error information to the client application. The API Gateway provides the **SOAP Fault**, **JSON Error**, and **Generic Error** filters. By default, the API Gateway returns a very basic fault to the client when a message filter fails. You can add a specific fault filter to a policy to return more detailed error information to the client.

For example, the following screen shows an authentication policy that includes a **SOAP Fault**:



Policy shortcuts

A policy shortcut enables you to create a link from one policy to another policy. For example, you could create a policy that inserts security tokens into a message, and another that adds HTTP headers. You can then create a third policy that calls the other two policies using **Policy Shortcut** filters.

A policy shortcut chain enables you to run a series of policies in sequence without needing to create a policy containing policy shortcuts. In this way, you can create modular policies to perform certain specific tasks, such as authentication, content filtering, returning faults, or logging, and then link these policies together in a sequence using a policy shortcut chain.

Alerts

The API Gateway can send alert messages for specified events to various alerting destinations. System alerts are usually sent when a filter fails, but they can also be used for notification purposes. For example, the API Gateway can send system alerts to the following destinations:

- Email Recipient
- Check Point Firewall-1 (OPSEC)
- Local Sylsog
- Remote Sylsog
- SNMP Network Management System
- Twitter
- Windows Event Log

You can configure alert destinations, and then add an **Alert** filter to a policy, specifying the appropriate alert destination.

Policy containers

A policy container is used to group similar policies together (for example, all authentication or logging policies), or policies that relate to a particular service. A number of useful policies are provided in the **Policy Library** container (for example, policies that return faults, and policies that block threatening content). You can add your own policies to this container, and add your own policy containers to suit your requirements.

Policy contexts

Policies can execute in a specified context. For example, you can set a context by associating a relative execution path or listener with a policy. When a policy is called from another policy, the context is set to the calling policy name (for example, **Authenticate**).

In Policy Studio, you can select a context from the **Context** drop-down list at the bottom of the policy canvas. The Policy Studio then displays whether the attributes required for execution are available in that context. The **Context** list includes all connected relative paths, listeners, web services, SMTP services, and policy shortcuts that use the selected policy. Click **View navigator node** to display the selected context.

Listeners

You can define different types of listeners and associate them with specific policies. For example, listeners include the following types:

- HTTP/HTTPS
- Directory Scanner
- POP mail server connection
- JMS connection
- TIBCO Rendezvous connection

The API Gateway can be used to provide protocol mediation (for example, receiving a SOAP request over JMS, and transforming it into a SOAP/HTTP request to a back-end service). For HTTP/HTTPS listeners, policies are linked to a relative path. Otherwise, policies are linked to the listener itself. You can associate a single policy with multiple listeners.

Remote hosts

You can define a remote host when you need more control of the connection settings to a particular server. The available connection settings include the following:

- HTTP version
- IP addresses
- Timeouts
- Buffers
- Caches

For example, by default, the API Gateway uses HTTP 1.0. You can force it to use HTTP 1.1 using remote host settings. You must also define a remote host to track real-time metrics for a particular host.

Servlet applications

The API Gateway provides a web server and servlet application server that can be used to host static content (for example, documentation), or servlets providing internal services. Static content or servlets can be accessed from a policy using the **Call Internal Service** filter. This feature is not meant to replace an enterprise J2EE server, but rather to enable you to write functionality using technology such as servlets.

Service virtualization

When you register an API service or web service, and deploy it to the API Gateway, the API Gateway virtualizes the service. Instead of connecting to the service directly, clients connect through the API Gateway. The API Gateway can then apply policies to messages sent to the destination service (for example, to enable security, monitoring, and acceleration).

Start the API Gateway tools

This topic describes the prerequisites and preliminary steps. It explains how to start the API Gateway Manager administrator tool and the Policy Studio developer tool.

Before you begin

Before you start the API Gateway tools, do the following:

Install the API Gateway and Policy Studio

If you have not already done so, see the *API Gateway Installation Guide*.

Configure a managed domain

If you have not already created a domain, use the `managedomain` script to configure a domain. You should ensure that the Admin Node Manager and an API Gateway instance are running. For more information on managing domains, see the *API Gateway Administrator Guide*.

Launch API Gateway Manager

To access the web-based API Gateway Manager administration tools, perform the following steps:

Note You must ensure that the Admin Node Manager is running before you can access the web-based API Gateway Manager tools.

1. Enter the following URL:

   ```
   https://HOST:8090/
   ```

   `HOST` refers to the host name or IP address of the machine on which API Gateway is running (for example, `https://localhost:8090/`).
2. Enter the administrator user name and password configured at installation time.
3. Click the appropriate button in the API Gateway Manager page in the browser. The **Dashboard** view is displayed by default.

The API Gateway Manager includes the following main views:

- **Dashboard**: System health, traffic summary, and topology (domain, hosts, API Gateways, and groups).
- **Traffic**: Message log and performance statistics on the traffic processed by API Gateway. For example, all HTTP, HTTPS, JMS, File Transfer, and Directory messages processed by API Gateway.

- **Monitoring**: Real-time monitoring of all the traffic processed by API Gateway. Includes statistics at the system level and for services invoked and remote hosts connected to.
- **Logs**: API Gateway trace log, transaction log, and access log files.
- **Events**: API Gateway transaction log points, alerts, and SLA alerts.
- **Settings**: Enables you to configure dynamic API Gateway logging, user roles, and credentials.

For more information on administering API Gateway, see the *API Gateway Administrator Guide*.

Start Policy Studio

To start the Policy Studio tool used to create and manage policies, perform the following steps:

1. In your Policy Studio installation directory, enter the `policystudio` command.
2. In Policy Studio, select **File > New Project**, and follow the steps in the wizard. For more details, see *Create a Policy Studio project* on page 31.

   Alternatively, if a project has already been created, select **File > Open Project**, or click a link to the existing project on the Policy Studio landing page. For more details, see *Manage API Gateway connections* on page 172.

Policy Studio enables you to perform the full range of API Gateway configuration and management tasks. This includes tasks such as develop and assign policies, import services, optimize API Gateway configuration settings, and manage API Gateway deployments.

For more details on using the Policy Studio to manage API Gateway processes and configurations, see *Manage API Gateway deployments* on page 208.

Create a Policy Studio project

A Policy Studio project is a design-time store of API Gateway configuration on a local file system, which can be deployed to a running API Gateway instance on a server. This topic explains how to create a new Policy Studio project from the following starting points:

- Template configuration (for example, factory)
- Deployment package (`.fed` file)
- Policy and environment packages (`.pol` and `.env` files)
- API Gateway instance
- Existing API Gateway configuration

For details on opening an existing project, see *Manage API Gateway connections* on page 172.

Launch the New Project wizard

To launch the **New Project** wizard in Policy Studio, select **File** > **New Project** in the main menu. Alternatively, click **New Project** on the welcome page.

Enter project details

Enter the following in the **Project Details** screen:

- **Name**: Enter the Policy Studio project name. This field is required.
- **Use default location**: Select whether to use the default location. This is selected by default.
- **Location**: Enter or browse to the project location (for example, `C:\Users\jane\apiprojects\my_test_project`). When **Use default location** is selected, the default location is `${user.home}/apiprojects/${project.name}`.

Click **Next**.

Select a project starting point

Select one of the following in the **Project starting point** window:

- **From a template configuration**
- **From a .fed file**
- **From .pol and .env files**
- **From an API Gateway instance**
- **From existing configuration**

New project from a template configuration

To create a new project based on template configuration:

1. Select the configuration template to use as a starting point:
    - **Factory template**

      Blank template that contains default factory configuration.
    - **Factory template with samples**

      Blank template that contains sample policies and other entities. This is for creating a project used for demo purposes.
    - **Team Development – Common Project (with Server Settings)**

      Blank template that contains **Server Settings**. This is for creating your common project only. This should include policies common to multiple API projects (for example, authentication and authorization). Your API Gateway configuration must contain only one project with **Server Settings**. All other projects must have no **Server Settings**.
    - **Team Development – API Project (without Server Settings)**

Blank template that does not contain **Server Settings**. This is for creating all your projects (except your common project). Your API Gateway configuration must contain only one project with **Server Settings**. All other projects must have no **Server Settings**.

Note The team development templates are only available if team development is enabled in Policy Studio. For more details on enabling team development, see *Policy Studio preferences* on page 180. For more details on team development, see the *API Gateway DevOps Deployment Guide*.

2. Click **Finish**.

Note There is no project template for API Manager configuration. For more details, see *Create API Manager projects* on page 35.

New project from a .fed file

To create a new project based on a deployment package (`.fed` file):

1. Configure the following:
   - **File**: Enter or browse to the location of an API Gateway deployment package (`.fed` file). For more details on deployment packages, see the *API Gateway DevOps Deployment Guide*.
   - **Passphrase**: Enter the encryption passphrase for the `.fed` file if one has been configured.
2. Click **Finish**.

New project from a policy package (.pol) file

To create a new project based on a policy package (`.pol` file):

1. Configure the following:
   - **Policy Package**: Enter or browse to the location of an API Gateway policy package (`.pol` file). This is required.
   - **Environment Package**: Enter or browse to the location of an API Gateway environment package (`.env` file). This is optional.
   - **Passphrase**: Enter the encryption passphrase if one has been configured.
   - **Advanced**: Click to expand, and select whether to **Allow unresolved references in policy package**. This setting is optional.
2. Click **Finish**.

For more details on policy packages and environment packages, see the *API Gateway DevOps Deployment Guide*.

New project from an API Gateway instance

To create a new project based on an API Gateway instance:

1. In the **Saved Session**s section, select the server session to use from the list. You can edit a session name by entering a new name and clicking **Save**. You can also click the appropriate button to **Add**, **Clone**, or **Remove** saved sessions.
2. In the **Connection Details** section, configure the following:
    - **Host**:
      Enter the server host to connect to. The default is `localhost`.
    - **Port**:
      Enter the port to connect on. The default Admin Node Manager port is `8090`.
    - **User name**:
      The deployment service is protected by HTTP basic authentication. Enter the administrator user name to use to authenticate to the server. For more details, see *Manage admin users* on page 216.
    - **Password**:
      Enter the password for the administrator user.
3. Click **Advanced** to enter the **URL** of the deployment service exposed by the server. This setting is optional. The default Admin Node Manager URL is `https://localhost:8090/api`.
4. Click **Next** to download the configuration from the server instance.

   If advisory warnings are configured, you must click **Next** again before downloading the configuration. For more details on advisory warnings, see the *API Gateway Administrator Guide*.
5. Configure the following in the **Make a server connection** window:

- **Group**:
  Select the API Gateway group from the list (for example, **QuickStart Group**), and select the server instance in the panel below (for example, **QuickStart Server**).
- **Passphrase**:
  Enter the API Gateway passphrase if one has been configured.

6. Click **Finish**.

Note To manage API Gateways in your network, you must connect to the Admin Node Manager server URL.

New project from existing configuration

To create a new project based on an existing configuration directory:

1. Configure the following:
    - **File**: Enter or browse to the location of the configuration directory (for example, `INSTALL_DIR\apigateway\conf\fed` is the Admin Node Manager configuration directory).

- **Passphrase**: Enter the encryption passphrase if one has been configured.

2. Click **Finish**.

Create API Manager projects

To create a project with API Manager configuration, perform the following steps:

1. Create a new project using any of the options detailed in *Select a project starting point* on page 32.
2. Select **File > Configure API Manager**. For more details on configuring API Manager in Policy Studio, see the *API Manager User Guide*.

Alternatively, if you already have API Manager configured you can create the project from the existing API Gateway configuration directory.

Note There is no project template currently available for API Manager configuration.

Automatic upgrade of project configuration from earlier versions

If you create a project from a `.fed` file, a `.pol` file, or a configuration directory from an earlier API Gateway version, the configuration is automatically upgraded to API Gateway 7.6.2 when the new project is created. The configuration directory can be from an API Gateway instance, Admin Node Manager, or API Gateway Analytics.

Note This feature does not apply when creating a project from an API Gateway instance. You can only download configuration from an API Gateway instance if the Policy Studio version is the same as the API Gateway version.

For more details on upgrading, see the *API Gateway Upgrade Guide*.

Get started with routing configuration

Overview

This topic describes how to configure API Gateway to send messages to external services. API Gateway offers a number of different filters that can be used to route messages. Depending on how API Gateway is perceived by the client, different combinations of routing filters can be used.

For example, API Gateway can act both as a proxy and as an endpoint (in-line) server for a client, depending on how the client is configured. In each case, the request received by API Gateway appears slightly different, and API Gateway can take advantage of this when routing the message onwards. Furthermore, API Gateway can provide *service virtualization* by shielding the underlying hierarchy of back-end web services from clients.

This topic explains how clients can use API Gateway both as a proxy and as an endpoint server. It then shows how *service virtualization* works. When these basic concepts are explained, this topic helps you to identify the combination of routing filters that is best suited to your deployment scenario.

Proxy or endpoint server

API Gateway can be used by clients as both a proxy server and an endpoint server. When a client uses API Gateway as a proxy server, it sends up the complete URL of the destination web service in the HTTP request line. API Gateway can use the URL to determine the host and port to route the message to. The following example shows an HTTP request received by API Gateway when acting as a proxy for a client:

```
POST http://localhost:8080/services/getHello HTTP/1.1
```

Alternatively, when API Gateway is acting as an endpoint (in-line) server, the client sends the request directly to API Gateway. In this case, the request line appears as follows:

```
POST /services/getHello HTTP/1.1
```

In this case, only the path on the server is specified, and no scheme, host, or port number is included in the HTTP request line. Because this information is not provided by the client, API Gateway must be explicitly configured to route the message on to the specific destination.

Service virtualization

It is sometimes desirable to shield the underlying structure of the directory hierarchy in which web services reside from external clients. You can do this by providing a mapping between the path that the client accesses and the actual path at which the web service resides.

For example, suppose you have two web services accessible at the `/helloService/getHello` and `/financeService/getQuote` URIs. You might wish to hide that these services are deployed under different paths, perhaps exposing them under a common `/services` base URI (for example, `/services/getHello` and `/services/getQuote`). The client is therefore unaware of the underlying hierarchy (for example, directory structure) of the two web services. This is termed *service virtualization*.

Choose the correct routing filters

To choose the correct combination of routing filters, you must first consider how your client will use API Gateway (as a proxy or as an endpoint). Additionally, you must consider if you require virtualization. This section includes several use cases that demonstrate how different filters can be used in different scenarios.

Proxy or endpoint

- If the client is using API Gateway as a proxy server, see *Use case 1: Proxy without service virtualization* on page 37 or *Use case 2: Proxy with service virtualization* on page 38, depending on whether *service virtualization* is required.
- Alternatively, if the client using API Gateway as the endpoint of the connection (as an in-line server), see *Use case 3: Endpoint without service virtualization* on page 39 or *Use case 4: Endpoint with service virtualization* on page 40.

Service virtualization

- To shield the hierarchy of protected web services by exposing a *virtual* view of these services, see *Use case 2: Proxy with service virtualization* on page 38, *Use case 4: Endpoint with service virtualization* on page 40, and *Use case 5: Simple redirect* on page 41.
- If *service virtualization* is not important, see *Use case 1: Proxy without service virtualization* on page 37 and *Use case 3: Endpoint without service virtualization* on page 39.

These permutations are summarized in the following table:

Proxy or endpoint	Service virtualization	Example
Proxy	No	See *Use case 1: Proxy without service virtualization* on page 37.
Proxy	Yes	See *Use case 2: Proxy with service virtualization* on page 38.
Endpoint	No	See *Use case 3: Endpoint without service virtualization* on page 39.
Endpoint	Yes	See *Use case 4: Endpoint with service virtualization* on page 40.
Proxy or Endpoint	Yes	See *Use case 5: Simple redirect* on page 41.

Use case 1: Proxy without service virtualization

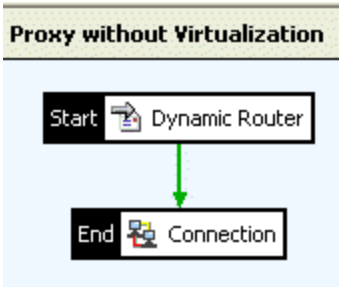
In this case, API Gateway is configured as an HTTP proxy for the client, and maintains the original path used by the client in the HTTP request. For example, if API Gateway is listening at `http://localhost:8080/`, and the web service is running at `http://localhost:5050/services/getQuote`, the request line of the client HTTP request appears as follows:

```
POST http://localhost:5050/services/getQuote HTTP/1.1
```

Because the client is configured to use the API Gateway instance running on `localhost` at port `8080` as its HTTP proxy, the client automatically sends all messages to the proxy. However, it includes the full URL of the ultimate destination of the message in the request line of the HTTP request.

When API Gateway receives the request, it extracts this URL from the request line and uses it to determine the destination of the message. In the above example, API Gateway routes the message on to `http://localhost:5050/services/getQuote`.

You can configure the following policy to route the message to the URL specified in the request line of the client request:



The following table explains the role of each filter in the policy. For more information on a specific filter, see the *API Gateway Policy Developer Filter Reference*.

Filter	Role in Policy
Dynamic Router	Extracts the URL of the destination web service from the request line of the incoming HTTP request. The **Dynamic Router** is normally used when API Gateway is perceived as a proxy by the client.
Connection	Establishes the connection to the destination web service, and sends the message over this connection. This connection can be mutually authenticated if necessary.

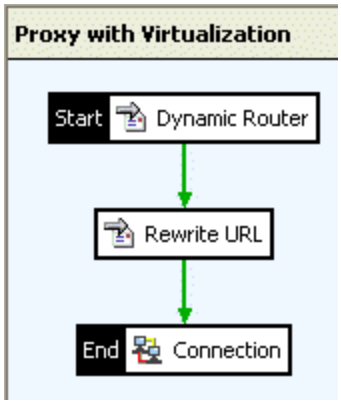
Use case 2: Proxy with service virtualization

In this case, API Gateway is also perceived as an HTTP proxy by the client. However, API Gateway exposes a *virtualized* view of the services that it protects. This is termed *service virtualization*.

To achieve this, API Gateway must provide a mapping between the path used by the client and the path under which the service is deployed. Assuming API Gateway is running at `http://localhost:8080/services`, and the web service is deployed at `http://localhost:5050/financialServices/quotes/getQuote`, the following example shows what the client might send up in the HTTP request line:

```
POST http://localhost:5050/services/getQuote HTTP/1.1
```

To achieve this, API Gateway must provide a mapping between what the client requests (`/services/getQuote`), and the address of the web service (`/financialServices/quotes/getQuote`). The **Rewrite URL** filter in the following policy fulfills this role:



The following table explains the roles of each filter in the policy. For more information on a specific filter, see the *API Gateway Policy Developer Filter Reference*.

Filter	Role in Policy
Dynamic Router	Extracts the URL of the destination web service from the request line of the incoming HTTP request. The **Dynamic Router** is normally used when API Gateway is perceived as a proxy by the client.
Rewrite URL	Specifies the mapping between the path requested by the client and the path under which the web service is deployed, therefore providing service virtualization.
Connection	Establishes the connection to the destination web service, and sends the message over this connection. This connection can be mutually authenticated if necessary.

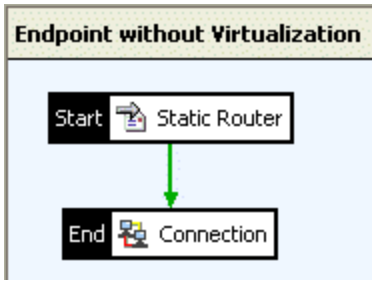
Use case 3: Endpoint without service virtualization

In this scenario, the client sees API Gateway as the endpoint to its connection, and API Gateway must be configured to route messages on to a specific destination. For example, assuming that API Gateway is running at `http://localhost:8080/services`, the request line of the client's HTTP request is received by API Gateway as follows:

```
POST /services HTTP/1.1
```

The request line above shows that no information about the scheme, host, or port of the destination web service is specified. Therefore, this information must be configured in API Gateway so that it knows where to route the message on to. The **Static Router** enables you to enter connection details for the destination web service.

Assuming that the web service is running at `http://localhost:5050/stockquote/getPrice`, the host, port, and scheme respectively are: `localhost`, `5050`, and `http`. You must explicitly configure this information in the **Static Router**. The following policy illustrates this scenario:



The following table explains the role of each filter in the policy. For more information on a specific filter, see the *API Gateway Policy Developer Filter Reference*.

Filter	Role in Policy
Static Router	Enables the user to explicitly specify the host, port, and scheme at which the web service is listening. This filter must be used when the client sees API Gateway as the endpoint to its connection (API Gateway is not acting as a proxy for the client).
Connection	Establishes the connection to the destination web service, and sends the message over this connection. This connection can be mutually authenticated if necessary.

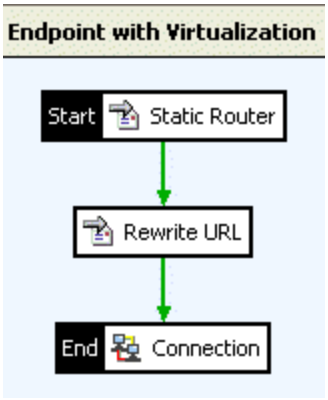
Use case 4: Endpoint with service virtualization

In this case, API Gateway acts as the endpoint to the client connection (and not as a proxy), and hides the deployment hierarchy of protected web services from clients (performs *service virtualization*).

In this scenario, the client sends messages directly to API Gateway. For example, assuming that API Gateway is running at `http://localhost:8080/services`, and the web service is running at `http://localhost:5050/stockquote/getPrice`, the request line of the client HTTP request is received by API Gateway as follows:

```
POST /services HTTP/1.1
```

You can then configure the **Static Router** filter to route the message on to port `8080` on `localhost` using the `http` scheme, while the **Rewrite URL** filter provides the mapping between the path requested by the client (`/services`) and the path under which the web service is deployed (`/stockquote/getPrice`). The following policy illustrates a sample policy that provides *service virtualization* when API Gateway is used as an endpoint:



The following table explains the role of each filter in the policy. For more information on a specific filter, see the *API Gateway Policy Developer Filter Reference*.

Filter	Role in Policy
Static Router	Enables you to explicitly specify the host, port, and scheme at which the web service is listening. This filter can be used when the client sees the API Gateway as the endpoint to its connection (not as a proxy for the client).
Rewrite URL	Provides the mapping between the path requested by the client and the path under which the web service is deployed.
Connection	Establishes the connection to the destination web service, and sends the message over this connection. This connection can be mutually authenticated if necessary.

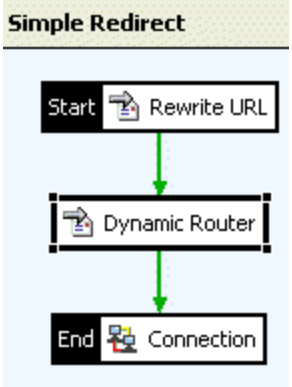
Note Alternatively, instead of using the **Static Router**, **Rewrite URL**, and **Connection** filters, you can use the **Connect to URL** filter, which is equivalent to using these three filters combined. You can configure the **Connect to URL** filter to send messages to a web service simply by specifying the destination URL. For more details, see "Connect to URL" in the *API Gateway Policy Developer Filter Reference*.

Use case 5: Simple redirect

In some cases, API Gateway must route the incoming message to an entirely different URL. You can use the **Rewrite URL** filter for this purpose, in addition to rewriting the path on which the request is received (as described in *Use case 2: Proxy with service virtualization* on page 38 and *Use case 4: Endpoint with service virtualization* on page 40).

Note The full URL of the destination web service should be specified in the **Rewrite URL** filter.

The following policy illustrates the use of the **Redirect URL** filter to route messages to a fully qualified URL:



The following table describes the role of each filter in the policy. For more information on a specific filter, see the *API Gateway Policy Developer Filter Reference*.

Filter	Role in Policy
Rewrite URL	Used to specify the fully qualified URL of the destination web service.
Dynamic Router	In this case, the **Dynamic Router** filter is used to parse the URL specified in the **Rewrite URL** filter into its constituent parts. The HTTP scheme, port, and host of the web service are extracted and set to the internal message object for use by the **Connection** filter.
Connection	Establishes the connection to the destination web service, and sends the message over this connection. This connection can be mutually authenticated if necessary.

Use case 6: Routing on to an HTTP proxy

This is a more advanced case where API Gateway is configured to route on through an HTTP proxy to the back-end web service sitting behind the proxy. When API Gateway is configured to route through a proxy, it connects directly to the proxy, and sends a request including the full URL of the target web service in the HTTP request URI. When the HTTP proxy receives this request, it uses the URL in the request line to determine where to route the message to. The following example shows the request line of a request made through a proxy:

```
POST http://localhost:8080/services/getQuote HTTP/1.1
```

The following filters are required to configure API Gateway to route through an HTTP proxy. For more information on a specific filter, see the *API Gateway Policy Developer Filter Reference*.

Filter	Role in Policy
Static Router	You must explicitly specify the host, port, and scheme of the HTTP proxy.
Rewrite URL	Enter the full URL of the web service (for example, `http://HOST:8080/myServices`). Because you are routing through a proxy, the full URL is sent in the request line of the HTTP request.
Connection	In this case, the **Connection** filter connects to the HTTP proxy, which in turn routes the message on to the destination server named in the request URI. The **Send via Proxy** option must be enabled in the **Connection** filter to facilitate this.

Note There are differences between how the filters are configured to route on through a proxy and the scenario described in *Use case 4: Endpoint with service virtualization* on page 40 where no proxy is involved:

- **Static Router**:
  When API Gateway routes on to an HTTP proxy, the **Static Router** filter is configured with the details of the HTTP proxy. Otherwise, the **Static Router** filter is given the details of the web service endpoint directly.
- **Rewrite URL**:
  The full URL of the web service endpoint must be specified in this filter when API Gateway routes through a proxy. The full URL is then included in the request line of the HTTP request to the proxy. In cases where no proxy is involved, the **Rewrite URL** filter is only necessary when the back-end web services are virtualized. In this case, API Gateway must send the request to a different URI than that requested by the client.
- **Connection**:
  When routing through a proxy, the **Send via Proxy** option must be enabled in the **Connection** filter. This is not necessary when no proxy sits between API Gateway and the back-end web service.

Summary

The following are the key concepts to consider when configuring API Gateway to connect to external web services:

- The **Connection** or **Connect to URL** filter *must* always be used because it establishes the connection to the web service.
- *Service virtualization* can be achieved using the **Rewrite URL** or **Connect to URL** filter.
- If the client is configured to use API Gateway as a proxy, API Gateway can use the **Dynamic Router** filter to extract the URL from the request line of the HTTP request. It can then route the message on to this URL.

- If the client sees API Gateway as the endpoint of the connection (not as a proxy), the **Static Router** filter can be used to explicitly configure the host, port, and scheme of the destination web service. Alternatively, you can use the **Connect to URL** filter to specify a URL.

Configure the sample policies

Overview

This topic introduces and explains how to set up the example policies available in the `samples` directory of your API Gateway installation. These include the following:

- **Conversion**: exposes a SOAP service over REST.
- **Security**:
  - Verifies the digital signature on the request and creates a signature on the response.
  - Decrypts the request and encrypts part of the response.
- **Throttling**: limits the number of calls for a service.
- **Virtualized Service**: combines threat protection, content-based routing (target a server according to request contents), and message transformation.

Tip If you are new to the API Gateway, you should first read the following to get familiar with the main concepts and basic steps:

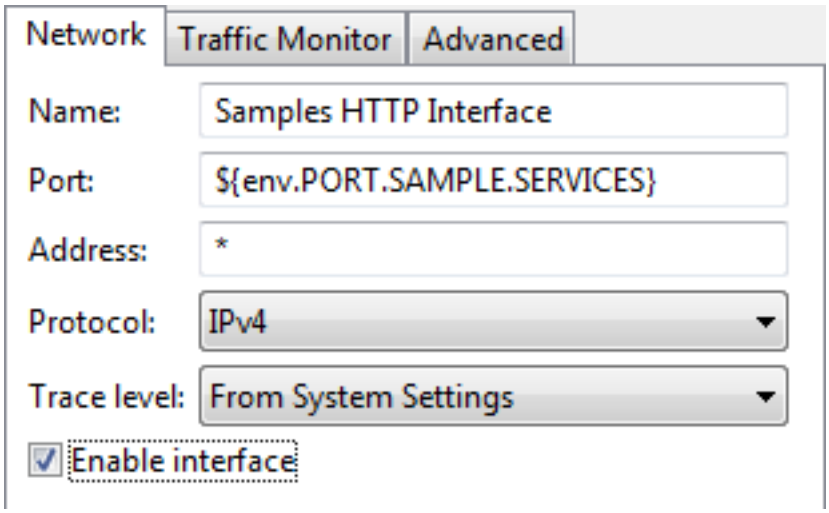
- *API Gateway Concepts Guide*
- *Policy development with Policy Studio* on page 23
- *Start the API Gateway tools* on page 30

This guide assumes that you have already installed and started the API Gateway and Policy Studio. The API Tester client GUI testing tool is optional.

Enable the sample services interface

The HTTP interface for the sample policy services is disabled by default. To enable this interface in Policy Studio, perform the following steps:

1. In the navigation tree, select **Environment Configuration** > **Listeners** > **API Gateway** > **Sample Services** > **Ports**.
2. In the **Interfaces** pane on the right, select ***:${env.PORT_SAMPLE_SERVICES}**.
3. Right-click, and select **Edit** to display the Configure HTTP Interface dialog.
4. Select the **Enable interface** setting.
5. Click **OK**.



Alternatively, you can also enable this HTTP interface using the web-based API Gateway Manager tool running on `http://HOST:8090`, where `HOST` is the machine on which the Node Manager is running.

1. Click the **Settings** button in the API Gateway Manager toolbar.
2. Select the HTTP interface node under **Sample Services** on the left.
3. Select the **Interface Enabled** setting on the right.
4. Click the **Apply** button.

Note Settings made in the web-based API Gateway Manager tool are dynamic settings only, which are not persisted.

Configure a different sample services interface

All sample policy services are defined in an HTTP services group named `Sample Services`. This group uses an HTTP interface running on the port specified in the `${env.PORT_SAMPLE_SERVICES}` environment variable. This external environment variable is set to `8081` by default. To use a different port, you must configure this variable in the `INSTALL_DIR/conf/envSettings.props` file. For example, you could add the following entry:

```
env.PORT.SAMPLE.SERVICES=8082
```

For more details on setting external environment variables for API Gateway instances, see the *API Gateway DevOps Deployment Guide*.

StockQuote demo service

All sample policies use a demo service named `StockQuote`, which is implemented using a set of policies. This service exposes two operations:

- **getPrice**: the policy for this operation uses a sample script to randomly calculate a quote value. Each call to `getPrice()` returns a different value.
- **update**: returns an `Accepted` HTTP code (202).

The `StockQuote` service is exposed on the following relative paths:

- `/stockquote/instance1`
- `/stockquote/instance2`

These relative paths are used in the virtualized service sample for content-based routing.

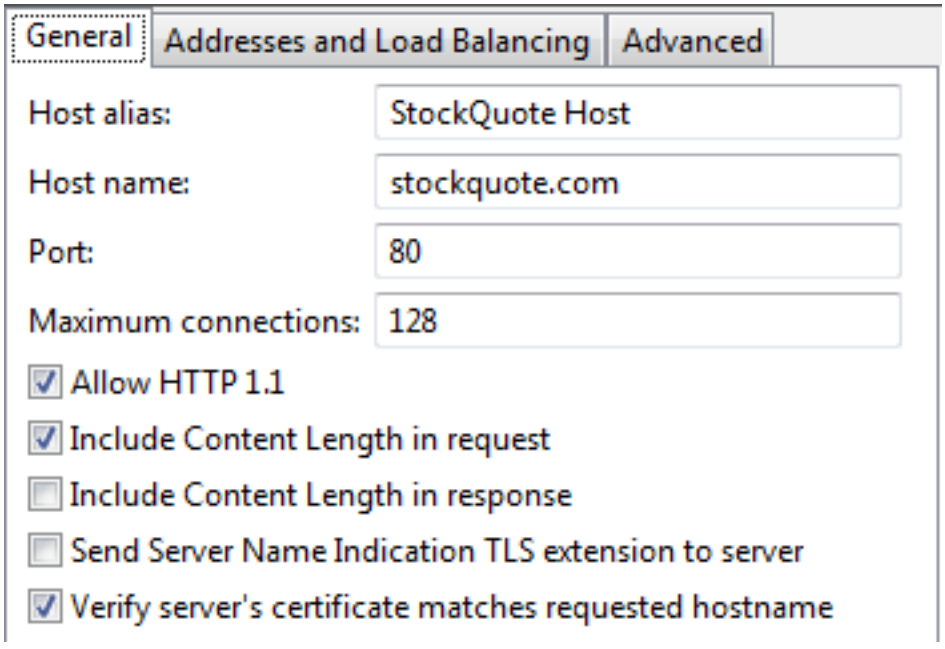
A **Connect to URL** filter with the following URL is used to invoke the `StockQuote` service from each of the sample policies:

```
http://stockquote.com/stockquote/instance1
```

The first part of this URL uses a *remote host* definition of `stockquote.com`. Remote hosts are logical names that decouple the host name in a URL from the server (or group of servers) that handles the request.

Remote host settings

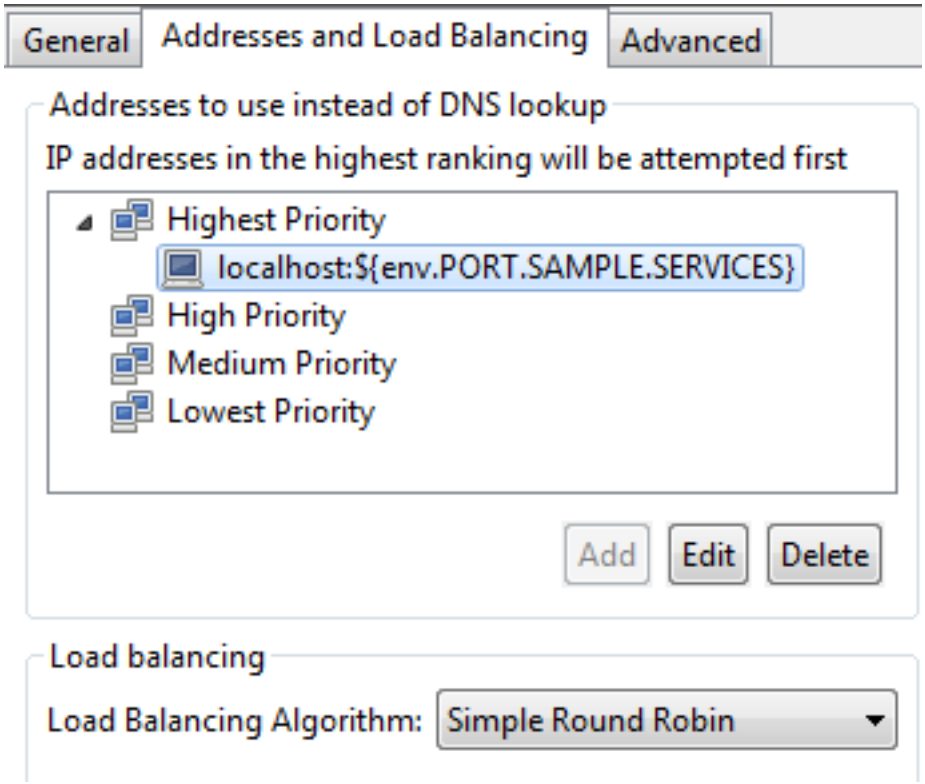
In Policy Studio, the remote host configuration is displayed under the API Gateway instance name (`API Gateway`) in the navigation tree, and is named `stockquote.com:80`. To view the remote host configuration, select **Environment Configuration** > **Listeners** > **StockQuote Host** and click **Edit**:



On the **General** tab, the remote host is set to:

- Use HTTP 1.1.
- Use port `80` by default.
- Include the `ContentLength` header in the request to the back-end server.
- In case of an SSL connection, verify the Distinguished Name (DN) in the certificate presented by the server against the server's host name.

On the **Addresses and Load Balancing** tab, the remote host is set to send requests to `localhost:${env.PORT_SAMPLE_SERVICES}`, which resolves to `localhost:8081` by default. You could also specify several servers in the **Addresses to use instead of DNS lookup** list, and the API Gateway would load balance the requests across servers in the same group using the specified algorithm.



For more details on these settings, see *Configure remote host settings* on page 268.

Conversion sample policy

Overview

The conversion sample policy takes a REST-style request and converts it into a SOAP call. This topic describes the **REST to SOAP** policy, and explains how to run this sample.

REST to SOAP policy

The **REST to SOAP** policy is as follows:

The **REST to SOAP** policy performs the following tasks:

1. Extracts the information from the request (a message attribute is created for each query string and/or HTTP header).
2. Creates a SOAP message using the **Set Message** filter.
3. Sends the request to the `StockQuote` demo service.
4. Extracts the value of the stock from the response using XPath.
5. Creates a plain text response.
6. Sets the HTTP status code to 200.

Run the conversion sample

You can call the sample service using the send request (`sr`) command or the API Tester GUI:

sr command

Enter the following command:

```
sr http://HOSTNAME:8081/rest2soap?symbol=ABC
```

For more details, see *Stress test with send request (sr)* on page 59.

API Tester

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

```
http://HOSTNAME:8081/rest2soap?symbol=ABC
```

2. Select `GET` as the verb.
3. Click the **Run** button.

For more details, see the topic on *Send a request with API Tester* on page 63.

Security sample policies

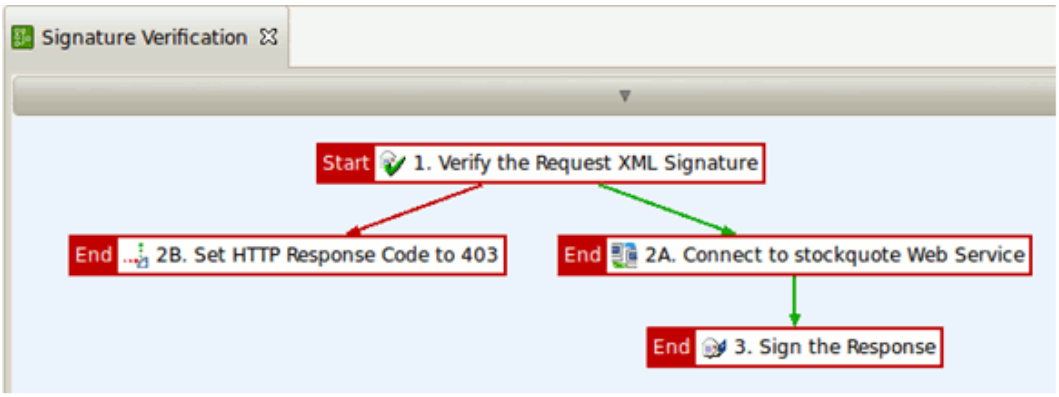
Overview

The security sample policies demonstrate digital signature verification and cryptographic operations (encryption and decryption). This topic describes the sample policies, and explains how to run these samples.

Signature verification

The **Signature Verification** sample policy sends a digitally signed version of the `StockQuote` request to the API Gateway. The message carries the signature into the web service header. A sample certificate/key pair (`Samples Test Certificate`) is used to sign the message and verify the signature. Signature verification is used for authentication purposes, and therefore an HTTP 403 error code is returned if a problem occurs.

The **Signature Verification** policy is as follows:



The **Signature Verification** policy performs the following tasks:

1. The signature contained in the request is verified. The signature must be located in a WS-Security block.
2. If the verification is successful, the `StockQuote` demo service is invoked.
3. The response body is signed and returned to the client.
4. If the verification fails, an HTTP 403 error code is returned to the client.

Run the signature verification sample

You can call the sample service using the send request (`sr`) command or the API Tester GUI:

sr command

Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/Security/SignatureVerification/Request.xml
http://localhost:8081/signatureverification
```

For more details, see the topic on *Stress test with send request (sr)* on page 59.

API Tester

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/signatureverification
   ```

2. Select `POST` as the **Verb**.
3. Click the **Close** button.

4. Select **File** > **Load**, and browse to the following file as input for the request:

```
INSTALL_DIR/samples/SamplePolicies/Security/SignatureVerification/Request.xml
```

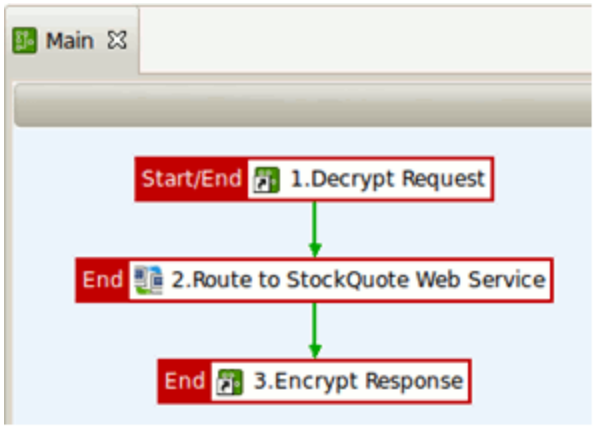
5. Click the Send Request button.

For more details, see the topic on *Send a request with API Tester* on page 63.

Encryption and decryption

This sample uses XML decryption on the request and applies encryption on the response. The sample policy includes a **Main** policy, which chains together the calls that decrypt the request, the invocation of the back-end service, and the encryption of the response.

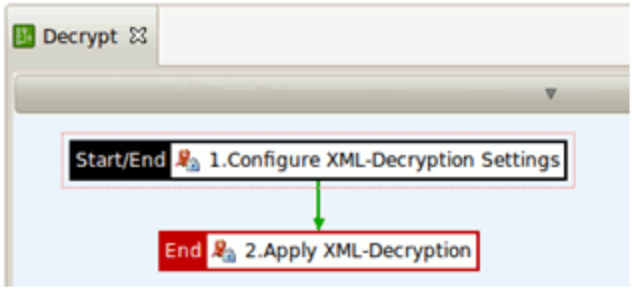
The **Main** policy is as follows:



The **Main** policy performs the following tasks:

1. **Decrypt Request** is a policy shortcut, which invokes another policy that takes the inbound request and decrypts it.
2. The decrypted request is routed to the back-end service.
3. The **Encrypt Response** policy shortcut invokes a policy that encrypts the response from the back-end service.

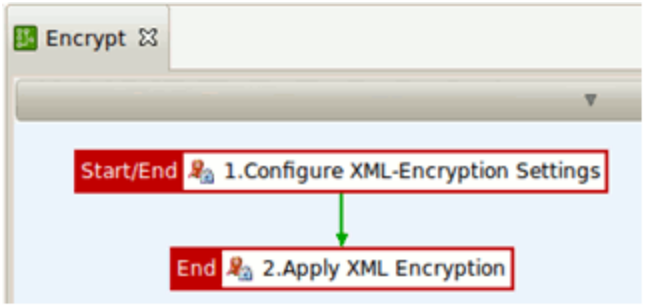
The **Decrypt** policy is as follows:



The **Decrypt** policy performs the following tasks:

1. The decryption settings are defined: what to decrypt and which key to use.
2. The XML decryption is executed based on the defined settings.

The **Encrypt** policy is as follows:



The **Encrypt** policy performs the following tasks:

1. The encryption settings are defined: what to encrypt, which symmetric key to use, which certificate to use, and how to encrypt (algorithm and where to place the encryption information).
2. The XML encryption is executed based on the defined settings.

Run the encryption and decryption sample

You can call the sample service using the send request (`sr`) command or the API Tester GUI:

sr command

Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/Security/Encryption/Request.xml
http://HOSTNAME:8081/encryption
```

For more details, see the topic on *Stress test with send request (sr)* on page 59.

API Tester

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/encryption
   ```

2. Select `POST` as the **Verb**.
3. Click the **Close** button.

4. Select **File** > **Load**, and browse to the following file as input for the request:

```
INSTALL_DIR/samples/SamplePolicies/Security/Encryption/Request.xml
```

5. Click the Send Request button.

For more details, see the topic on *Send a request with API Tester* on page 63.

Throttling sample policy

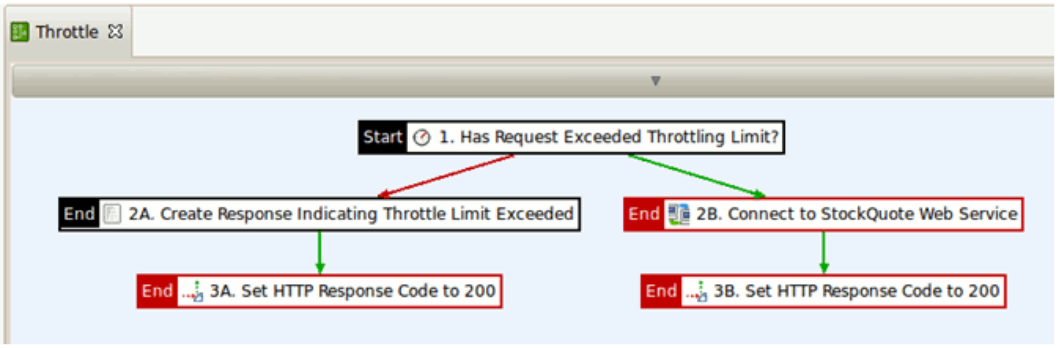
Overview

The throttling sample policy is used to limit the number of calls for a service. This topic describes the **Throttle** policy, and explains how to run this sample.

Throttling refers to restricting incoming connections and the number of messages to be processed. It can be applied to XML, SOAP, REST, or any payload, request, or protocol. Traffic can be regulated for a single API Gateway or for a cluster of API Gateways. You can apply traffic restrictions rules for a service, an operation, or even time of day. For example, these restrictions can be applied depending on the service name, user identity, IP address, content from the payload, protocol headers, and so on.

Throttling policy

The **Throttle** policy is as follows:



The **Throttle** policy performs the following tasks:

1. The first filter checks whether the limit has been reached. The limit is set to 3 requests per 15 sec. The caller's IP address is used to track the consumer ID. The counter is kept in a local cache.
2. If the limit has been reached, an error message is created, and the response status code is set to 500.

3. If the authorized limit has not been reached, the back-end service is invoked, and the HTTP status code is set to 200.

Run the throttling sample

You can call the sample service using the send request (`sr`) command or the API Tester GUI:

sr command

Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/Throttling/Request.xml
http://HOSTNAME:8081/throttle
```

For more details, see the topic on *Stress test with send request (sr)* on page 59.

API Tester

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/throttle
   ```

2. Select `POST` as the **Verb**.
3. Click the **Close** button.
4. Select **File** > **Load**, and browse to the following file as input for the request:

   ```
   INSTALL_DIR/samples/SamplePolicies/Throttling/Request.xml
   ```

5. Click the Send Request button.

For more details, see the topic on *Send a request with API Tester* on page 63.

Virtualized service sample policy

Overview

The virtualized service sample policy is more advanced and combines the following features:

- Content filtering, XML complexity, and message size filters to block unwanted SOAP messages.
- Content filtering to block unwanted REST requests.

- Fault handling.
- Content-based routing.

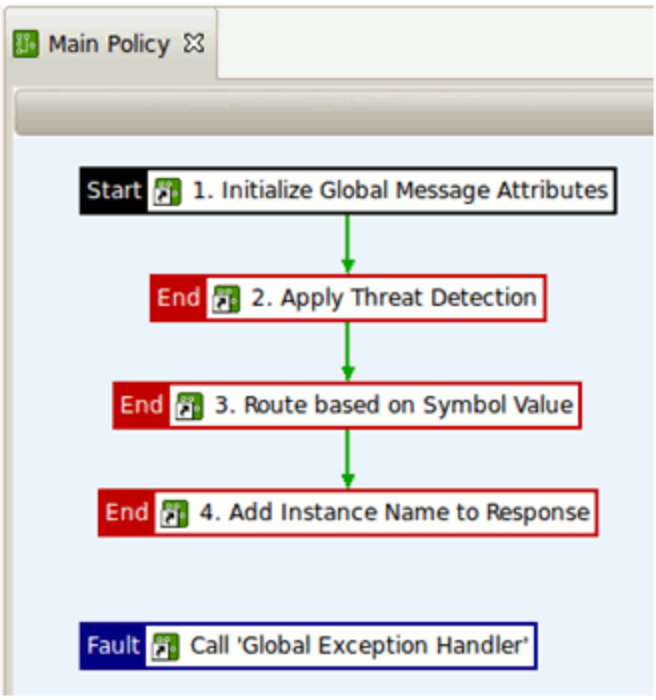
This topic describes the policies displayed in the **Sample Policies** > **Web Services** > **Virtualized StockQuote Service** policy container in Policy Studio, and explains how to run this sample.

Virtualized service policies

The **Virtualized StockQuote Service** sample policy container includes the following policies:

- Virtualized service main policy
- Threat protection policy
- Content-based routing policies
- Response transformation policy

The **Main Policy** is as follows:

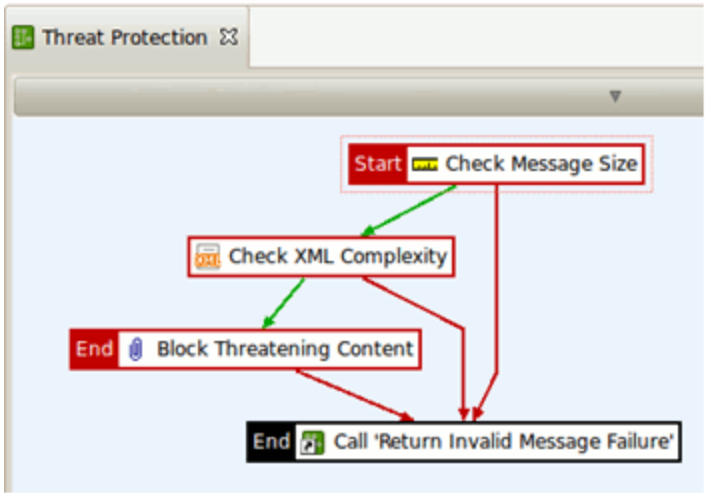


The **Main Policy** uses policy shortcuts to perform the following tasks:

1. The main fault handler relies on some variables to be initialized, which is performed as soon as the policy is entered.
2. The **Threat Detection** policy is applied to the incoming SOAP message and HTTP headers.
3. The symbol value is extracted from the incoming message, and used to decide whether the request should be sent to one server instance or another.
4. The name of the instance that served the request is added to the response.

5. In case of errors, a global fault handler is invoked. This is used to return a custom error message to the user.

The **Threat Protection** policy is as follows:



The **Threat Protection** policy performs the following tasks:

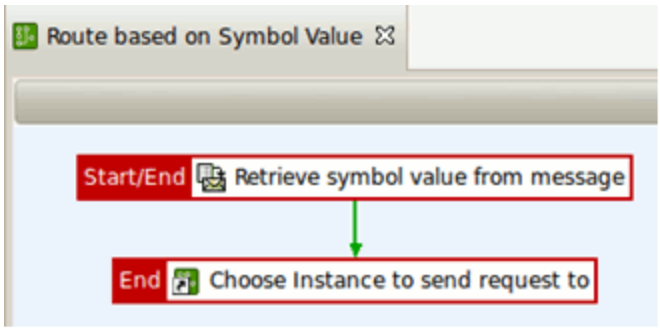
1. The incoming request size (including attachments) is checked to be less than 1500 KB.
2. The complexity of the XML is checked in terms of number of nodes, attributes per node, or number of child nodes per node.
3. XML and eventually HTTP headers are checked for threatening content such as SQL injection or XML processing instructions.
4. If any of these filters return an error, the corresponding error handler is called. The error handler is implemented as a policy that sets the value of the error code and message for this error, and then re-throws the exception so that the global fault handler catches it.

Content-based routing policies

The **Route Based on Symbol Value** policy extracts the contents of the symbol XML node and checks whether the first letter's value is between `A-L` or `K-Z`. Depending on the result, it routes the request to the first or second instance of the `StockQuote` server. These servers are simulated by the following relative path URIs defined in the API Gateway:

- `/stockquote/instance1`
- `/stockquote/instance2`

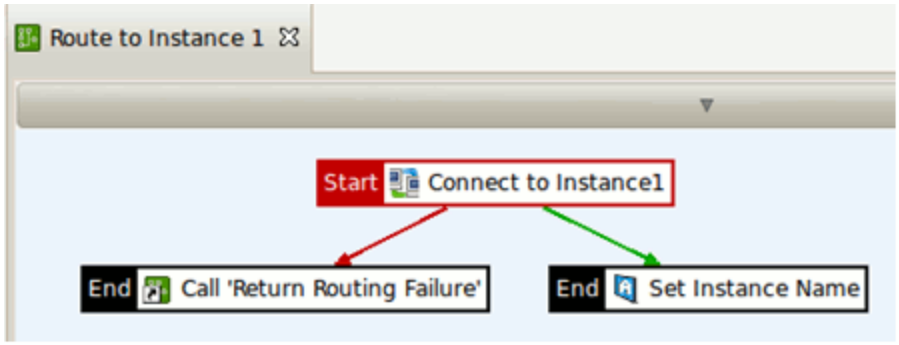
The **Route Based on Symbol Value** policy is as follows:



The **Route Based on Symbol Node** policy performs the following tasks:

1. The value of the symbol node is extracted from the request using XPath. The result is placed in a message attribute named `message.symbol.value`.
2. A **Switch on attribute value** filter is used to check the value of the message attribute (using a regular expression), and a different policy is called to send the request to `instance1` or `instance2`.

The **Route to Instance1** policy is as follows:



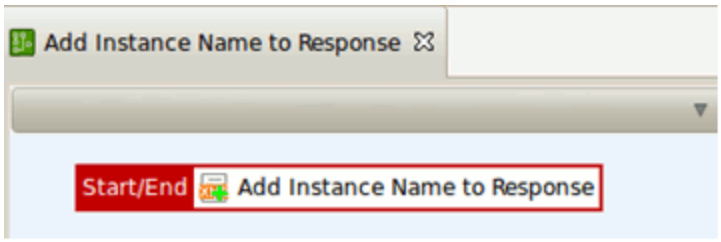
The **Route to Instance1** policy (called from the Switch filter) performs the following tasks:

1. Connects to the `instance1` URI.
2. If successful, the instance name (`instance1`) is placed in a message attribute (`stockquote.instance.name`). This is used later on to insert the instance name into the response.

The **Route to Instance2** policy performs the same tasks but using the `instance2` URI instead.

Response transformation policy

When the response is obtained from the back-end server, the **Add Instance Name to Response** policy changes it to insert the instance name into a new XML node (`instanceName`). The **Add Instance Name to Response** policy is as follows:



This policy adds the instance name (the value of the `stockquote.message.name` message attribute) to the response, using an **Add XML node** filter, as part of the `SOAPbody`. XPath is used to define where the new node must be added.

Run the virtualized service sample

You can call the sample service using the send request (`sr`) command or the API Tester GUI:

sr command

Enter the following command:

```
sr -f INSTALL_DIR/samples/SamplePolicies/VirtualizedService/Request.xml
http://HOSTNAME:8081/main/stockquote
```

For more details, see the topic on *Stress test with send request (sr)* on page 59.

API Tester

Perform the following steps:

1. Specify the following URL in the **Request Settings**:

   ```
   http://HOSTNAME:8081/main/stockquote
   ```

2. Select `POST` as the **Verb**.
3. Click the **Close** button.
4. Select **File** > **Load**, and browse to the following file as input for the request:

   ```
   INSTALL_DIR/samples/SamplePolicies/VirtualizedService/Request.xml
   ```

5. Click the Send Request button.

For more details, see the topic on *Send a request with API Tester* on page 63.

Stress test with send request (sr)

The API Gateway provides a command-line tool for stress testing named send request (`sr`). The sr tool is available in the following directory of your API Gateway installation:

```
INSTALL_DIR/posix/lib
```

The `sr` tool is also available from the root directory of the API Tester installation.

Note On Linux, the `LD_LIBRARY_PATH` environment variable must be set to the directory from which you are running the `sr` tool, and you must use the `vrun sr` command. For example:

```
vrun sr http://testhost:8080/stockquote
```

Basic sr command examples

The following are some basic examples of using the `sr` command:

HTTP GET:

```
sr http://testhost:8080/stockquote
```

POST file contents (content-type inferred from file extension):

```
sr -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Send XML file with SOAP Action 10 times:

```
sr -c 10 -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Send XML file with SOAP Action 10 times in 3 parallel clients:

```
sr -c 10 -p 3 -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Send the same request quietly:

```
sr -c 10 -p 3 -qq -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

Run test for 10 seconds:

```
sr -d 10 -qq -f StockQuoteRequest.xml http://testhost:8080/stockquote
```

POST file contents with SOAP Action:

```
sr -f StockQuoteRequest.xml -A SOAPAction:getPrice http://testhost:8080/stockquote
```

Advanced sr command examples

The following are some advanced examples of using the `sr` command:

Send form.xml to http://192.168.0.49:8080/healthcheck split at 171 character size, and trickle 200 millisecond delay between each send with a 200 Content-Length header:

```
sr -h 192.168.0.49 -s 8080 -u /healthcheck -b 171 -t 200 -f form.xml
-a "Content-Type:application/x-www-form-urlenprogramlistingd" -a "Content-
Length:200"
```

Send a multipart message to http://192.168.0.19:8080/test, 2 XML docs are attached to message:

```
sr -h 192.168.0.49 -s 8080 -u /test -{ -a Content-Type:text/xml -f soap.txt
-a Content-Type:text/xml -f attachment.xml -a Content-Type:text/xml -} -A c-
timestamp:1234
```

Send only headers using a GET over one-way SSL running 10 parallel threads for 86400 seconds (1 day) using super quiet mode:

```
sr -h 192.168.0.54 -C -s 8443 -u /nextgen -f test_req.xml -a givenName:SHViZXJ0
-a sn:RmFuc3dvcnRo -v GET -p10 -d86400 -qq
```

Send query string over mutual SSL presenting client certificate and key doing a GET running 10 parallel threads for 86400 seconds (1 day) using super quiet mode:

```
sr -h 192.168.0.54 -C -s 8443 -X client.pem -K client.key
-u "https://localhost:8443/idp?TargetResource=http://axway.test.com" -f test_req.xml
-v GET -p10 -d86400 -qq
```

Send zip file in users home directory to testhost on port 8080 with /zip URI, save the resulting response content into the result.zip file, and do this silently:

```
sr -f ~/test.zip -h testhost -s 8080 -u /zip -a Content-Type:application/zip
-J result.zip -qq
```

sr command arguments

The main arguments to the `sr` command include the following:

Argument	Description
`--help`	List all arguments
`-a attribute:value`	Set the HTTP request header (for example, `-a Content-Type:text/xml`)
`-c [request-count]`	Number of requests to send per process
`-d [seconds]`	Duration to run test for
`-f [content-filename]`	File to send as the request
`-h [host]`	Name of destination host
`-i [filename]`	Destination of statistics data
`-l [file]`	Destination of diagnostic logging
`-m`	Recycle SSL sessions (use multiple times)
`-n`	Enable nagle algorithm for transmission
`-o [output]`	Output statistics information every [milliseconds] (only with `-d`)
`-p [connections]`	Number of parallel client connections (threads) to simulate
`-q , -qq, -qqq`	Quiet modes (quiet, very quiet, very very quiet)
`-r`	Do not send HTTP Request line
`-s [service]`	Port or service name of destination (default is `8080`)
`-t [milliseconds]`	Trickle: delay between sending each character

Argument	Description
`-u [uri]`	Target URI to place in request
`-v [verb]`	Set the HTTP verb to use in the request (default is `POST`)
`-w [milliseconds]`	Wait for [milliseconds] between each request
`-x [chunksize]`	Chunk-encode output
`-y [cipherlist]`	SSL ciphers to use (see OpenSSL manpage ciphers(1))
`-z`	Randomize chunk sizes up to limit set by `-x`
`-A attribute:value`	Set the HTTP request header (for example, `-a Content-Type:text/xml`) in the outermost attachment
`-B`	Buckets for response-time samples
`-C`	enCrypt (use SSL protocol)
`-I [filename]`	File for Input (received) data (`- = stdout`)
`-K`	RSA Private Key
`-L`	Line-buffer `stdout` and `stderr`
`-M`	Multiplier for response-time samples
`-N`	origiN for response-time samples
`-O [filename]`	File for Output (sent) data (`- = stdout`)
`-S [part-id]`	Start-part for multipart message
`-U [count]`	reUse each connection for `count` requests
`-V [version]`	Sets the HTTP version (`1.0`, `1.1`)
`-X`	X.509 client certificate
`-Y [cipherlist]`	Show expanded form of `[cipherlist]`
`[-{/-}`	Create multipart body (nestable: use `-f` for leaves)

Further information

For a listing of all arguments, enter `sr --help`.

Send a request with API Tester

Overview

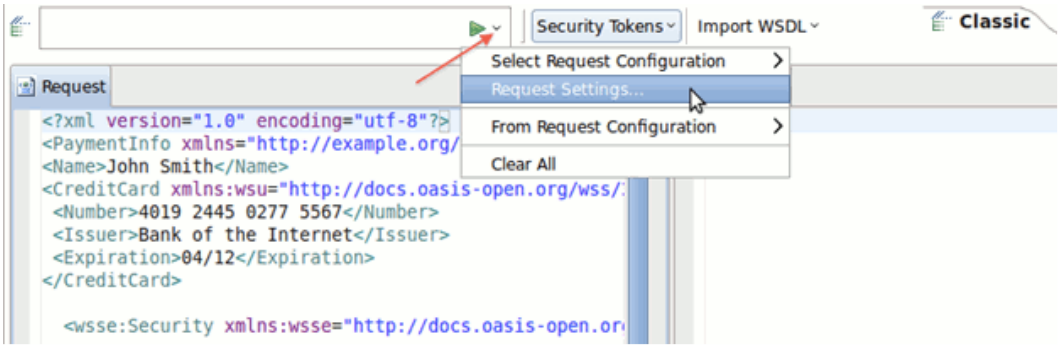
This topic describes how to create and send a request in the API Tester test client GUI. You can start API Tester using the `apitester` command from the installation directory.

Note API Tester is deprecated and will be removed in a future release.

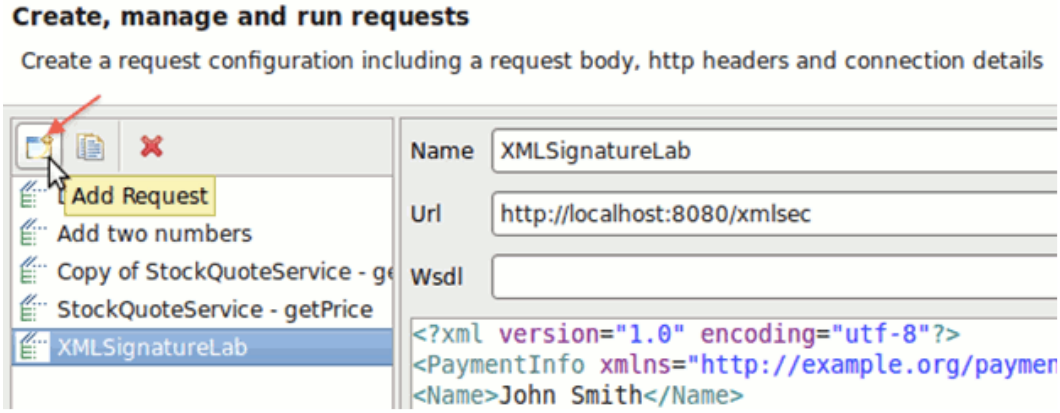
Create a request in API Tester

To create a request, perform the following steps:

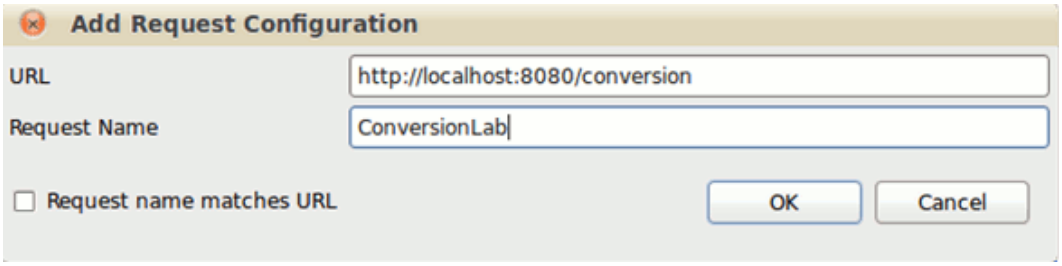
1. Click the down arrow button beside the Send Request button in the toolbar, and select **Request Settings**:



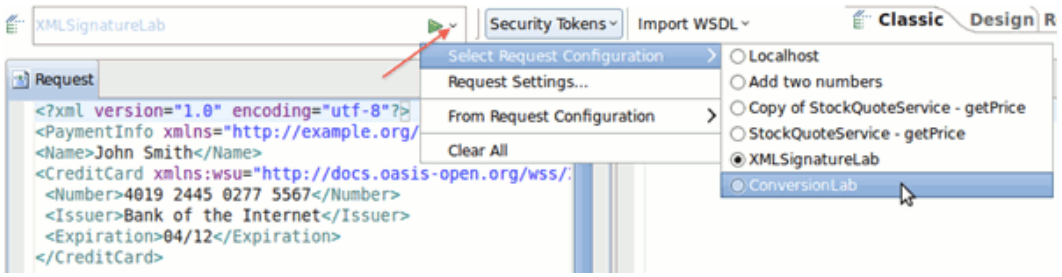
2. In the Request Settings dialog, click the **Add Request** button in the toolbar:



3. Enter the details for the request to execute in the Add Request Configuration dialog. For example, enter `http://localhost:8080/conversion` in the **URL** field. If the **Request name matches URL** setting is not selected, you can supply a custom **Request Name** for this request. Click **OK** to save the request configuration.



4. Click the down arrow button beside the Send Request button in the toolbar, and select the request that you created in the **Select Request Configuration** menu:



5. In the main menu, select **File** >**Load Request**, and browse to the file to use as input for this request. For example, you can select the following file for the Virtualized Service sample:

```
INSTALL_DIR/samples/SamplePolicies/VirtualizedService/Request.xml
```

6. Click the Send Request button in the toolbar to send the request.

Further information

For more details on using the API Tester client GUI tool, see the API Tester online help.

Manage policies 2

This section describes how to manage API Gateway policies.

Configure policies manually

Overview

This topic describes how to use Policy Studio to configure an API Gateway policy manually. It also applies to cases where a web service definition is not available in a Web Services Description Language (WSDL) file, meaning that the policy used to protect a web service must be configured manually.

However, the recommended way to configure a policy to protect a web service is to import the WSDL file for that service. If WS-Policy information is contained in the WSDL file, the policy assertions can also be used to produce a complex policy with minimum effort for administrators. If your web service has WSDL-based definitions, see *Configure policies from WSDL files* on page 77.

Configuration

The following steps outline how to manually create a policy to protect a web service and then test it.

Step 1: Create the policy

To create a policy manually, complete the following steps:

1. Right-click the **Policies** node in the Policy Studio tree, and select the **Add Policy** menu option.
2. Enter a suitable name (for example `TestPolicy`) for the new policy in the **Name** field, and click the **OK** button. The new policy is now visible in the tree.
3. Click the new policy in the tree to start configuring the filters for the policy. You can easily configure the policy by dragging the required filters from the filter palette on the right of Policy Studio, and dropping them on to the policy canvas.

Most policies attempt to check characteristics of the message, such as message size and format, and attempt to authenticate or authorize the sender of the message. When the message successfully passes all configured filters, it is usually routed on to the protected web service.

For demonstration purposes, this example creates a simple policy consisting of two filters. The first filter checks the size of the message, and the second echoes the request message back to the client if it is below a certain size.

4. Expand the **Content Filtering** category of filters from the filter palette, and drag and drop the **Message Size** filter on to the canvas.
5. Enter `10` in the **At least** field and `1000` in the **At most** field to make sure that only messages between 10 bytes and 1000 bytes are reflected back to the client. Select all other defaults, and click the **Finish** button.
6. Right-click the newly added filter, and select the **Set as Start** menu option to indicate that this is the first filter to be executed in this policy. The icon for the filter changes to indicate that it is the start of the policy.
7. Open the **Utilities** category of filters, and drag the **Reflect** filter onto the canvas. Drop it on to the previously configured **Message Size** filter. Select the defaults for the **Reflect** filter, and click the **Finish** button.

   Because you dropped the **Reflect** filter on to the **Message Size** filter, both filters are automatically linked with a *success path*. This means that if the first filter runs successfully, the next filter on the success path is executed. To link in more filters, add the filters to the canvas, and click the **Success Path** button at the top of the palette. Click the first filter followed by the second filter in the success path to link both filters.

   You can also configure *failure paths* for filters in the same way. Failure paths are followed when the checks configured in the filter fail.

This completes the configuration of the simple policy.

Step 2: Create a new relative path

You must now create a **Relative Path** on the API Gateway instance, which maps incoming requests on a particular URI to the new policy. Complete the following steps:

1. In the Policy Studio tree, select **Environment Configuration** > **Listeners** > **API Gateway**.
2. Right-click the **Default Services** node, and select **Add Relative Path**.
3. On the **Configure Relative Path** dialog, enter a suitable URI (for example, `TestPolicy`) on which to receive requests that are to be processed by the new policy.
4. To map requests received on this URI to our new policy, select the `/TestPolicy` policy from the list of policies in the tree. Click the **OK** button when finished.

Step 3: Deploy to API Gateway

Before the new configuration changes can take effect, you must deploy them to API Gateway. You can do this by clicking the **Deploy** button on the right of the toolbar. Alternatively, press the **F6** key.

Step 4: Test the policy

You can use the tool of your choice (for example, Axway API Tester) to send SOAP requests to the new policy. You should send requests of different sizes to the following URL, assuming a default installation of API Gateway running on the local machine:

```
http://localhost:8080/TestPolicy
```

Request messages that fall between the configured size are reflected to the client. Those that fall outside of the configured size are blocked, and a SOAP Fault is returned to the client.

Step 5: Next steps

Try running more complicated checks on request messages by adding new filters to the `TestPolicy`. Try also adding failure paths to the original **Message Size** filter to handle messages that fall outside of the 10-1000 byte range.

Use the **Help** button on each filter window to find out more about the configuration fields that are available on each window.

Configure global policies

Overview

Global policies enable you to label policies with specific roles in the API Gateway configuration. For example, you can label a specific policy such as **XML Threat Policy** as a **Global Request policy**. This policy can be executed globally on the request path for all messages passing through API Gateway.

Using a global policy in this way enables you to use the same policy on all requests, and for multiple services. It also means that you can change the labeled global policy to a different policy without needing to rewire any existing policies.

For example, using a **Policy Shortcut Chain** filter in a policy enables you to delegate to one or more policies to perform specific tasks, before continuing execution of the remaining filters in the

current policy. Using this approach to encapsulate specific functionality in a policy facilitates modularity and reusability when designing API Gateway policies. This enables you to build up a policy library of reusable routines over time.

Each shortcut in a **Policy Shortcut Chain** points to a specific policy, which is called at each point in the execution chain. However, consider a policy whose role is to be called first in all message handling contexts before any context-specific policies are run, and call this the run-first role. To realize this, you must create a **Policy Shortcut Chain** with a link to the run-first policy as its first entry, the context-specific policy as its second link, and so on.

One of the shortcomings of this approach is that if you have set up a large number of **Policy Shortcut Chain** filters, each calling the run-first policy, and you need to change the run-first policy globally, you must update each **Policy Shortcut Chain** filter individually to point to the newly designated run-first policy. Similarly, if you wish to ignore the run-first Policy globally, you must remove the first entry in each filter.

Global policies enable you to label a specific policy in terms of its role. You can delegate to the policy using its label instead of a specific link to the policy. This indirection using a label makes it very easy to globally change which policy is delegated to, merely by moving the label from one policy to another. Each filter that refers to the policy using its label now resolves the label to the new policy without needing to change the filter configuration. Similarly, if the label is not applied to a specific policy, nothing is executed for this link.

Global policy roles

The following global policy roles have a reserved label and a specific meaning in the API Gateway policy framework:

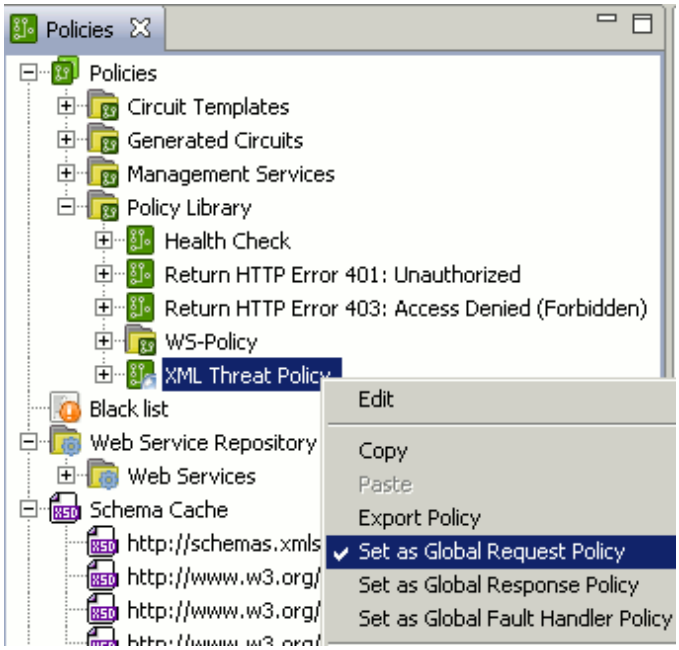
Role	Label	Description
Global Request Policy	`system.policy.request`	Executed globally for all messages passing through the API Gateway on the request path.
Global Response Policy	`system.policy.response`	Executed globally for all messages passing through the API Gateway on the response path.
Global Fault Handler Policy	`system.policy.faulthandler`	If any policy aborts during execution, or a top-level policy fails and has not specified a Fault Handler filter, this policy is executed instead of the internal **SOAP Fault** filter.

You can select specific policies with these roles under the **Policies** node in the Policy Studio tree. You can then create links to these roles when creating a **Policy Shortcut Chain**. These steps are explained in the next sections.

Select a global policy

To select a global policy, right-click a policy under the **Policies** node, and select one or more global policies (for example, **Set as Global Request Policy**, **Set as Global Response Policy**, or **Set as Global Fault Handler Policy**). These policies are executed globally for all messages passing through API Gateway.

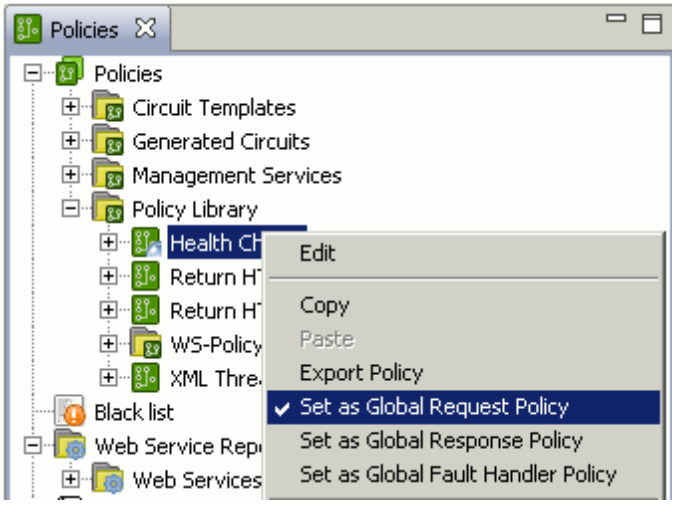
The following example shows the **XML Threat Policy** set as the **Global Request Policy**. The policy node labeled for the specific role is displayed with a globe icon:



When you have selected the policy for a specific role, you can then reference the labeled policy in a **Policy Shortcut Chain** filter, or at the service level in a relative path or web service resolver. Referencing a labeled policy is different from referencing a specific policy directly. Referencing a policy directly involves selecting a specific policy to execute in the chain. Referencing a labeled policy means selecting a filter by its label only.

The main advantage of this approach is that you can configure a policy to run in a policy shortcut chain in a specific role, and then select a different policy as the global policy for that role. All references to the global policy label in the various shortcut chain filters are changed to use the newly selected policy, without requiring you to modify each policy shortcut chain filter individually to explicitly point to a different policy.

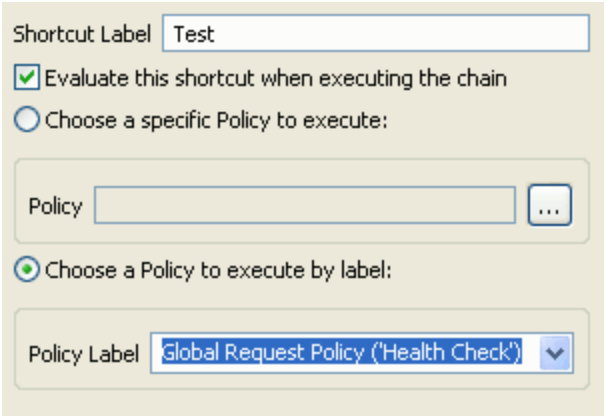
Selecting another policy in a global role deselects the previously selected policy. The following example shows the **Health Check** set in the global role, and the **XML Threat Policy** is no longer selected:



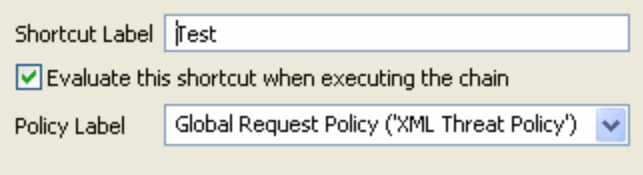
Note You cannot select a policy for a specific role if, in doing so, you create a loop in the policies. For example, if a **Policy Shortcut Chain** filter has a reference to a labeled policy, and the filter's parent policy is marked as the labeled policy, the filter would call back to itself in a loop. This error is caught, and a trace line is output to Policy Studio **Console** view.

Configure global policies in a policy shortcut chain

When adding a policy shortcut in a **Policy Shortcut Chain** filter, you can select to call a labeled policy instead of selecting a specific policy. The following example from the **Add a new Shortcut to a Policy** dialog shows adding a shortcut to the **Global Request Policy (Health Check)** policy label:



Then if you select a different policy as the request policy in the Policy Studio tree, when you subsequently view this shortcut in the chain filter, you see that the details for the shortcut have changed. The following example from the **Edit the Shortcut to the Policy** dialog shows the policy label changed to **Global Request Policy (XML Threat Policy)**.

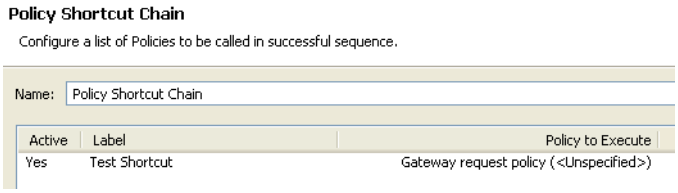


For more details on configuring these windows, see "Policy shortcut chain" in the *API Gateway Policy Developer Filter Reference*.

Note If you remove a label from a policy by deselecting it in the Policy Studio tree, any reference to that labeled policy is not called when evaluating the shortcut in the chain, irrespective of whether **Evaluate this shortcut when executing the chain** is selected (the **Active** status column in the table view).

This corresponds with the behavior for a specific policy in the chain. If a link to a policy is not set for a shortcut, the link is not evaluated.

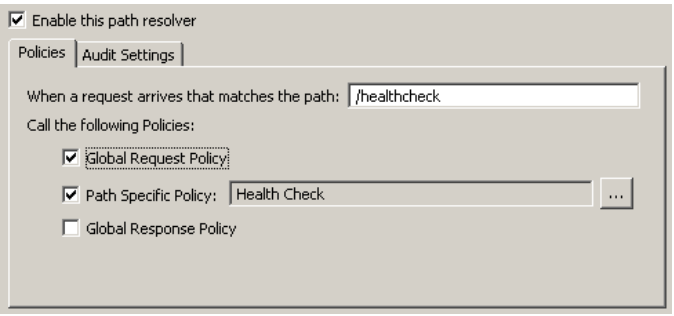
In this example, the table shows that the shortcut is configured to point to the labeled policy, but the label does not resolve to a policy (for example, it is unspecified because there is no policy in the specified role):



Configure global policies for a service

Under the **Environment Configuration** > **Listeners** node, you can also configure global policies at the service level to run on a specific relative path or web service resolver when messages are received by API Gateway. A relative path binds a policy to a specific relative path location (for example `/healthcheck`). A web service resolver maps messages destined for a specific web service to a **Service Handler** or **Web Service Filter**.

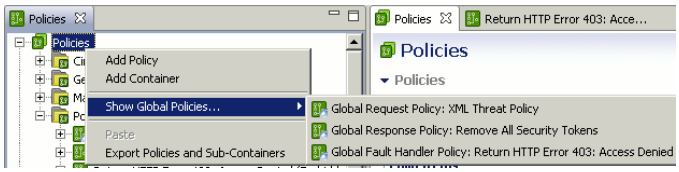
You can configure a global policy at the service level to run as part of a policy chain invoked when incoming messages are received by API Gateway. The following example shows the **Global Request Policy** configured to run first on the `/healthcheck` relative path:



For more details, see *Configure relative paths* on page 293.

Show global policies

To view the currently configured global policies, right-click the **Policies** root tree node, and select **Show Global Policies**. This displays all currently configured global policies in the context menu, for example:



Note If there are no global policies configured, the **Show Global Policies** menu item is not available.

Configure policy assemblies

Overview

In certain cases, you might need to convert a policy into a modular reusable piece of functionality that can be called from other policies.

For example, you have a complex policy that creates a WS-Security `UsernameToken`, inserts it into the message, and subsequently creates an XML Signature over the token and SOAP body. Depending on the message recipient, the content might need to be signed using slightly different settings. One service might require a `<sp:Basic128/>` algorithm suite, while another might require `<sp:Basic256/>`).

Similarly, subtle differences in security requirements might require the token and signature to be generated differently. For example:

- Use a basic or digest password for the `UsernameToken`
- Insert a `<dsig:CarriedKeyName>` into the XML-Signature
- Create an enveloped or enveloping signature
- Include a `<wsse:BinarySecurityToken>`
- Use one signing key over another
- Sign different parts of the message

If you need to create separate policies for such cases, interoperating with different vendor services can become arduous. This involves creating several complicated policies that might only differ in one field in each filter. To avoid this duplication, you can create a *policy assembly* that inserts the WS-Security `UsernameToken` into the message and generates the XML-Signature.

However, instead of explicitly configuring fields mentioned above (for example, enveloped or enveloping signature, include a `<wsse:BinarySecurityToken>`, or signing key to use), the policy assembly can use selectors for these fields, which are configured dynamically at runtime. For more details, see *Select configuration values at runtime* on page 421.

The policy assembly advertises that it requires configuration details to be called generically from other policies. For example, it requires the key to sign the message. By templating the signing policy as a policy assembly, and making it available to call from other policies like any other filter, the caller must set the signing key for the policy assembly. In this way, different policies that require a signed `UsernameToken` can call the same policy assembly. By using selectors to pass in different signing keys, messages are signed using the appropriate key for each calling policy.

When a policy has been configured as a policy assembly, it is displayed in the Policy Studio filter palette, and you can drag and drop it into any policy that requires the functionality in the assembly. You must configure any fields required by the policy assembly when it is dragged and dropped on to the canvas of another policy.

Configure a policy assembly

To configure a policy as a policy assembly in Policy Studio, perform the following steps:

1. Right-click the policy in the tree on the left, and select **Policy Assembly** > **Create**.
2. Specify the following settings on the **General** tab:
   - **Palette Category**:
     Enter the filter palette category in which to display the policy assembly (for example, `Monitoring`).
   - **Palette Icon**:
     Enter the path to the palette icon to display for the policy assembly (for example, `C:\Axway\apigateway\icons\monitor.ico`).
3. The **Input** tab lists all required message attributes for the policy assembly. You can enter user-friendly names for each attribute to be displayed in the **Policy Activation** filter for the policy assembly (for example, `HTTP Headers` for the `http.headers` attribute).
4. The **Output** tab lists the generated message attributes for the policy assembly. To add a generated attribute, click **Add**, and enter the following details:
   - **Expression**:
     Enter the selector expression for the attribute (for example, `${content.body}`).
   - **Attribute Type**:
     Enter the message attribute type (for example, `com.vordel.mime.Body`).
   - **Output Attribute Name**:
     Enter the message attribute generated by the policy assembly (for example, `content.body`).
5. When finished, click **OK**.

6. Click the **Deploy** button in the toolbar to deploy the newly created policy assembly to API Gateway.

Apply a policy assembly

When a policy is configured as a policy assembly, it is available for reuse in the Policy Studio filter palette. Dragging and dropping the policy assembly on to the policy canvas displays the **Policy Activation Filter** window for that policy assembly. This enables you to specify any required message attributes and filter-level monitoring settings.

Specify required attributes

The **Required Attributes** tab enables you to set the configuration fields required by the policy assembly (for example, those configured with selectors for dynamic configuration). Click **Add** to specify the following fields:

- **Required Attribute**:
  Enter the name of the required attribute for display (for example, `HTTP Header`).
- **Raw Attribute Name**:
  Enter the message attribute name (for example, `http.headers`).
- **Attribute Type**:
  Enter the message attribute type (for example, `com.vordel.mime.HeaderSet`).
- **Value/Selector**:
  Enter a message attribute value or selector (for example, `${http.headers}`).

Specify monitoring settings

The **Traffic Monitor** tab enables you to set the following filter-level monitoring settings:

- **Record outbound transactions**:
  Select whether to record outbound message transactions sent from API Gateway to remote hosts. This is enabled by default.
- **Record policy path**:
  Select whether to record the policy path for the message transaction, which shows the filters that the message passes through. This is enabled by default.

Web services 3

This section describes how to register and secure web services.

Register and secure web services

Overview

Policy Studio provides the following features that enable you to register and secure web services:

- You can register a web service in Policy Studio by importing a WSDL file into the web service repository. For more information on registering web services, see *Manage web services* on page 92.
- The **Import WSDL** wizard enables you to automatically generate the policies to protect web services. For more information on using the wizard, see *Configure policies from WSDL files* on page 77.
- You can also manually configure policies to protect web services (for example, if a WSDL file is not available). For more information, see *Configure policies manually* on page 65.
- You can also invoke registered web services from policies, for example, when you expose a SOAP web service as a REST API, the REST API you define calls a policy to implement the API, which in turn invokes the SOAP web service. For more information, see *Expose a web service as a REST API* on page 114.

WSDL and XML schema cache

API Gateway maintains a global cache of WSDL and XML schema documents. For more information on adding, updating, and deleting WSDL and XML schema documents, see *Manage WSDL and XML schema documents* on page 98.

API Gateway validates XML schemas and WSDL documents during import. For more information, see *XML schema and WSDL document validation* on page 104. To test a WSDL for WS-I compliance, see *Test a WSDL for WS-I compliance* on page 106.

JSON schema cache

API Gateway maintains a global cache of JSON schemas. For more information, see *Manage JSON schemas* on page 107.

WSDLs from a UDDI registry

WSDL documents can be imported from and published to a Universal Description, Discovery, and Integration (UDDI) registry. For more information, see the following topics:

- *Connect to a UDDI registry* on page 129
- *Retrieve WSDL files from a UDDI registry* on page 133
- *Publish WSDL files to a UDDI registry* on page 144

Data maps

You can create data maps to map XML and JSON messages to other XML and JSON message formats using a graphical editor. For more information, see *Manage data maps* on page 108.

Policy Studio filters

The following Policy Studio filters are of interest when working with web services:

- The **Web Service** filter is the main filter generated when a WSDL file is imported into the web service repository. It contains all the routing information and links all the policies that are to be run on the request and response messages into a logical flow.
- The **Schema Validation** filter is used to validate XML messages against XML schemas stored in the global cache.
- The **JSON Schema Validation** filter is used to validate JSON messages against JSON schemas stored in the global cache.
- You can use the **Execute Data Map** filter to execute a mapping you defined in a data map as part of a policy.

For more information on any of these filters, see the *API Gateway Policy Developer Filter Reference*.

Configure policies from WSDL files

Overview

When you import a WSDL file describing a web service into the web service repository, API Gateway exposes a *virtualized* version of the web service. This virtualized service is exposed on a host and port of the machine running API Gateway (for more information on how this is achieved see *Publish the WSDL* on page 97). In this way, a client can retrieve the WSDL for the virtualized web service from API Gateway without knowing its real location.

When you import a WSDL file, Policy Studio automatically generates policies that can be used to talk to the back-end web service. For example, a **Service Handler** is created to resolve and validate requests to the web service and responses from the web service. The service handler is automatically configured based on the operation definitions in the WSDL.

A WSDL file can include WS-Policy assertions that define the security policies or security requirements that a client must adhere to in order to communicate with the corresponding service. For example, a web service might require the client to sign sensitive parts of the message and include a WS-Security `UsernameToken` to authenticate to the web service.

If the imported WSDL file contains WS-Policy assertions, API Gateway is responsible for making sure the requests it sends to the web service adhere to the security constraints specified in the policy. In this case API Gateway is considered as the *initiator* of the web service.

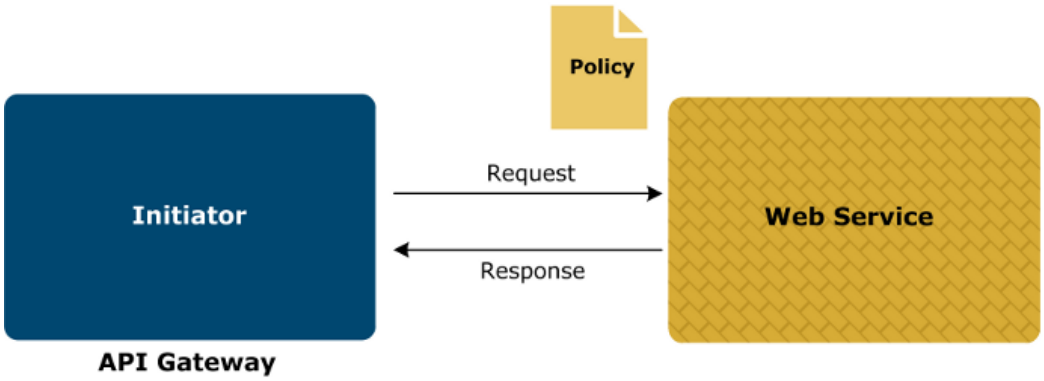
During the WSDL import process, you can select which operations to expose to clients and what path to expose the service on, and you can also specify the WS-Policy assertions that API Gateway enforces on the messages it receives from clients. In this case, API Gateway is considered to be the *recipient* to a consumer (that is, client) of the virtualized web service.

You can use the **Import WSDL** wizard to configure API Gateway as a web service initiator, as a recipient to a consumer of the virtualized web service, or both. The steps involved in the wizard, and the configuration windows displayed, differ according to whether WS-Policy assertions are being enforced between API Gateway and the web service (initiator case) or between a client and API Gateway (recipient case). For more information on the **Import WSDL** wizard, see *Import a WSDL file* on page 79.

The **Import WSDL** wizard enables you to configure complex policies to *secure and virtualize* web services with only a few clicks and minimal intervention.

API Gateway as the web service initiator

The following figure demonstrates the case where API Gateway is the initiator of the web service:

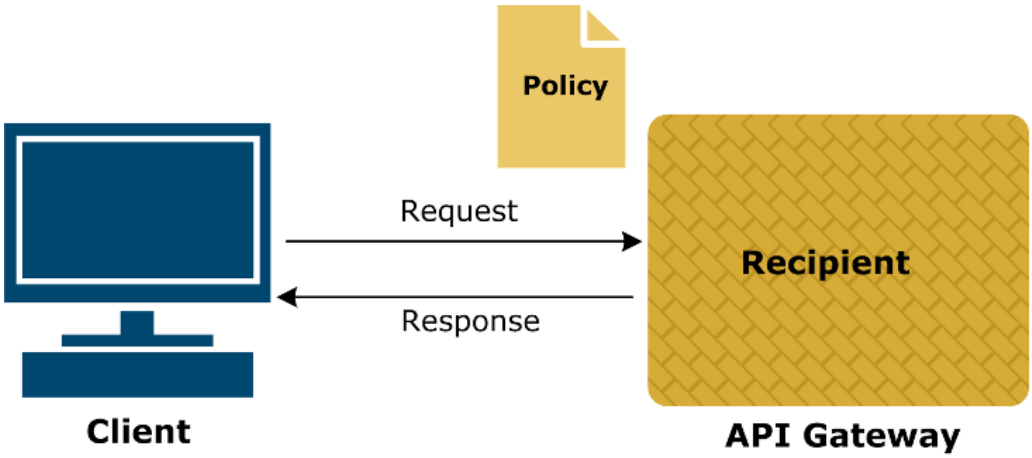


When the imported WSDL describing the web service includes WS-Policy assertions, API Gateway is responsible for making sure that the requests it sends to the web service adhere to the security constraints specified in the policy.

Policy Studio automatically generates the policies required to talk to the web service, and to enforce the security requirements specified by the policy assertions in the WSDL. These policies contain filters, most of which are configured automatically. However, a small number of filters require you to manually configure specific fields. For example, when signing or encrypting a message, you must specify the signing or encrypting key.

API Gateway as the web service recipient

The following figure demonstrates the case where API Gateway is the recipient of the web service:



When importing the WSDL, the **Import WSDL** wizard enables you to select the web service operations to expose to the client, set the path on which to expose the service, and optionally to specify the security policies that API Gateway enforces on the messages it receives from the client.

Policy Studio automatically generates the policies that API Gateway uses to talk to the client, and to enforce the security requirements you specified during WSDL import. These policies contain filters,

most of which are configured automatically. However, a small number of filters require you to manually configure specific fields. For example, when configuring the duration of a WS-Security timestamp, you might need to specify a longer or shorter duration.

Import WSDL summary

The steps involved in the **Import WSDL** wizard are summarized as follows:

1. Load the WSDL from a file, URL, or UDDI.
2. Retrieve and validate the WSDL.
3. Select the operations to expose.
4. Configure the WS-Policy settings. If the imported WSDL file includes WS-Policy settings, policies are generated to secure the connection between API Gateway and the web service. You can also secure the virtualized service by configuring WS-Policy settings between the client and API Gateway at this step.
5. Select the path to expose the services on.
6. If you selected to secure the virtualized service using WS-Policy settings, configure the policies that API Gateway needs to enforce for messages it receives. You can select gateway and message level policies at this step.
7. Configure the recipient security settings. At this step you can manually configure the policies that have been automatically generated by Policy Studio to enforce the WS-Policy settings between the client and API Gateway.
8. Configure initiator security settings. At this step you can manually configure the policies that have been automatically generated by Policy Studio to enforce the WS-Policy settings between API Gateway and the back-end web service.
9. Review the summary information for the virtualized service. At this step you can also override the default validation and routing policies with custom policies, and decide whether or not to publish the WSDL to clients.

Import a WSDL file

To import a WSDL file into the web service repository, complete the following steps:

1. In the Policy Studio tree view, expand the **APIs** > **Web Service Repository** node.
2. Right-click the **Web Services** node, and select **Register Web Service** to launch the **Import WSDL** wizard.
3. In the **Load WSDL** window, select the WSDL location from one of the following options:
    - **WSDL File**:
      Click the **Browse** button to select a WSDL file from the file system.

- **WSDL URL**:
  Enter a URL where the WSDL can be loaded from. This option supports HTTP basic authentication. Under **Authentication** select the **Use HTTP Basic** option and enter a user name and password.
- **WSDL from UDDI**:
  Click the **Browse UDDI** button to select a WSDL file from a Universal Description, Discovery, and Integration (UDDI) registry. For more information on how to retrieve a WSDL file from a UDDI registry, see *Retrieve WSDL files from a UDDI registry* on page 133.

Click **Next**.

4. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the WSDL. Click **Next** to continue.

   Note If the WSDL fails validation, an error is displayed. For more information, see *XML schema and WSDL document validation* on page 104 and *XML schema and WSDL document limitations* on page 104.

5. In the **WSDL Operations** window, select the operations to be exposed on the virtualized service and click **Next**. Alternatively, click **Select All** or **Select None** to select all or none of the operations defined in the WSDL file.
6. On the **WS-Policy Options** window, select **Secure this virtualized service with a WS-Policy** to use a WS-Policy to secure the virtualized service. If you select this option, the **Secure Virtual Service** dialog is displayed at a later step, which enables you to configure WS-Policy settings between the client and API Gateway and generate policies that API Gateway uses to enforce for messages it receives from clients.
7. On the **WS-Policy Options** window, select **Use the WS-Policy in the WSDL to connect securely to the back-end Web Service** to generate policies to secure the connection between API Gateway and the web service, based on the WS-Policy settings included in the WSDL file. This is enabled (and selected by default) only if the imported WSDL file includes WS-Policy information. Click **Next** to continue.
8. On the **Expose Services** window, select or enter a unique relative path on which to expose the virtualized service (for example, `Default Services`).

   Note If the WSDL file contains multiple services, you must select a relative path for each service before continuing. Select a service from the **Select a service** field, and then select or enter a unique relative path to expose that service on. Repeat for each service.

9. Click **Finish**. The configuration is prepared based on your selections and one or more of the following windows are then displayed:
   - **Secure Virtual Service**
     If you selected to secure the virtualized service with a WS-Policy, this window is displayed. For more information, see *Configure a security policy* on page 81.
   - **Configure Recipient Security Settings**
     This window is displayed if you selected to secure the virtualized service with a recipient WS-Policy. For more information, see *Configure recipient security settings* on page 82.

- **Configure Initiator Security Settings**
  This window is displayed if selected to use the WS-Policy in the WSDL to secure messages sent from API Gateway (as initiator) to the web service. For more information, see *Configure initiator security settings* on page 82.

10. After you have completed configuring the security settings, the **Summary** window is displayed. This window summarizes the details for the imported WSDL file. It includes the location of the WSDL file, and a tab for each web service virtualized by API Gateway. Each tab includes the following:
    - The name of the virtualized service.
    - The path that the WSDL for the virtualized service is exposed on.
    - **Validation**:
      API Gateway uses the WSDL to validate incoming messages by default. Alternatively, you can select the check box and click the browse button to use a dedicated validation policy for all messages sent to the web service. For example, this enables you to delegate to a custom validation policy used by multiple web services.
    - **Routing**:
      API Gateway routes to the displayed URL by default. Alternatively, you can select the check box and click the browse button to use a dedicated routing policy to send all messages on to the web service. For example, this enables you to delegate to a custom routing policy used by multiple web services.
    - **WSDL Access Options**:
      Select the **Allow the Gateway to publish WSDL to clients** option to make the WSDL for this web service available to clients. This option is selected by default. The published WSDL represents a virtualized view of the web service. Clients can retrieve the WSDL from API Gateway, generate SOAP requests, and send them to API Gateway, which routes them on to the web service. For more details, see *Publish the WSDL* on page 97.

    These options enable you to configure the underlying auto-generated **Service Handler** without having to navigate to it in the **Policies** tree. Review the information on each tab and click **OK** to close the wizard.

Configure a security policy

The **Secure Virtual Service** dialog enables you to specify the policy that API Gateway enforces on the messages that it receives from the client. To specify a policy, perform the following steps:

1. Select a policy from the **Gateway Policy** field. A description for the currently selected policy is displayed in the dialog. For details on all the available policies, see *WS-Policy reference* on page 447.

2. Select a message-level request policy from the **Request Policy** field. The available policies are as follows:
    - Encrypt SOAP Body
    - Sign SOAP Body
    - Sign and Encrypt SOAP Body
3. Select a message-level response policy from the **Response Policy** field. The available policies are the same as for **Request Policy**.
4. Click **OK**.

Configure recipient security settings

If API Gateway is configured as a recipient for the client, the **Configure Recipient Security Settings** window is displayed. This window enables you to configure the filters used to implement the rules required by the WS-Policy settings you selected in the **Secure Virtual Service** window. The exact sequence of filters differs depending on the policies that you selected.

For example, if a policy with a SAML token is selected, the **Validate SAML Authentication Assertion** filter is displayed, whereas if a WS-Policy requiring a WS-Security `UsernameToken` is selected, the **Validate WS-Security Username Token** filter is displayed. The effort required to configure these filters is minimal because most of the fields are populated automatically from the WS-Policy assertions. For example, the layout, signing, encryption, and key wrapping algorithms, key referencing method, user name digest, and clear password are all automatically populated from the WS-Policy assertions. This means that you only need to configure a small number of settings, such as the signing key, encryption certificate, and timestamp validity period.

Configure initiator security settings

If API Gateway is configured as an initiator of the web service, the **Configure Initiator Security Settings** window is displayed. This window enables you to configure the filters used to implement the rules required by the WS-Policy settings specified in the imported WSDL file. The exact sequence of filters differs depending on the WS-Policy assertions contained in the WSDL.

For example, if an `sp:SamlToken` assertion is specified, the wizard contains a window for the **Insert SAML Authentication Assertion** filter. The effort required to configure these filters is minimal because most of the fields are populated automatically from the WS-Policy assertions in the WSDL.

In the case of the **Sign Message** filter, the decision to use asymmetric or symmetric signatures is based on whether the policy uses an asymmetric or symmetric binding. The layout rules are determined by the `sp:Layout` assertion. The digest method, signature method, and key wrap algorithm (for symmetric signatures) are all populated automatically based on the contents of the `sp:AlgorithmSuite` assertion. The `KeyInfo` section of the XML Signature can be taken from various properties set in the WSDL. The parts of the message to be signed can be inferred from assertions such as `sp:SignedParts`, `sp:SignedElements`, and `SignedSupportingTokens`.

The same is true for the **XML Encryption Settings** filter where the encryption algorithms and key types can all be taken from the assertions in the WSDL. The `ConfirmationMethod` for SAML assertions can be inferred from the context of the `SamlToken` assertion. For example, an `sp:SamlToken` that appears as a child of the `sp:SignedSupportingTokens` assertion uses a `sender-vouches` confirmation method, whereas if it appears as a child of an `sp:EndorsingSupportingTokens` assertion, the `holder-of-key` confirmation method can be assumed.

Configure recipient policy filters

The following tables describe the filters that can be created when API Gateway is configured as a recipient. You can configure these filters in the **Configure Recipient Security Settings** window. For simplicity, only filters that require manual input are shown, and only the fields that might require manual input are detailed in the tables.

Insert Timestamp Filter

Field Name	Description
Expires In	You can specify a more appropriate lifetime for the assertion (instead of the default one hour) by configuring the various time period fields.

Signed Parts Outbound Filter

Field Name	Description
Signing Key	If the policy uses an asymmetric binding, on the **Asymmetric** tab, click the **Signing Key** button, and select a key from the certificate store to sign the message parts with. Alternatively, if the policy specifies a symmetric binding, on the **Symmetric** tab, click the **Signing Key** button, and select a key to wrap the symmetric signing key with.

Find Recipient Certificate for Encryption

Field Name	Description
Certificate Store	Click the **Select** button to choose the recipient's certificate from the certificate store. The public key contained in this certificate is used to encrypt the message parts so that only the recipient is able to decrypt them using the corresponding private key.

Validate SAML Authentication Assertion

Field Name	Description
Drift Time	You can specify a drift time value to allow for a time differential between the clock on the machine hosting API Gateway and the machine hosting the web service.
Trusted Issuers	On the **Trusted Issuers** tab, click **Add** to specify the Distinguished Name of a SAML authority whose certificate has been added to the certificate store, and click **OK**. Repeat this step to add more SAML authorities to the list of trusted issuers.

Configure SSL Certificate

Field Name	Description
X.509 Certificate	On the **Network** tab, click the **X.509 Certificate** button to create or import an SSL certificate.
SSL Server Name Identifier (SNI)	On the **Network** tab, click the **Add** button to configure a server name in the **SSL Server Name Identifier (SNI)** dialog. You can specify the server name in the **Client requests server name** field. Click the **Server assumes identity** button to import a certificate authority certificate into the certificate store.
Mutual Authentication	On the **Mutual Authentication** tab, select root certificate authorities trusted for mutual authentication.

Insert MTOM Content

Field Name	Description
XPath Location	When the `wsoma:OptimizedMimeSerialization` WS-MTOM Policy assertion is specified in a recipient policy, you must configure an **Insert MTOM Content** filter. Configure an XPath expression that points to the base64-encoded element content to insert and create an MTOM type attachment for.

Configure initiator policy filters

The following tables describe the filters that can be created when API Gateway is configured as an initiator. You can configure these filters in the **Configure Initiator Security Settings** window. For simplicity, only filters that require manual input are shown, and only the fields that might require manual input are detailed in the tables.

Insert WS-Security Timestamp

Field Name	Description
Expires In	You can specify a more appropriate lifetime for the assertion (instead of the default one hour) by configuring the various time period fields.

Sign Message

Field Name	Description
Signing Key	If the policy uses an asymmetric binding, on the **Asymmetric** tab, click the **Signing Key** button, and select a key from the certificate store to sign the message parts with. Alternatively, if the policy specifies a symmetric binding, on the **Symmetric** tab, click the **Signing Key** button, and select a key to wrap the symmetric signing key with.

Insert WS-Security Username

Field Name	Description
Username	Enter the user name inserted into the WS-Security `UsernameToken` block. By default, the name of the authenticated user is used, which is stored in the `authentication.subject.id` message attribute. However, you can enter any value in this field.
Password	If the policy requires a password, the password for the user entered above must be specified here. You can use the default authenticated user password by selecting the `authentication.subject.password` message attribute. Alternatively, you can enter any suitable password. The decision to use a **Clear** or **Digest** password is taken from the corresponding policy assertions.

Insert SAML Authentication Assertion

Field Name	Description
Expire In	Specify a suitable lifetime for the SAML assertion by configuring the various time period fields.
Drift Time	You can specify a drift time value to allow for a time differential between the clock on the machine hosting API Gateway and the machine hosting the web service.
Issuer Name	Select the alias of the certificate from the certificate store to use to identify the issuer of the assertion. The alias name is used as the value of the `Issuer` attribute of the `saml:Assertion` element.
Holder of Key: Signing Key	In cases where the `sp:SamlToken` appears as a child of `EndorsingSupportingTokens` or an `InitiatorToken`, the `holder-of-key` SAML confirmation method is inferred. In this case, if an asymmetric binding is used, on the **Asymmetric** tab, specify a key from the certificate store by clicking the **Signing Key** button. Alternatively, if a symmetric binding is used in the policy, on the **Symmetric** tab, specify a key to use to encrypt the symmetric key with by clicking the **Signing Key** button.

Find Recipient Certificate for Encryption

Field Name	Description
Certificate Store	Select this option, and click the **Select** button to choose the recipient's certificate from the certificate store. The public key contained in this certificate is used to encrypt the message parts so that only the recipient is able to decrypt them using the corresponding private key.

Connect to URL

Field Name	Description
Trusted Certificates	To connect to an external web service over SSL, you need to trust that web service's SSL certificate. You can do this on the **Trusted Certificates** tab of the **Connect to URL** filter. Assuming you have already imported this certificate into the trusted certificate store, simply select it from the list.
Client SSL Authentication	If the web service requires the client to present an SSL certificate to it during the SSL handshake, you must select that certificate from the list on the **Client SSL Authentication** tab. **Note** This certificate must have a private key associated with it that is also stored in the certificate store.

Extract MTOM Content

Field Name	Description
XPath Location	When the `wsoma:OptimizedMimeSerialization` WS-MTOM Policy assertion is specified in an initiator policy, you must configure an **Extract MTOM Content** filter. Configure an XPath expression that points to the base64-encoded element content to extract and create an MTOM type attachment for.

Edit the recipient or initiator WS-Policy

To edit a previously configured WS-Policy (for example, to change the signing key in the auto-generated policy), right-click the web service in the Policy Studio tree, and select **WS-Policy > Edit Recipient WS-Policy** or **WS-Policy > Edit Initiator WS-Policy**. You can also add or

remove a recipient WS-Policy after import by selecting the **WS-Policy > Add Recipient WS-Policy** or **WS-Policy > Remove Recipient WS-Policy** options. These options are described as follows:

Edit Recipient WS-Policy
If you configured a recipient WS-Policy during WSDL import (using the **Secure Virtual Service** window), you can edit its filters using this option. If you did not configure a recipient WS-Policy during WSDL import, this option is disabled.

Add Recipient WS-Policy
If you did not configure a recipient WS-Policy during WSDL import (using the **Secure Virtual Service** window), this option enables you to add a recipient WS-Policy after import. The **Secure Virtual Service** window is displayed when you select this option followed by the **Configure Recipient Security Settings** window. This enables you to select and configure a WS-Policy to enforce between the client and API Gateway.

Remove Recipient WS-Policy
If you configured a recipient WS-Policy during WSDL import (using the **Secure Virtual Service** window), you can remove it using this option. If you did not configure a recipient WS-Policy during WSDL import, this option is disabled.

Edit Initiator WS-Policy
If the imported WSDL file included WS-Policy assertions, you can edit the filters used to implement the WS-Policy assertions in the WSDL using this option.

However, if there was no WS-Policy in the imported WSDL file, you cannot use this option. You cannot add an initiator WS-Policy after WSDL import because that would break the contract between API Gateway and the back-end web service. If the contract for the web service changes (for example, a WS-Policy is applied to it at the back-end), you can reimport the WSDL.

Configure a recipient WCF WS-Policy

API Gateway provides four WS-Policies that are identical to those exposed by WCF (Windows Communication Foundation) web services. When one of these policies is exposed by a virtual service in API Gateway, the `svcutil` .NET web services utility can consume the WS-Policy and auto-generate clients that communicate securely with API Gateway.

The security settings for a WCF web service are configured in its `web.config` file, in which the `security` element determines the WS-Policy applied to the service. For example, the following extract from a WCF web service `web.config` file shows the configuration:

```
<customBinding>
  <binding name="MyCustomBinding">
    <textMessageEncoding messageVersion="Soap11" />
    <security defaultAlgorithmSuite="Basic256"
allowSerializedSigningTokenOnReply="true"
      authenticationMode="MutualCertificate" requireDerivedKeys="false"
includeTimestamp="true"
      messageProtectionOrder="SignBeforeEncrypt"
```

```
messageSecurityVersion="WSSecurity10..."
      requireSecurityContextCancellation="false">
    </security>
  </binding>
</customBinding>
```

In this example, the `authenticationMode` for a `customBinding` is set to `MutualCertificate`, which means that messages sent to and from the web service must be signed and encrypted with mutual certificates. The following example shows an example of the WCF `wsHttpBinding` configuration, which is less verbose:

```
<wsHttpBinding>
  <binding name="MyWsHttpBinding">
    <security mode="Message">
      <message clientCredentialType="Certificate" />
    </security>
  </binding>
</wsHttpBinding>
```

The following table shows how the WCF WS-policies provided with API Gateway correspond to a particular configuration of the `security` element in the WCF web service `web.config` file. As shown in the preceding examples, the configuration settings are slightly different, depending on the WCF binding (`customBinding` or `wsHttpBinding`). The following table shows the available settings:

WS-Policy Name	WCF Binding	Authentication Mode	Security Mode	Client Credential Type
WCF MutualCertificate Service	`customBinding`	`MutualCertificate`	—	—
WCF UsernameForCertificate Service	`customBinding`	`UserNameForCertificate`	—	—
WCF UsernameOverTransport Service	`customBinding`	`UsernameForTransport`	—	—
WCF BrokeredX509Authentication Service	`wsHttpBinding`	—	`Message`	`Certificate`

Note If you intend to consume the WS-Policy exposed by API Gateway with a WCF client, you should use one of the WCF WS-Policies. All of these policies can be consumed seamlessly by the WCF `svcutil` utility to auto-generate secure clients. While the other WS-Policies

exposed by API Gateway can be consumed by `svcutil`, you need to make additional configuration changes to the auto-generated WCF client to communicate securely with API Gateway. For more details on making any necessary configuration changes, see your WCF documentation.

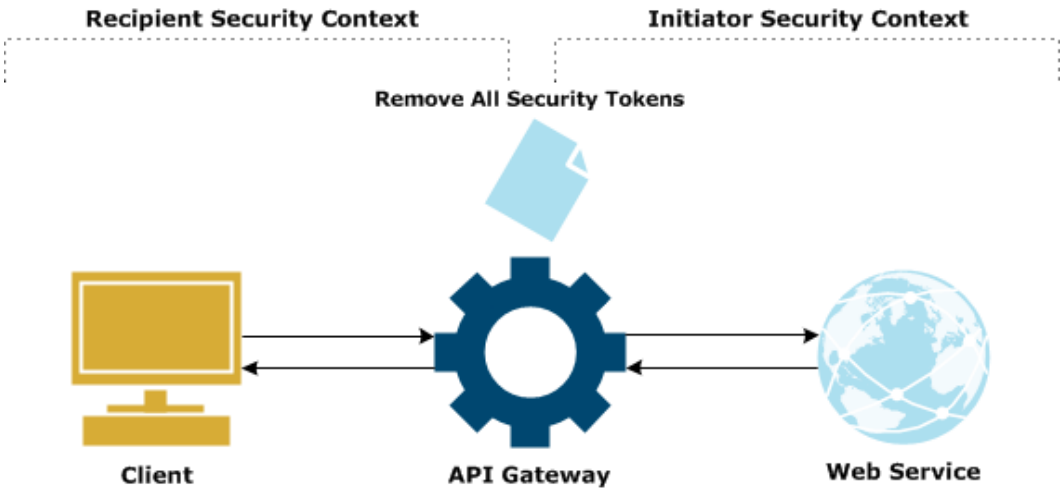
Remove security tokens

When you import a WSDL file containing a WS-Policy into the web service repository, the **Remove All Security Tokens** filter is enabled in the **Service Handler** for the imported web service. To view the configured policy, double-click the **Service Handler**, and select the **Message Interception Points > 2. User-defined Request Hooks** tab.

The **Remove All Security Tokens** policy ensures that the following security contexts are kept separate:

- *Recipient security context*: This is between the client and API Gateway, and is determined by the WS-Policy selected in the **Secure Virtual Service** window.
- *Initiator security context*: This is between API Gateway and the back-end web service, and is determined by the WS-Policy contained in the imported WSDL file for the back-end web service.

The **Remove All Security Tokens** policy prevents tokens from one context passing over into the other context, which could breach the security contract governing that context. This ensures that each security context receives a clean SOAP message, on which it can then act to enforce the security requirements of the relevant WS-Policy. The following diagram shows both security contexts and the **Remove All Security Tokens** policy:



Recipient-side WS-Policy only

In this case, a recipient WS-Policy is configured in the **Secure Virtual Service** window to protect a virtual service exposed by API Gateway. The recipient WS-Policy defines the security contract between the client and API Gateway. Any security tokens sent by the client are intended for consumption by API Gateway. They are *not* intended for the back-end web service.

For example, the web service might not understand SAML, WS-Security, XML Signature, and so on, which might result in a serialization error if these tokens are propagated to it. In addition, it would add unnecessary overhead to the message to propagate security tokens to it. On the response side, the response that API Gateway returns to the client must adhere to the selected recipient WS-Policy. For example, if the web service has returned a SAML token (out of scope of any WS-Policy requirements), this must not be returned to the client because it would breach the recipient WS-Policy.

Initiator-side WS-Policy only
In this case, the WS-Policy is contained in the imported WSDL file. The WS-Policy defines the security contract between API Gateway and the back-end web service defined in the WSDL. On the request side, any security tokens sent by the client to API Gateway, which are out of scope of the initiator WS-Policy between API Gateway and web service, are removed before API Gateway starts enforcing the initiator WS-Policy on the request, and before it sends the request to the web service.

For example, if the client sends a `wsu:Timestamp` in the request message and the initiator policy stipulates that a `wsu:Timestamp` must be sent by API Gateway to the web service, two timestamps could be sent in the request, which is invalid. This means that the timestamp and any other security tokens sent by the client to API Gateway, which might contradict the rules in the initiator contract (between API Gateway and the web service) must be stripped out before API Gateway starts adding security tokens to the message. This ensures that the message adheres to the initiator WS-Policy.

Similarly, any security tokens returned by the web service are only present because the web service complies with the contract between the web service and API Gateway. Therefore, any tokens returned by the web service are only intended for use by API Gateway. They are *not* intended for consumption by the client. In other words, the security context is only between API Gateway and the web service. If the web service returns a `UsernameToken`, it is consumed by API Gateway.

If a token must be returned to the client, this is a user-enforced rule, which is out of scope of the WS-Policy configuration in the WSDL. If necessary, you can override the default behavior by removing the **Remove All Security Tokens** filter from the **Service Handler** to allow the `UsernameToken` to be propagated to the client.

Initiator-side and Recipient-side WS-Policy
This occurs when you import a WSDL file that includes a WS-Policy (initiator case), and you also select a WS-Policy in the **Secure Virtual Service** window (recipient case). This scenario includes both the *recipient security context* between the client and API Gateway, and the *initiator security context* between API Gateway and the web service.

It is vital that these security contexts are kept separate because if tokens from one context pass over into the other context, it is highly likely that the security contract for that context will be breached. For example, if the recipient contract between the client and API Gateway requires a `UsernameToken`, but the initiator contract between API Gateway and the web service requires a SAML token, the `UsernameToken` must not pass over into the initiator context and be sent to the web service.

Note The **Remove All Security Tokens** policy only applies when a WS-Policy is configured, and is *not* enabled when a WS-Policy is not configured. In addition, any non-standard

behavior that requires a security token to be propagated over to another security context can be handled by disabling the **Remove All Security Tokens** policy in the **Service Handler** for the imported WSDL.

Manage web services

Overview

The **Web Service Repository** stores information about web services whose definitions have been imported using Policy Studio. The WSDL files that contain these web services definitions are stored together with their related XML schemas. Clients of the web service can then query the repository for the WSDL file, which they can use to build and send messages to the web service using API Gateway.

The web service repository enables you to register web services by importing a WSDL file, and to group related web services into groups. When you register a web service in the web service repository, Policy Studio auto-generates a **Service Handler** that is used to control and validate requests to the web service and responses from the web service. If a web service is updated after initial import, you can resynchronize with it to import new versions of the WSDL and XML schemas, and the **Service Handler** is regenerated to reflect the updates.

This topic describes how to manage web services and groups, and explains what happens when you import a WSDL file to register a web service. This topic also describes how to export a registered web service, how to update (or resynchronize) an existing web service, and how to expose additional operations on a web service.

Manage web services and groups

The **Web Service Repository** is displayed under the **APIs** node in the Policy Studio tree. WSDL files are imported into web service groups, which provide a convenient way of keeping groups of related web service definitions together. To import a WSDL file into the default group, right-click the **Web Services** node, and select **Register Web Service**.

Alternatively, to add a new web service group, right-click the default **Web Services** group, or the **Web Service Repository** node, and select **Add a new web services group**. When the new group is added, you can right-click it in the tree, and select **Register Web Service**.

Register a web service

Registering a web service involves importing the WSDL file that contains the definitions for the web service. Policy Studio provides an **Import WSDL** wizard to make registering a web service a simple process that requires minimal manual intervention. For more information on registering a web service by importing a WSDL file, see *Configure policies from WSDL files* on page 77.

The **Import WSDL** wizard auto-generates policies and service handlers for each web service imported. These are automatically configured wherever possible, based on the imported WSDL. This means that only a small number of fields need to be configured manually.

After a web service has been registered using the **Import WSDL** wizard, the WSDL for the service is *published* by API Gateway. Clients can specify `WSDL` on a request query string to retrieve the WSDL file (for example, `http://localhost:8080/HelloWorldService/HelloWorld?WSDL`). For more details, see *Publish the WSDL* on page 97.

Tip You can also register a web service using a script. For more information, see *Use scripts to manage web services* on page 97.

Results of registering a web service

Service handlers and policies are auto-generated when you register a web service by importing a WSDL file in Policy Studio. The specific policies that are created depends on whether you configured a policy to enforce security between the client and API Gateway, and whether the imported WSDL file contained any WS-Policy assertions. The following list summarizes what is created by the wizard:

Web service
A new web service tree node is created in the **Web Service Repository** tree for each web service that you selected to expose operations from in the imported WSDL. The web service is created in the **Web Services** group by default.

Click the new web service node to list the WSDL file for the service and any imported WSDL files and XML schemas. To view the source for a WSDL or XML schema file, click the file and a read-only view of the source is displayed.

Note It is important to realize that the WSDL displayed in this view represents the WSDL that is exposed to the client. Therefore, it contains only those operations that were selected during the import process, together with any recipient WS-Policy that has been applied to the virtualized web service. You can view the WSDL in its original form under the **Resources > WSDL Document Bundles** node.

You can add, remove, or edit recipient WS-Policies for the service from this node. For example, to edit an existing recipient WS-Policy, right-click the web service node, and select **WS-Policy > Edit Recipient WS-Policy**. If the imported WSDL file included WS-Policy assertions, you can also edit the initiator WS-Policy (for example, to change the signing key). Right-click the web service node and select **WS-Policy > Edit Initiator WS-Policy** . For more information, see *Edit the recipient or initiator WS-Policy* on page 87.

WSDL resources
A new WSDL document bundle is created for the imported WSDL file under the **Resources > WSDL Document Bundles** tree node. This document bundle contains the original WSDL document for the back-end web service, as retrieved at the time of the WSDL import. If multiple versions of this WSDL have been imported, each version is available here. The document bundle also contains the original versions of any resources imported by the WSDL, such as other WSDL files or XML schemas.

Click the WSDL document bundle to list the versions that have been imported. Click a particular version to list the WSDL file and any imported WSDL files and XML schemas, as they appeared for that version. To view the source for a WSDL or XML schema file, click the file and a read-only view of that version of the source is displayed.

For more information about WSDL document bundles, see *Manage WSDL and XML schema documents* on page 98.

Policy container
A container for the newly generated policies is created under the **Generated Policies** node in the **Policies** tree. The new container is named after the service (for example, `Web Services.HelloWorldService`).

Policy
A policy for the web service is created in the generated policy container. The policy name is the name of the service (for example, `HelloWorldService`). This policy contains the **Service Handler** for the service. The service handler is an auto-configured **Web Service Filter**.

Service handler
The **Service Handler** is used to control and validate requests to the web service and responses from the web service. The **Service Handler** is named after the web service (for example, `Service Handler for 'HelloWorldService'`). The **Service Handler** is a **Web Service Filter**, and is used to control the following:

- Routing
- Validation
- Message request/response processing
- WSDL options
- Monitoring options

For more details, see "Web service filter" in the *API Gateway Policy Developer Filter Reference*.

Security policies
If you configured a WS-policy to enforce security between the client and API Gateway (as described in *Configure a security policy* on page 81), or if the imported WSDL file contained WS-Policy assertions, a number of additional policies are automatically created in a generated policy container named `WSPolicy`. Any recipient policies are created in a container named `Recipient` and any initiator policies are created in a container named `Initiator`. These generated policies include the filters required to generate and validate the relevant security tokens (for example, SAML tokens, WS-Security `UsernameToken` elements, and WS-Addressing headers). These policies perform the necessary cryptographic operations (for example, signing/verifying and encryption/decryption) to meet the security requirements of the specified policies.

Export a web service

To export a web service, right-click on the web service node under the **Web Service Repository** and select **Export Web Service**.

The following items are exported:

- All circuit containers for the web service, including policies and containers that were generated as part of WS-Policy configuration.
- The current version of the WSDL and its schemas.

  Note If multiple versions of the WSDL are available, only the current version is exported. The complete history of the WSDL is *not* exported.

- Optionally, other configuration used by the policies can be exported (for example, remote host configuration).

Update a web service

Over the lifetime of a web service, the service definition for the web service can change. You can manage the lifecycle of a web service easily, by using the **Resynchronize Web Service** option in Policy Studio. This enables you to update the web service definition for a service by revisiting the location of the WSDL.

To update a web service, right-click the web service node, and select **Resynchronize Web Service**. You are prompted for the location of the latest version of the WSDL. This defaults to the location from which the WSDL was originally loaded. API Gateway compares the contents of the WSDL and any of its imported schemas to those stored in its cache. If the contents are different, it creates a new version of the WSDL.

Examples of possible updates to the WSDL or XML schemas that trigger creation of a new version are as follows:

- **Service name (local part)**
  If the service name has changed in the WSDL, the import exits with the error `"Couldn't find target Service in this WSDL"`. You can import this WSDL definition as a new web service using the **Register Web Service** option.
- **WSDL namespace**
  If the namespace has changed in the WSDL, the import continues with a warning.

  Note Namespace changes result in a new node being created under the **Resources > WSDL Document Bundles** tree. Any subsequent resynchronizations of the service result in new versions being added to this node (assuming no further namespace changes).

- **WSDL 2.0**
  If the WSDL has changed to use the WSDL 2.0 specification, import exits with an error. API Gateway supports WSDL 1.1 only, it does not support WSDL 2.0.
- **Operation added**
  If a new `operation` has been added to a `portType` in the WSDL, import succeeds. API Gateway regenerates the configuration, using older operations as a template for the WS-Policy settings, if attached. The service handler for the web service is updated to include a resolver for the new operation.

- **Operation modified**
  If an `operation` has been modified on a `portType` in the WSDL (for example, changes to the `input`, `output`, or `fault`), import succeeds. API Gateway regenerates the configuration and updates the service handler for the web service.
- **SOAPAction**
  If the `SOAPAction` has changed for an `operation` in the WSDL, import succeeds. API Gateway regenerates the configuration and updates the service handler for the web service.
- **Operation removed**
  If an `operation` has been removed from a `portType` in the WSDL, import succeeds. API Gateway removes the operation from the configuration. The service handler for the web service is updated to remove the resolver for the removed operation.
- **XML schema (when WSDL validation is in use)**
  If you are using the WSDL to validate incoming messages (this is the default option when you import a WSDL file), and the schemas in the WSDL have been updated, API Gateway retrieves the updated schemas.

Updating a web service is *not* possible for the following types of WSDL or XML schema changes:

- Port, binding, or operation changes that require user intervention for WS-Policy configuration.
- Port change (new endpoint location).
- Port added (SOAP flavor).
- Port change (new binding).
- Binding style or use change.
- WS-Policy added.
- WS-Policy modified.
- WS-Policy removed.
- Operation message part changes (for example, reference to an element changes to a type, part name changes, and so on).
- XML schema changes (when custom schema validation is in use).

Change the operations exposed by a web service

When you register a web service by importing a WSDL file, you are prompted to select the operations that are to be exposed to the client. You can change the operations that are exposed after import by using the **Select Exposed Operations** option in Policy Studio. For example, if you imported a web service containing two operations, `foo` and `bar`, and you only selected the `foo` operation at import time, you can use this feature to also expose the `bar` operation after import.

To change the operations exposed by a web service, right-click the web service node, and select **Select Exposed Operations**. All of the operations defined in the WSDL file are displayed, and the operations that are currently exposed are selected. Select the operations to be exposed and deselect the operations that are not to be exposed and click **OK**.

API Gateway regenerates the configuration. The service handler for the web service is updated to include resolvers for any newly exposed operations, and to remove resolvers for any removed operations. The WSDL exposed to clients of the virtualized service is updated to reflect the changes.

Delete a web service

To delete a web service, right-click on the web service node under the **Web Service Repository** and select **Delete**.

Note To delete the WSDL document or XML schemas associated with a web service, see *Delete cached WSDL or XML schema documents* on page 103.

Use scripts to manage web services

You can also use scripts to manage web services. The following sample scripts are provided in the `INSTALL_DIR/apigateway/samples/scripts/ws` directory:

- `listWebServices.py` – List web services
- `registerWebService.py` – Register a web service
- `removeWebService.py` – Delete a web service

You could also create your own custom scripts, for example, to update a web service. However, it would only be possible to script the same type of WSDL or XML schema updates that are supported by the Policy Studio **Resynchronize Web Service** option. For more information on the types of WSDL or XML schema updates that are supported, see *Update a web service* on page 95.

Publish the WSDL

When the WSDL has been imported into the web service repository, it can be retrieved by clients. In effect, by importing the WSDL into the repository, you are *publishing* the WSDL. In this way, consumers of the services defined in the WSDL can learn how to communicate with those services by retrieving the WSDL for those services. However, to do this, the location of the service must be changed to reflect the fact that API Gateway now sits between the client and the defined service.

For example, assume that the WSDL file states that a particular service resides at `http://www.example.com/services/myService`:

```
<service name="myService">
  <port binding="SoapBinding" name="mySample">
    <wsdl:address location="http://www.example.com/services/myService"/>
  </port>
</service>
```

When deployed behind API Gateway, this URL is no longer accessible to consumers of the service. Because of this, clients must send SOAP messages through API Gateway to access the service. In other words, they must now address the machine hosting API Gateway instead of that directly hosting the service.

When returning the WSDL to the client, API Gateway dynamically changes the value of the `location` attribute in the `service` element in the WSDL file to point to the machine on which API Gateway resides. API Gateway is responsible for routing messages on to the machine hosting the service.

Assuming that API Gateway is running on port `8080` on a machine called `SERVICES`, the location specified in the exported WSDL file is changed to the following:

```
<service name="myService">
  <port binding="SoapBinding" name="mySample">
    <wsdl:address location="http://SERVICES:8080/services/myService"/>
  </port>
</service>
```

When the client retrieves this modified WSDL file, it routes messages to the machine hosting API Gateway instead of attempting to directly access the web service.

Access the WSDL

For the client to access this modified WSDL file, Policy Studio provides a WSDL retrieval facility whereby clients can query the web service repository for the WSDL file for a particular web service. To do this, the client must specify `WSDL` on a request query string to the relative path mapped to the policy for this web service.

For example, if the policy is deployed under `http://SERVICES:8080/services/getQuote`,the client can retrieve the WSDL for this web service by sending a request to `http://SERVICES:8080/services/getQuote?WSDL`. When the client has a copy of the updated WSDL file, it knows how to create correctly formatted messages for the service, and more importantly, it knows to route messages to API Gateway rather than to the web service directly.

Publish to UDDI

For details on how to publish a WSDL file registered in the web service repository to a UDDI registry, see *Publish WSDL files to a UDDI registry* on page 144.

Manage WSDL and XML schema documents

Overview

WSDL files often contain XML schemas that define the elements that appear in SOAP messages. When you import a WSDL file to register a web service, the imported WSDL file, and any XML schemas included in the WSDL, are added to a global cache of WSDL and XML schema documents. You can also add XML schemas and WSDL documents to the cache independently.

If you select a cached WSDL file or XML schema in a **Schema Validation** filter, API Gateway can retrieve it from the cache instead of fetching it from its original location. This improves the runtime performance of the filter, and also ensures that an administrator has complete control over the schemas used to validate messages.

API Gateway can maintain multiple versions of WSDL and XML schema documents in the global cache, and keeps an explicit version history as they change over time. The cache is prepopulated with many of the common XML schema documents that are used in web services, and these are shared across multiple web services.

Structure of the global cache

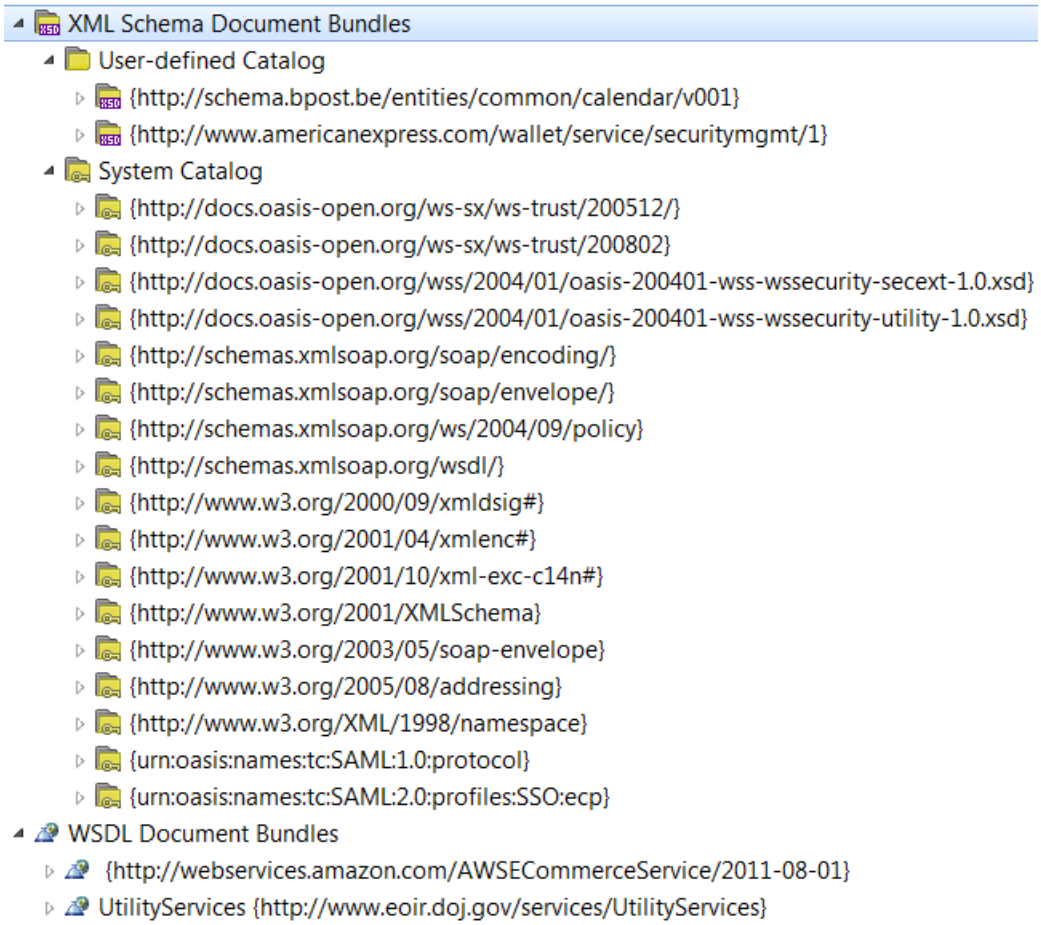
The global cache consists of both WSDL documents and XML schema documents.

The global cache of WSDL documents contains all WSDL documents that have been imported (either directly, or by registering a web service) into API Gateway.

The global cache of schema documents contains:

- User-defined catalog – This contains all user-defined schema documents that have been imported into API Gateway.
- System catalog – This contains all common schemas (for example, the SOAP encoding schema) that are preloaded during API Gateway installation.

The following figure shows the structure.



View cached WSDL or XML schema documents

Policy Studio provides a *read-only* view of cached WSDL and XML schema documents.

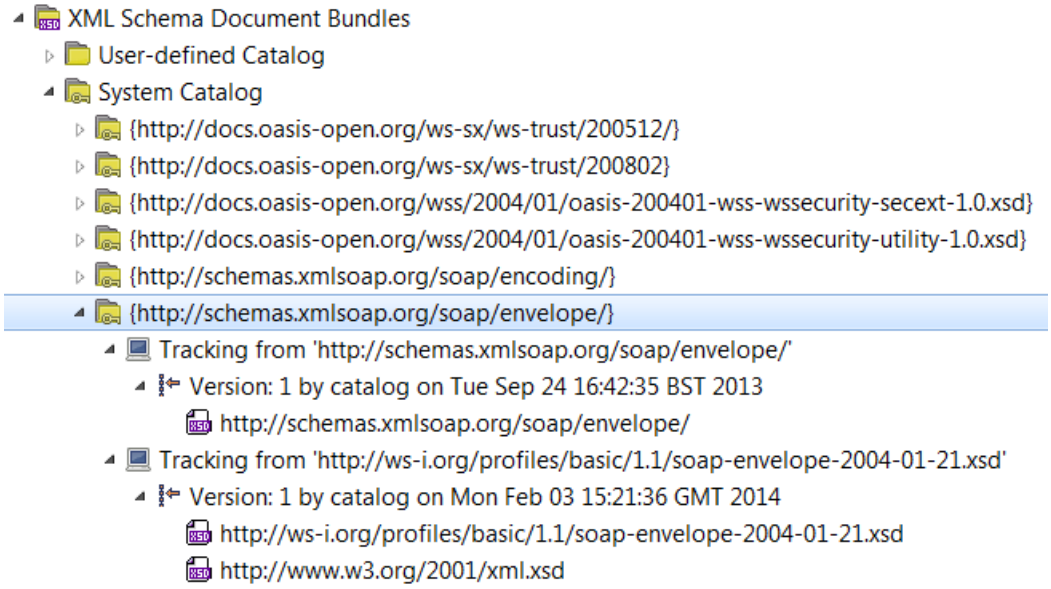
To view the global cache of WSDL documents, expand the **Resources > WSDL Document Bundles** tree node. The list of documents present in the cache is shown in the tree. To view the contents of a document, click the document node. A read-only view of the document is displayed in a tab on the right. WSDL documents can be added directly to this node, and they can be resynchronized.

To view the global cache of XML schema documents, expand the **Resources > XML Schema Document Bundles** tree node. This node contains two subnodes: **User-defined Catalog** and **System Catalog**. To view the documents in each catalog, expand the catalog node. The list of documents present in the catalog is shown in the tree. To view the contents of a document, click the document node. A read-only view of the document is displayed in a tab on the right.

XML schemas in the **System Catalog** are preloaded during API Gateway installation. You cannot add schemas to the system catalog, and schemas in the system catalog cannot be resynchronized. This is indicated by a key icon on these nodes.

The **User-defined Catalog** contains schemas that you have imported. These schemas can be resynchronized, and you can add new schemas directly to this catalog.

Document bundles are categorized by namespace/name, with subcategories used to indicate where the node is being tracked from (the URL it was retrieved from), and further subcategories for version information. The following figure shows an example of this.



Click a document bundle in the tree to list the versions that have been imported. Click a version to list all the documents contained in that version. Click a WSDL or XML schema document to view the WSDL or XML source for the document. A read-only view of the source is shown in a tab on the right.

Add XML schemas to the cache

To add an XML schema to the user-defined catalog in the global cache, perform the following steps:

1. In the Policy Studio tree, right-click the **XML Schema Document Bundles > User-defined Catalog** node, and select **Add Schema**. The **Load Schema** dialog enables you to load a schema directly from the file system.
2. In the **File location** field, enter or browse to the location of the schema file and click **Next**.
3. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the schema. Click **Next**.

   Note If the XML schema fails validation, an error is displayed. For more information, see *XML schema and WSDL document validation* on page 104 and *XML schema and WSDL document limitations* on page 104.

4. Click the **Finish** button to import the schema into the cache.

Add WSDL documents to the cache

WSDL documents are cached automatically when you import a WSDL file to register a web service. For more information, see *Configure policies from WSDL files* on page 77.

Alternatively, you can import a WSDL document directly into the cache. To add a WSDL to the global cache, perform the following steps:

1. In the Policy Studio tree, right-click the **WSDL Document Bundles** node, and select **Add a WSDL**. The **Load WSDL** dialog enables you to load a WSDL file directly from the file system, from a URL, or from a UDDI registry.
2. Select the appropriate option and enter or browse to the location of the WSDL file in the fields provided. To retrieve the WSDL file from a UDDI registry, click the **WSDL from UDDI** option, and click the **Browse UDDI** button. This option enables you to connect to a UDDI registry and search it for a particular WSDL file. For more information on how to retrieve a WSDL file from a UDDI registry, see *Retrieve WSDL files from a UDDI registry* on page 133. Click **Next** to continue.
3. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the WSDL.
4. Click the **Finish** button to import the WSDL document into the cache.

Note If the WSDL fails validation, an error is displayed. For more information, see *XML schema and WSDL document validation* on page 104 and *XML schema and WSDL document limitations* on page 104.

If you import a WSDL document directly into the cache using the **WSDL Document Bundles** node, Policy Studio does *not* automatically generate policies and service handlers. To auto-generate policies and service handlers for a web service, you must use the **Register Web Service** option under the **APIs** > **Web Service Repository** node.

Update cached WSDL or XML schema documents

You can update a WSDL document or XML schema in the cache by using the **Resynchronize WSDL** or **Resynchronize Schema** options. This enables you to import a more recent version of a WSDL or schema directly to the global cache.

To update a WSDL, perform the following steps:

1. Expand the **WSDL Document Bundles** node.
2. Expand the document bundle for the WSDL to be updated.
3. Right-click the **Tracking from** node and select **Resynchronize WSDL**.
4. Enter the location of the latest version of the WSDL and click **Next**. This defaults to the location from which the WSDL was originally loaded.

5. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the WSDL.
6. Click **Finish** to import the new version of the WSDL document into the cache.

API Gateway compares the contents of the WSDL and any of its imported schemas to those stored in its cache. If the contents are different, it creates a new version of the WSDL.

To update an XML schema, perform the following steps:

1. Expand the **XML Schema Document Bundles > User-defined Catalog** node.
2. Expand the document bundle for the schema to be updated.
3. Right-click the **Tracking from** node and select **Resynchronize Schema**.
4. Enter the location of the latest version of the schema and click **Next**. This defaults to the location from which the schema was originally loaded.
5. In the **Retrieve and Validate** window, enter a user name and a comment for this version of the schema.
6. Click **Finish** to import the new version of the schema document into the cache.

API Gateway compares the contents of the schema and any of its imported schemas to those stored in its cache. If the contents are different, it creates a new version of the schema.

See *Update a web service* on page 95 for more information on the types of changes to a WSDL or schema that trigger creation of a new version.

Delete cached WSDL or XML schema documents

You can delete a WSDL document or XML schema in the cache by using the **Remove Tracking Details** option.

To delete a WSDL, perform the following steps:

1. Expand the **WSDL Document Bundles** node.
2. Expand the document bundle for the WSDL to be deleted.
3. Right-click the **Tracking from** node and select **Remove Tracking Details**.
4. Click **Yes** to confirm the removal.

API Gateway removes the WSDL from the cache.

To delete an XML schema from the user-defined catalog, perform the same steps under the **XML Schema Document Bundles > User-defined Catalog** node.

XML schema and WSDL document validation

XML schemas and WSDL documents are validated during import. If validation fails, an error is displayed. For example, validation fails if a schema type is referenced in a WSDL or schema, but that WSDL or schema does not contain an explicit import statement for the schema that contains the type definition for the referenced type.

Note API Gateway requires all XML schemas and WSDL documents to be present and valid at runtime, and applies very strict validation checks during import. Therefore, WSDL documents that validate successfully in other tools do not necessarily validate in API Gateway, and you might need to preprocess any XML schemas or WSDL documents to make them valid, before attempting to import them in Policy Studio.

A good starting place is to ensure that all WSDLs and their schemas are Web Service Interoperability (WS-I) compliant before attempting to import them into Policy Studio. WS-I compliance ensures maximum interoperability between different vendors' implementation of web services standards, such as SOAP and WSDL. For more information on how to test for WS-I compliance, see *Test a WSDL for WS-I compliance* on page 106.

An out-of-the-box installation of API Gateway contains a number of common XML schemas (for example, from OASIS and W3C) preloaded in the global cache. If you import a WSDL that imports any of these standard schemas, API Gateway uses the cached version of those schemas, instead of retrieving them from the web.

For example, the SOAP encoding schema is often imported into WSDLs that serialize message parts as SOAP encoding arrays. When such a WSDL is imported into Policy Studio, it recognizes that this schema already exists in the cache and does not download and import a duplicate version.

This optimization avoids unnecessary duplication of shared schema resources and reduces the overall memory footprint of the API Gateway's configuration store. For more information, see *Version and duplicate management* on page 106.

XML schema and WSDL document limitations

There are some additional limitations when importing XML schema or WSDL documents:

- **Non-SOAP bindings**
  Any HTTP and MIME bindings in the WSDL document are ignored, and only SOAP 1.1 and SOAP 1.2 bindings are imported.
- **Multiple ports for the same service**
  If the WSDL contains multiple ports for the same service (for example, a service is available over SSL and in the clear, where the URL differs, but the binding is to the same SOAP service), you can select only one of the ports for import.

  If you absolutely require both endpoints to be virtualized on API Gateway, you can create a separate service for each port in the WSDL. A distinct service handler is created for each service, which is responsible for processing requests for that service and routing them on to the endpoint URL specified in the port.

- **Schemas using the XML Schema namespace to extend element types, but not importing the namespace explicitly**
  Although some tools can work with invalid schemas like this, API Gateway requires them to be valid so it can run schema validation checks against the messages. The schema must be modified to import the namespace explicitly before you can import the schema in Policy Studio.
- **SOAP bodies with no children**
  For a SOAP binding style of `"document"`, API Gateway does not support any operation whose request SOAP Body has no child element.
- **Input-only SOAP operations**
  API Gateway does not currently support input-only SOAP operations, as these operations have no concept of a response message, and API Gateway has problems generating the **Web Service Filter** for these operations.

  When a **Web Service Filter** is generated as a result of virtualizing a web service, it handles both the request and response messages for the operations that are exposed by that service. Because input-only or notification-style SOAP operations have no concept of a response message, the **Web Service Filter** is not ideal, and a custom policy is recommended instead.
- **Output-only SOAP operations**
  API Gateway does not currently support output-only SOAP operations, as API Gateway has problems generating the **Web Service Filter** for these operations.

  Similar to input-only operations, output-only or solicit-response operations cause difficulties for a generated **Web Service Filter** and a custom policy is recommended to process this type of request instead.
- **Multiple bindings with different WS-Policy requirements**
  This results in multiple sets of security requirements that need to be configured by the user, and this is not currently supported by the **Import WSDL** wizard in Policy Studio.
- **WSDL messages that contain multiple parts**
  API Gateway does not support importing a WSDL where the `<wsdl:message>` contains multiple parts. A workaround is to change the WSDL so that each `<wsdl:message>` contains a single `<wsdl:part>` that references a schema complex type that *wraps* the message parts currently defined in each `<wsdl:message>`.
- **Schemas without the schemaLocation attribute**
  API Gateway requires schemas to import other schemas using both the namespace and schemaLocation attributes, except in the following circumstances:
  - The schemas are embedded in the WSDL. The schemaLocation attribute can be omitted from the schema import element and the schema resolver looks for the imported schemas in the WSDL itself.
  - The schemas import a schema from the system catalog (which comes preloaded with a number of common schemas, for example, the SOAP encoding schema). A schema from the system catalog can be imported into any schema using just the namespace attribute.

Version and duplicate management

API Gateway manages the resources associated with any WSDL documents or XML schema documents stored in the global cache. This includes the WSDL or XML schema documents themselves, and any WSDL or XML schemas imported by those documents.

When a new resource is being added to the global cache (for example, when you import a WSDL or add a new XML schema), API Gateway compares each resource against the existing resources in the cache. API Gateway tracks two items for each resource:

- The location of the resource
- The contents of the resource at that location

By tracking these two items, API Gateway can identify when a resource is a new version of an existing resource at the same location, or when a resource is the same as an existing resource at a different location. In both cases a new version of the resource is created.

This means that resources that are shared by multiple web services are not duplicated in the global cache, and that web services can be updated easily if the WSDL defining the back-end web service changes. For more information on updating a web service, see *Update a web service* on page 95.

A common example of this is where a vendor implements multiple web services that share common data structures, for example, an object that stores employee details. When the WSDL and schema files for these services are generated, they typically all import a common schema file that defines the employee details data structure.

On importing these WSDLs into Policy Studio, API Gateway recognizes that the services all share a common `employees` schema, and avoids importing multiple copies of the schema into the cache. Instead, each imported web service *points* at the common schema.

Validate messages against XML schemas

The **Schema Validation** filter is used to validate XML messages against schemas stored in the cache. It can be configured to validate messages against schemas stored in the cache, and also against schemas embedded within WSDL stored in the cache. This filter is found in the **Content Filtering** category of filters in Policy Studio. For more information on configuring this filter, see "Schema validation" in the *API Gateway Policy Developer Filter Reference*.

Test a WSDL for WS-I compliance

Before importing a WSDL file, you can check it for compliance with the WS-I Basic Profile. The Basic Profile consists of a set of assertions and guidelines on how to ensure maximum interoperability between different implementations of web services. For example, there are recommendations on what style of SOAP to use (`document/literal`), how schema information is included in WSDL files, and how message parts are defined to avoid ambiguity for consumers of WSDL files.

Policy Studio uses the Java version of the WS-I testing tools to test imported WSDL files for compliance with the recommendations in the Basic Profile. A report is generated showing which recommendations have passed and which have failed. While you can still import a WSDL file that does not comply with the Basic Profile, there is no certainty that consumers of the web service can use it without encountering problems.

Note Before you run the WS-I compliance test, you must ensure that the Java version of the WS-I testing tools is installed on the machine on which Policy Studio is running. You can download these tools from www.ws-i.org.

To configure the location of the WS-I testing tools, select **Window** > **Preferences** from the Policy Studio main menu. In the **Preferences** dialog, select **WS-I Settings**, and browse to the location of the WS-I testing tools. You must specify the *full path* to these tools (for example, `C:\Program Files\WSI_Test_Java_Final_1.1\wsi-test-tools`). For more details on configuring WS-I settings, see *Policy Studio preferences* on page 180.

Run the WS-I compliance test

To run the WS-I compliance test on a WSDL file, perform the following steps:

1. Select **Tools** > **Run WS-I Compliance Test** from the Policy Studio main menu.
2. In the **Run WS-I Compliance Test** dialog, browse to the **WSDL File** or specify the **WSDL URL**.
3. Click **OK**. The WS-I analysis tools run in the background in Policy Studio.

The results of the compliance test are displayed in your browser in a **WS-I Profile Conformance Report**. The overall result of the compliance test is displayed in the **Summary**. The results of the WS-I compliance tests are grouped by type in the **Artifact:description** section. For example, you can access details for a specific port type, operation, or message by clicking the link in the **Entry List** table. Each **Entry** displays the results for the relevant WS-I test assertions.

Manage JSON schemas

JSON schemas define the structure of JSON data. For more details on JSON schemas, see http://www.json-schema.org.

You can manage JSON schemas and schema containers under the **Resources > JSON Schemas** node in the Policy Studio tree.

Add JSON schema

To add a JSON schema, right-click the **JSON Schemas** node and select **Add Schema**. Browse to the location on the JSON file on disk and click **Open**.

You can also view or edit existing schemas. To view a schema, double-click it in the tree. The JSON is displayed on the right. You can edit the JSON in this view. Click **Save JSON** in the toolbar to save the changes.

Add JSON schema container

Schema containers are used to group related schemas. To add a JSON schema container, right-click the **JSON Schemas** node and select **Add Schema Container**.

Manage data maps

Data maps enable you to define how to map XML and JSON messages to other XML and JSON message formats. Mappings can be from XML to XML, XML to JSON, JSON to JSON, or JSON to XML.

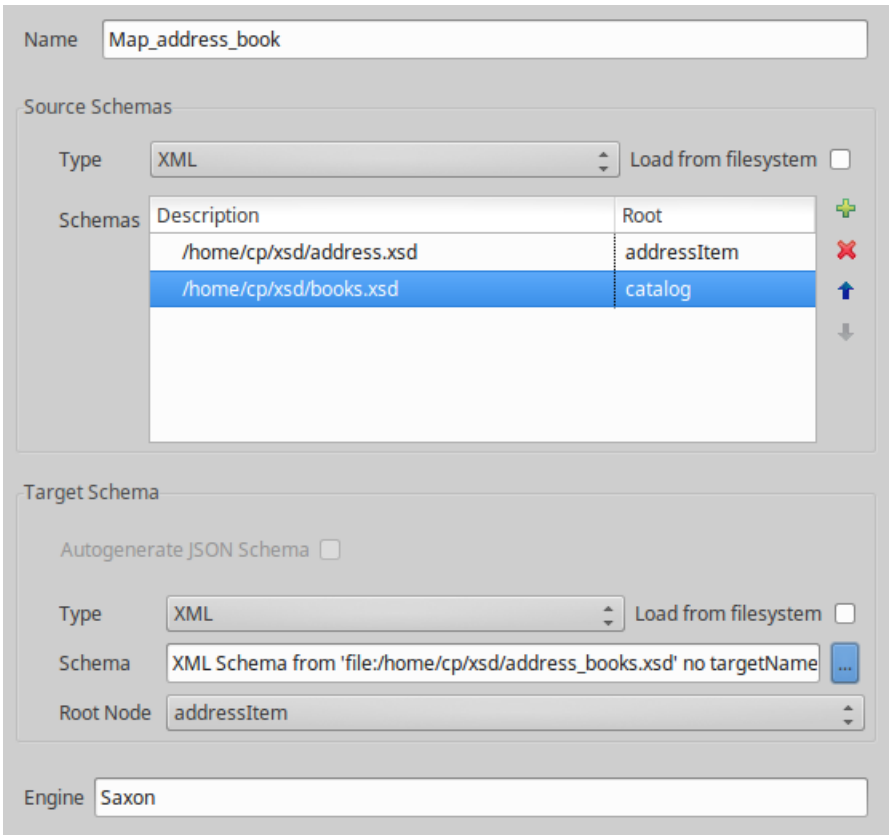
To create a data map, you select one or more source (input) schemas and one target (output) schema. After creating the data map, you can use the Data Map Editor to define the mapping between the source and target schemas.

To manage data maps, use the **Resources > Data Maps** node in the Policy Studio tree.

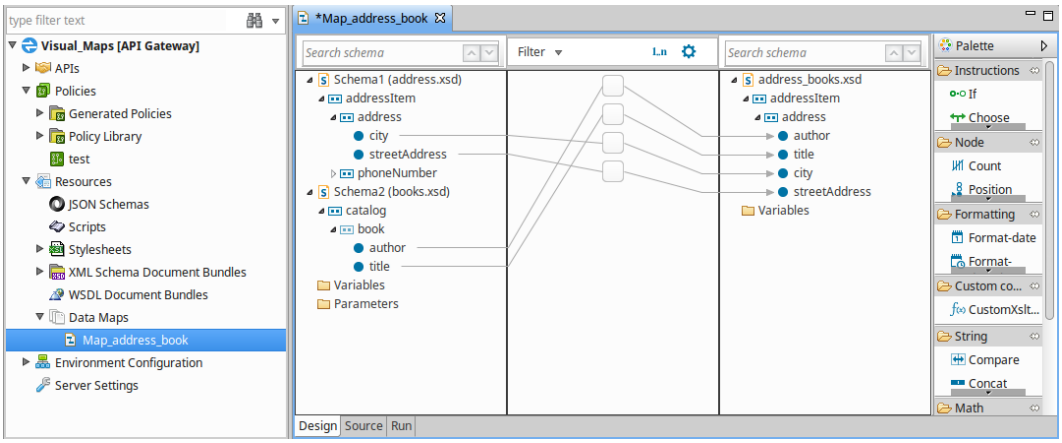
Add data map

To define a new data map, follow these steps:

1. Right-click the **Data Maps** node and select **Add New Data Map**. The Data Map dialog is displayed.



2. Enter a unique **Name** for the data map.
3. In the Source Schema Details section, click the Add button to select an input schema. You can add multiple input schemas, but all input schemas must be of the same type (either JSON or XML). You can also delete schemas and reorder schemas in the list. For more details on input schema settings, see *Schema settings* on page 110.
4. In the Target Schema Details section, click the Browse button to select an output schema. For more details on output schema settings, see *Schema settings* on page 110.
5. The read-only field **Engine** displays the XSLT processor used by the data map (for example, `Saxon`).
6. When you have completed the fields, click **OK** to open the map for editing in the Data Map Editor Design view. The following example shows a map with two input schemas.



For more information on Visual Mapper and the Data Map Editor, see the *API Gateway Visual Mapper User Guide*.

Schema settings

The Source Schema Details and the Target Schema Details sections share the following common fields:

- **Type**: Select the type of the schema, XML or JSON. Choose XML if the schema is defined by an XSD, or choose JSON if the schema is defined as a JSON schema. You can choose one of four possible schema types depending on the required mapping type:

Mapping Type	Source Schema Type	Target Schema Type
XML to XML	XML	XML
JSON to JSON	JSON	JSON
XML to JSON	XML	JSON
JSON to XML	JSON	XML

Note If you select multiple input schemas for a data map, the input schemas must be of the same type (either JSON or XML).

- Select the **Load from filesystem** check box if the schema definition is located in a file. If the type is XML, it is typical to select an XSD file. If the type is JSON, it is typical to select a JSON file that contains the JSON schema definition. Deselect this option to use an XSD or JSON schema that you have previously imported into API Gateway (for example, under **Resources > JSON Schemas**, **Resources > XML Schema Document Bundles**, or **Resources > WSDL Document Bundles**).
- **Schema**: For more information on schemas, see *Manage JSON schemas* on page 107 and *Manage WSDL and XML schema documents* on page 98.

- If you selected the **Load from filesystem** option you can select the schema file on the file system. If the **Type** is XML, you can select an XSD file containing the schema definition. If the **Type** is JSON, you can select a JSON file containing the JSON schema definition.
- If you did not select the **Load from filesystem** option, you can select from the schemas that you previously imported into API Gateway (for example, under **Resources > JSON Schemas**, **Resources > XML Schema Document Bundles**, or **Resources > WSDL Document Bundles**).

- **Root Node**: The root node applies to XML schemas only. If there is more than one root element in the XML schema, you are presented with a list of all elements that exist at the root node level. You must select one of these elements for the mapping. If there is only one root element, this root element is listed and selected automatically.

Data map options

To open a data map for editing in the Data Map Editor, select the data map node under **Resources > Data Maps** in the Policy Studio tree. The Data Map Editor displays the content of the map and you can make changes. Any changes made to the map must be saved and redeployed. For more information on Visual Mapper and the Data Map Editor, see the *API Gateway Visual Mapper User Guide*.

Note The schemas selected for the data map typically cannot change. To use a different schema, you must create a new data map. However, there is one exception to this: you can modify a JSON schema stored in **Resources > JSON Schemas** after you have created an associated data map. For more information, see *Update previously imported JSON schemas on page 112*.

TBC: Is above note still correct?

You can right-click on an existing data map node in the Policy Studio tree, and select one of the following options:

- **Edit** – Change the name of the data map or select a different version of the XML schema.
- **Copy** – Make a copy of the data map. Right-click the **Data Maps** node and select **Paste** to paste a copy of the map. A copy of the data map is created and it is opened in the Data Map Editor for editing.
- **Export Data Map** – Export the data map as API Gateway configuration data to an XML file. You can use **File > Import > Import Configuration Fragment** to reimport this data map at a later stage.
- **Edit Source Schema** – Allows you to select and edit a source schema.
- **Edit Target Schema** – Allows you to edit the target schema.
- **Delete** – Delete the data map. If policies exist that use this data map, you must delete them first.
- **Show all references** – Shows all objects that refer to this data map. Typically, you will see any policies associated with the data map.

TBC: Any changes to above options? Are all above options still available? Should we even document them?

Update previously imported JSON schemas

TBC: Is this section still correct?

You can update a previously imported JSON schema (for example, if you add a new field to the JSON schema, the field will be visible when the associated Data Map is reloaded).

To delete or rename a field in the JSON schema, remove any references to this field (for example, links in the associated Data Map) before you change it in the schema. If you attempt to delete the previously imported JSON schema, a message is displayed that states there are Data Maps that depend on the JSON schema.

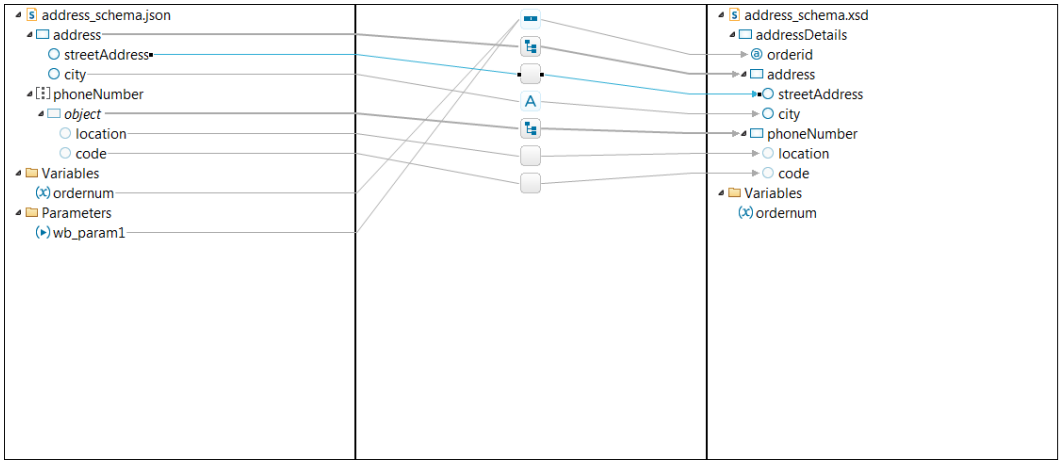
Use a data map in a policy

TBC: Is this example still valid?

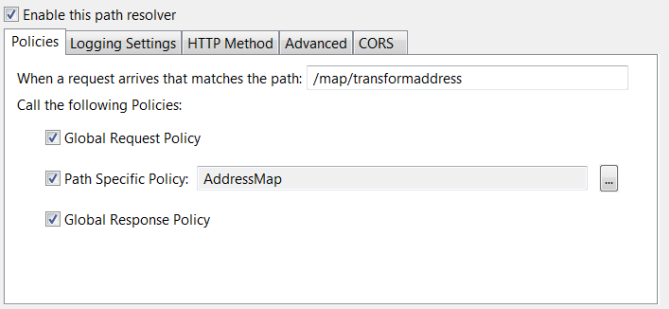
To use a data map in a policy, add an **Execute Data Map** filter. For more details, see "Transform with data map" in the *API Gateway Policy Developer Filter Reference*.

A simple example is described in the following steps:

1. Define a data map called `addressjsonxmlmap` that maps a JSON address schema to an XML address schema.



2. Create a policy called `AddressMap` containing an **Execute Data Map** filter that executes the `addressjsonxmlmap` data map.
3. Add a relative path for the `AddressMap` policy so that all requests received by API Gateway on the path `/maps/transformaddress` are processed by the `AddressMap` policy. On the **HTTP Method** tab of the path resolver, set the method to `POST`.



4. Use a REST client such as POSTMAN to send a POST request to API Gateway containing an address in JSON format. The response should be the address mapped to XML format according to the data map you defined.

For example, send a POST request to the URL:

```
http://localhost:8080/map/transformaddress
```

If the body of the request contains the following JSON formatted address:

```
{
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York"
  },
  "phoneNumber": [
    {
      "location": "home",
      "code": 44
    }
  ]
}
```

The mapping defined by the `addressjsonxmlmap` data map results in the following response (an XML formatted address):

```
<?xml version="1.0" encoding="UTF-8"?>
<addressDetails orderid="12345POST">
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>NEW YORK</city>
    </address>
    <phoneNumber>
        <location>home</location>
        <code>44</code>
    </phoneNumber>
</addressDetails>
```

Expose a web service as a REST API

Overview

You can import a WSDL file into Policy Studio, and invoke it from a policy instead of exposing it to a client.

For example, in a SOAP to REST use case, the web service is registered in Policy Studio by importing a WSDL file into the web service repository. A REST API is then defined in Policy Studio, which calls a policy to implement the API, and in turn, this policy invokes the web service.

Summary of steps

The steps involved in exposing a SOAP web service as a REST API are summarized as follows:

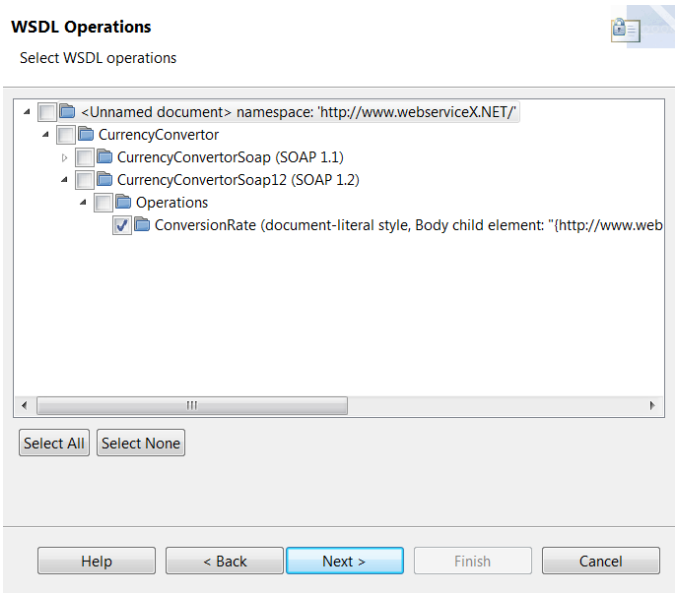
1. *Virtualize a SOAP web service* on page 114
2. *Define a REST API* on page 115
3. *Route REST requests through the virtualized SOAP service* on page 116
4. *Test the REST to SOAP mapping* on page 120

Virtualize a SOAP web service

To expose a virtualized version of a SOAP web service on API Gateway, import a WSDL file describing the web service into the web service repository.

The following figure shows importing the WSDL for a currency conversion SOAP web service.

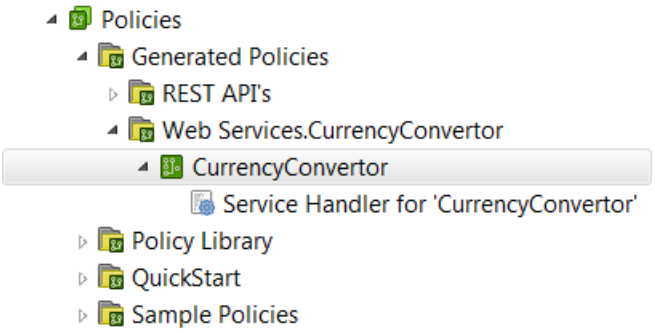
Note This is an example for demonstration purposes only. Adapt the following steps to suit your web service.



For more information on using the **Import WSDL** wizard, see *Import a WSDL file* on page 79.

When you register a web service in Policy Studio, service handlers and policies are autogenerated.

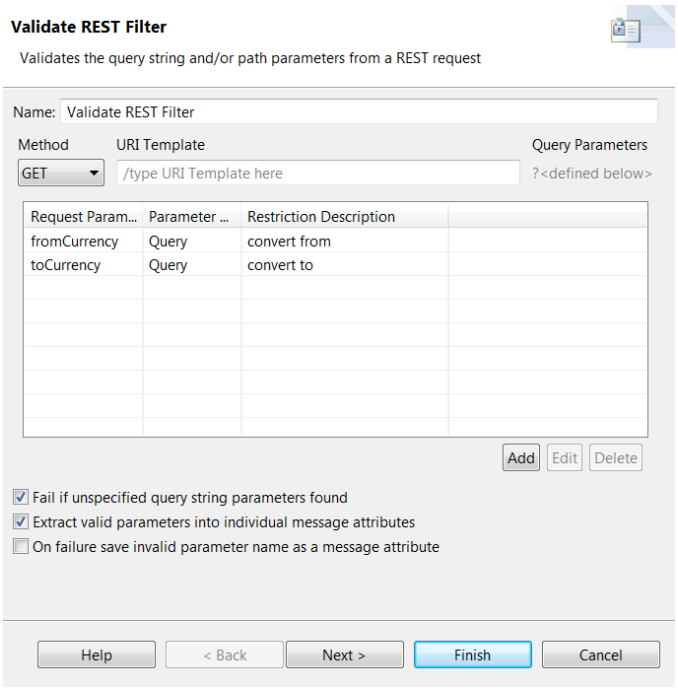
The following figure shows the generated policies for the currency conversion service.



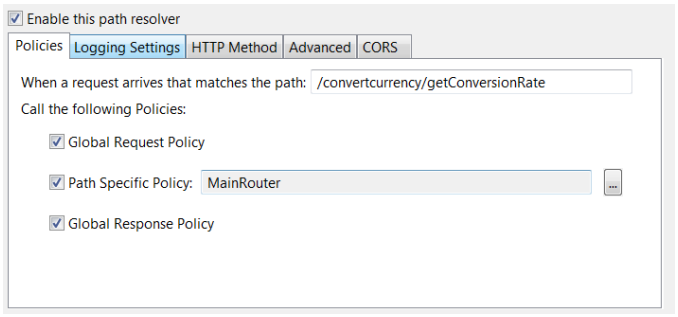
Define a REST API

The next step is to define a REST API for a currency conversion service as follows:

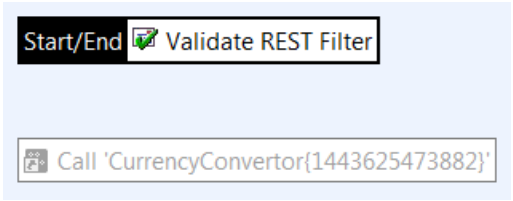
1. Create a new policy container called `CurrencyConversion`.
2. Within this policy container create a new policy called `MainRouter`.
3. Add a **Policy Shortcut** filter to the `MainRouter` policy, and set it to call the `CurrencyConvertor` policy that was autogenerated when you imported the WSDL for the web service.
4. Add a **Validate REST** filter to the `MainRouter` policy. Set the **Method** to `GET`, add two request parameters called `fromCurrency` and `toCurrency`, and select the **Extract valid parameters into individual message attributes** check box:



5. Add a relative path for the `MainRouter` policy so that all REST requests received by API Gateway on the path `/convertcurrency/getConversionRate` are processed by the `MainRouter` policy. On the **HTTP Method** tab of the path resolver, set the method to `GET`.



At this stage, the MainRouter policy should look like this:



Route REST requests through the virtualized SOAP service

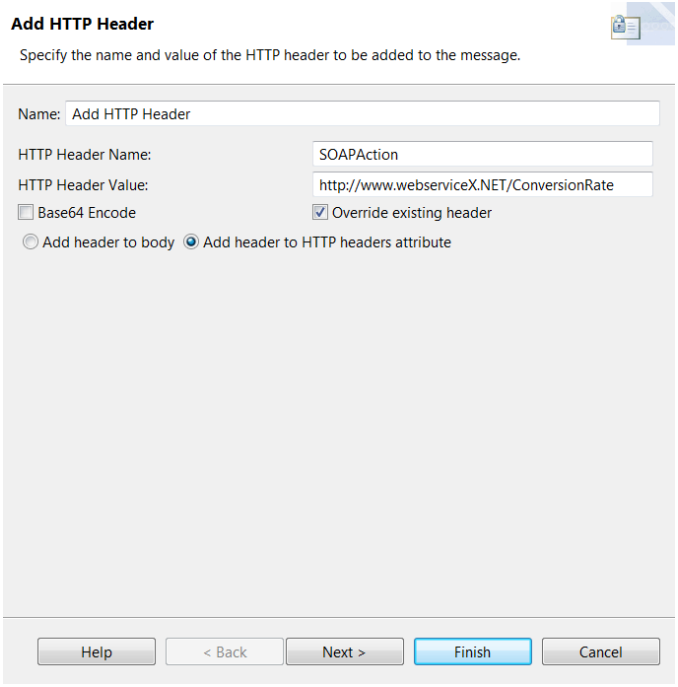
To route REST requests through the virtualized SOAP service, perform the following sequence of tasks:

1. *Create a request processing policy* on page 117
2. *Create a response processing policy* on page 119

Create a request processing policy

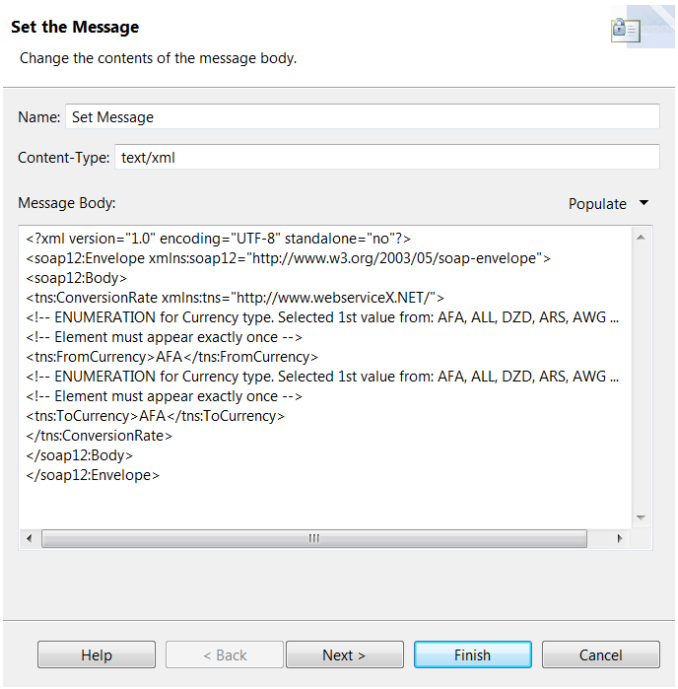
First, create a dedicated request processing policy to create the SOAP request message body to send to the SOAP service:

1. Create a request processing policy called `GetCurrencyConvertorRequest`.
2. Add a **Set HTTP verb** filter to the policy and enter `POST` in the **HTTP Verb** field.
3. Add an **Add HTTP header** filter to the policy with the following settings:



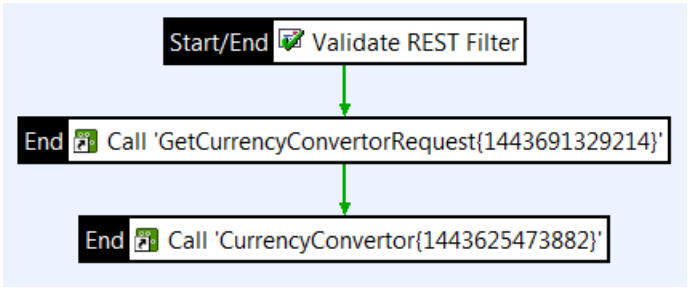
4. Add a **Set Message** filter to the policy and enter `text/xml` as the **Content-Type**.

   Select **From web service operation** from the **Populate** menu and select the `ConversionRate` operation from the currency conversion web service. This populates the contents of the message body.

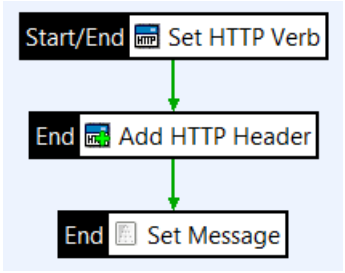


You need to replace the currencies in the message body with the currencies from the incoming REST request. To enable the autocompletion mechanism in the **Set Message** filter, which will allow you to insert the `fromCurrency` and `toCurrency` selector strings in the message body, you must first connect up the filters in the `MainRouter` and `GetCurrencyConvertorRequest` policies.

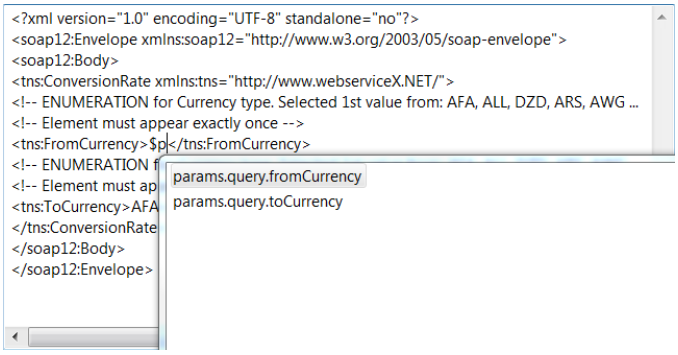
5. Add a **Policy Shortcut** filter to the `MainRouter` policy, and set it to call the `GetCurrencyConvertorRequest` policy.
6. Connect the filters in the `MainRouter` policy as follows:



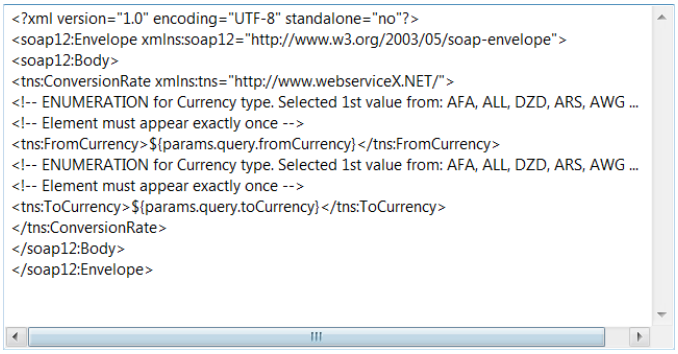
7. Connect the filters in the `GetCurrencyConvertorRequest` policy as follows:



8. Edit the **Set Message** filter. Select the currency type in the message body and start typing to see matching selectors.



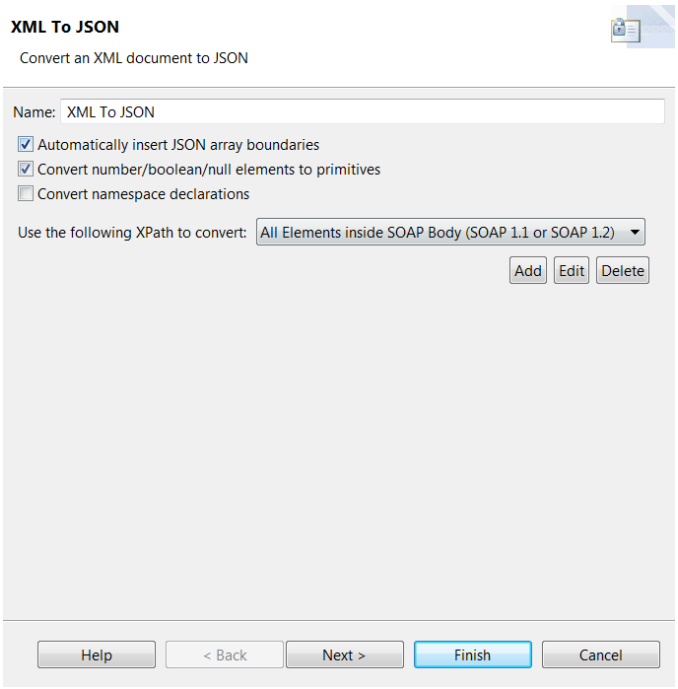
9. Insert the selectors `$params.query.fromCurrency` and `$params.query.toCurrency` in place of the currency types.



Create a response processing policy

Next, create a response processing policy to convert the XML returned from the SOAP web service to JSON format:

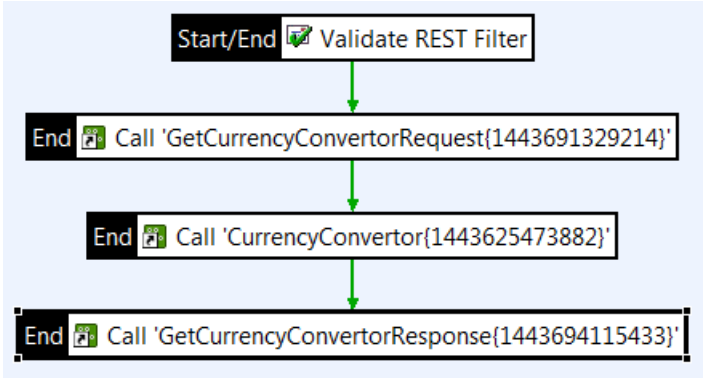
1. Create a response processing policy called `GetCurrencyConvertorResponse`.
2. Add an **XML to JSON** filter to the policy. Configure it to extract the SOAP Body content first and remove any namespaces:



3. The GetCurrencyConvertorResponse policy should now look like this:

4. Add another **Policy Shortcut** filter to the `MainRouter` policy, and set it to call the `GetCurrencyConvertorResponse` policy.

   At this stage, the MainRouter policy should look like this:



Test the REST to SOAP mapping

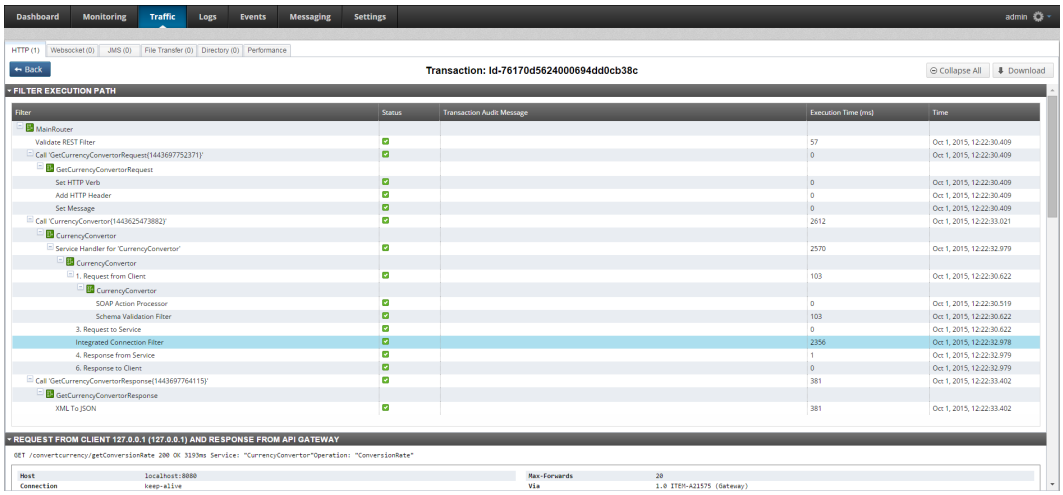
To test the REST to SOAP mapping, deploy the configuration on the API Gateway and send a REST request from a REST client. For example, to get the conversion rate for EUR to USD, send a request to the URL:

```
http://localhost:8080/convertcurrency/getConversionRate?fromCurrency=EUR&toCurrency=USD
```

The following is an example JSON response:

```
{
    "ConversionRateResponse": {
        "ConversionRateResult": 1.1194
    }
}
```

You can use API Gateway Manager to view the filter execution path for the REST request, and to examine the request from the client and the response from API Gateway, and the request from API Gateway and the response from the web service.



For more information on API Gateway Manager, see the *API Gateway Administrator Guide*.

Develop REST APIs in Policy Studio

Policy developers can use the REST API development wizard in Policy Studio to create new REST APIs based on existing back-end REST and non-REST APIs in an iterative development cycle. For example, this includes exposing a SOAP web service, or a cloud-based application as a REST API. Creating new REST APIs using Policy Studio is known as *API development*. You must specify a custom routing policy for REST API methods. You can specify optional policies for request and response processing as required.

The **REST API Repository** node in the Policy Studio tree stores information about REST APIs created using the REST API development wizard. When new REST APIs are added in the **REST API Repository**, they can then be registered in API Manager for consumption by API consumers and administration by API administrators.

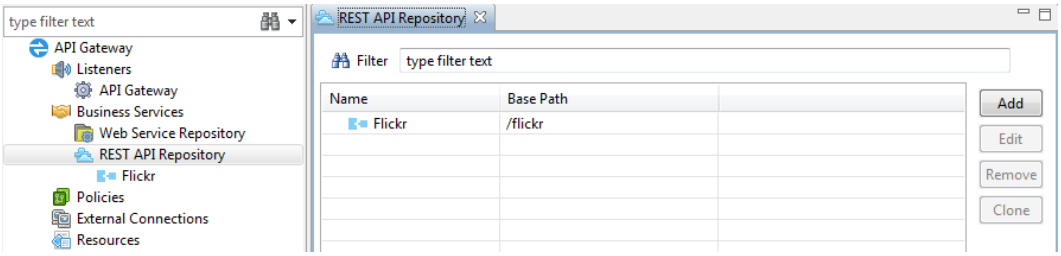
APIs created using the REST API development wizard are registered by importing as a back-end API into API Manager. Registered APIs are governed by the API Gateway using policies defined in the client registry. API administrators can use API Manager to manage registered APIs, and API consumers can use API Manager to consume registered APIs in their client applications.

Note You can use Policy Studio to create new REST APIs that route to custom policies. You must use API Manager to register a front-end REST API that routes to the destination REST API. For more details, see the *API Manager User Guide*.

Virtualized REST APIs

The **REST API Repository** is displayed under the **APIs** node in the Policy Studio tree. When a REST API is added to this repository, it becomes *virtualized* by the API Gateway. A virtualized REST API is assigned a path that gives it a namespace in a running API Gateway instance (for example, `/flickr/v1`). This means that all of its methods reside under a URL with that path (for example, `/flickr/v1/photos`). You can view the list of virtualized REST APIs by selecting the **APIs** node in the tree.

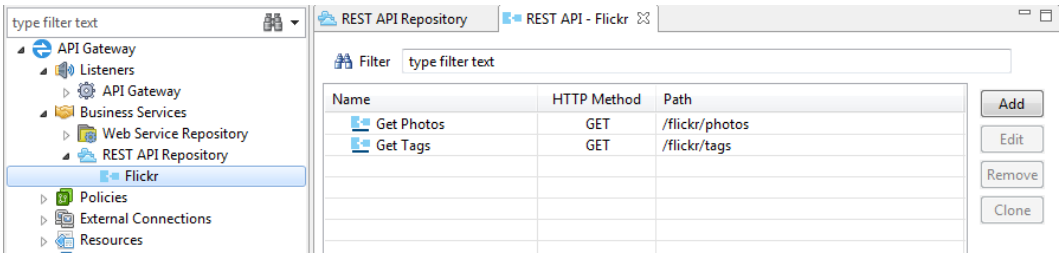
The following example shows a virtualized Flickr REST API:



Note Because the REST API encompasses an inbound listener and resolver in a running API Gateway instance, the paths associated with the API are also displayed in the tree (for example, under **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services** > **Paths**). These REST API paths are read-only under the **Listeners** node. You can create or update these paths as described in *Virtualize a REST API* on page 123.

REST API methods

Each REST API consists of zero or more REST API methods. Each method can provide custom polices for request, routing, and response processing. You can view the list of methods for a virtualized REST API by clicking a specific API node under **REST API Repository**. The following example shows API methods for a virtualized **Flickr** API:



Virtualize a REST API

To virtualize a REST API in the repository, perform the steps described in this section.

Define the REST API

Perform the following steps to define the REST API:

1. Right-click the **REST API Repository** node, and select **Add REST API** to launch the wizard.
2. Enter a **Name** for the REST API (for example, `Flickr API`).
3. Enter a short **Summary** of the API functionality.
4. For a detailed **Description**, select to **Manually enter a description** in the text box. Alternatively, select to **Specify markdown file location**. The default location is:
5. Enter the API **Version**. Defaults to `1.0`.

   ```
   ${environment.VINSTDIR}/docs/REST_API_NAME/REST_API_NAME.md
   ```

   For details on writing documentation with Markdown, see http://daringfireball.net/projects/markdown/.
6. Click **Next**.

Specify exposure settings

Perform the following steps to configure how the REST API is exposed:

1. Specify the details for the inbound path that accepts requests:
   - **Listening on HTTP**: Select the HTTP Services group on which the API listens. Defaults to **Default Services**.
   - **Virtual Host**: Click the browse button to select a virtual host in the dialog. If no virtual host has already been configured, right-click the HTTP service node (for example, **Default Services**), and select **Add a Virtual Host**. For more details, see *Configure virtual hosts* on page 315.
   - **Base Path**: Enter the base path prefix for the API (for example, `/flickr/v1`). This is the path that the API Gateway listens on for messages for the REST API. This path gives the API a namespace in the running API Gateway instance, under which its methods can reside (for example, `/flickr/v1/getPhoto`).
2. Click **Finish**.

Edit the REST API

When the REST API has been virtualized in the repository, it is displayed when you select the **REST API Repository** node. At any stage, you can select the REST API in the panel on the right, and click **Edit** to edit any of its configuration settings. Similarly, click **Remove** to remove the API from the repository.

Virtualize a REST API method

To virtualize a REST API method in the repository, perform the steps described in this section.

Define the method

Perform the following steps to define the REST API method:

1. Right-click a configured REST API node, and select **Add API Method** to launch the wizard.
2. Enter a **Name** for the REST API method (for example, `Photo Search`).
3. Enter a short **Summary** of the method functionality.
4. For a more detailed **Description**, select to **Manually enter a description** in the text box. Alternatively, select to **Specify markdown file location**. The default location is:

   ```
   ${environment.VINSTDIR}/docs/REST_API_NAME/REST_API_METHOD_NAME.md
   ```

   For details on writing documentation with Markdown, see http://daringfireball.net/projects/markdown/.
5. Click **Next**.

Specify exposure settings

Perform the following steps to configure the exposure settings:

1. Specify the details for the inbound path that accepts requests:
   - **HTTP Method**: Select the HTTP method (defaults to **GET**).
   - **Using path**: Enter the path for the REST API method (for example, `/getPhoto`).

Note You can also specify path parameters (for example, an inbound path of `/getPhoto/{userID}`, where *userID* is a variable part of the path). When you specify an inbound path parameter, the parameter (in this case, `userID`) is automatically added in the **Inbound parameters** table below.

- **Exposed on**: Enter the base path that the method is exposed on. Defaults to the API base path (for example, `/flickr/v1`). For more details, see *Specify exposure settings* on page 123.

- **Resolved using**: Select whether the path is resolved using **Exact path match** (default) or **Longest path match**. For more details, see *Configure relative paths* on page 293.

2. Under **Inbound parameters**, click **Add** to specify a path parameter exposed by the method, and enter values for the following settings:
   - **Name**: Enter the parameter name (for example, `userID`).
   - **Description**: Enter a short parameter description.
   - **Param Type**: Select the parameter type from the list. Defaults to **query**.
   - **Data Type**: Select the data type from the list. Defaults to `string`.
   - Select whether the parameter is **Required** (unselected by default).
   - Select whether to **Allow Multiple** parameters (unselected by default)

   The inbound parameters entered in this table are displayed on the API Manager page for this REST API method. For more details, see *Example inbound parameters* on page 126.
3. Select whether the inbound request should **Fail if unspecified parameters are found** (selected by default).

   Checking for unspecified parameters is only supported for `Query` and `Form` parameters. It is not supported for `Header` parameters.
4. Under **Content Types**, select whether **I wish to check inbound content types based on the following settings**.

   If selected, choose whether to **Allow Content Types** or **Deny Content Types**, and select the types that you wish to allow or deny.
5. Under **Error Response Codes**, click **Add** to specify an error response exposed by the method, and enter values for the following settings:
   - **HTTP Response Code**: Select or enter the response code name (for example, `400`).
   - **Reason**: Select or enter the response code reason (for example, `Bad Request`).

   The response codes entered in this table are displayed on the API Manager page for this REST API method.
6. Click **Next**.

Configure policies and monitoring settings

You can specify the following policy and monitoring settings for the API method:

1. Select a required **Routing** policy, which configures how inbound requests are routed.
2. Click **Next** to select **Monitoring** options to configure how the API method is displayed in the API Gateway Manager and API Gateway Analytics consoles. Alternatively, click **Finish**.

   The monitoring options are as follows:
   - **Monitor API Service usage**:
     Specifies whether to store message metrics for this API service. This is selected by

default.

- **Message attribute**:
  Enter the message attribute to use to identify authenticated clients. The default `authentication.subject.id` attribute stores the identifier of the authenticated user (for example, the user name or X.509 Distinguished Name).

3. Click **Finish**.

Edit the REST API method

When the API method has been virtualized in the repository, it is displayed under the **REST API Repository** node. At any stage, you can select the REST API method in the panel on the right, and click **Edit** to edit any of its configuration settings. Similarly, click **Remove** to delete the API method from the repository.

Example inbound parameters

The REST API method wizard enables you to virtualize REST API methods in the **REST API Repository**, and to specify any inbound path parameters (for details, see *Specify exposure settings* on page 124).

For example, given a virtualized REST API such as the Yahoo Finance API, you can use the REST API method wizard to specify inbound stock quote URL path parameters, which can then be mapped to specific query string parameters. This enables you to expose an inbound path such as `/stock/quote/{symbol}`, and to map `{symbol}` to the `s` parameter in the query string. You can then make parameterized API calls such as the following at runtime:

```
finance.yahoo.com/d/quotes.csv?s=symbol&f=sb2b3jk
```

This call returns a stock quote such as the following at runtime:

```
AMZN    272.99  270.03  185.51  284.72
```

Supported parameter mappings

The REST API parameter mappings supported by the API Gateway are as follows:

Parameter type	Mapping
Query string	• Query string to query string • Query string to form body • Query string to path • Query string to header

Parameter type	Mapping
Path	• Path to query string • Path to form body • Path to path • Path to header
Form-encoded body	• Form body to query string • Form body to form body • Form body to path • Form body to header
Header	• Header to query string • Header to form body • Header to path • Header to header

You can configure any required mappings using request, routing, or response policies. For more details, see *Configure policies and monitoring settings* on page 125. For example, you can use a **Set Header** filter to map to a header parameter, or a **Scripting** or **Set Message** filter to map to a path or query parameter.

How to access parameter values at runtime

When inbound parameters are specified in your REST API, you can access their values on the API Gateway message whiteboard at runtime using the following selectors in your policies:

- `${params.path.param_name}`
- `${params.query.param_name}`
- `${params.form.param_name}`
- `${params.header.param_name}`

The API Gateway parses the input parameters and auto-generates the message attributes. The configured mappings take these message attributes and use them for the output parameter values (query, form, path, or header).

JSON messages

When the message content is JSON, you can also access fields in the message body using `${params.body.field}`. For example, given the following JSON message content:

```
{
```

```
    "id":"28a384e7-0b13-4679-ae1c-0c63666d3d8e"
    "activities": ["running", "swimming", "cycling"],
    "subObject":
      "foo": "bar",
      "hello": "world",
      "yetAnotherObject": {
         "array": ["a", "b", "c"],
         "val": "greetings from yetAnotherObject"
       }
     }
}
```

You can use the following selectors to access the JSON message body content in your policy at runtime:

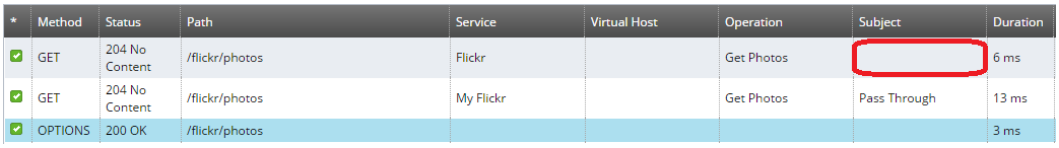
Selector	Value
`${params.body.id}`	`28a384e7-0b13-4679-ae1c-0c63666d3d8e`
`${params.body.activities[1]}`	`swimming`
`${params.body.subObject.hello}`	`world`
`${params.body.subObject.yetAnotherObject.array[0]}`	`a`

For more details, see *Select configuration values at runtime* on page 421.

Monitor APIs in API Gateway Manager

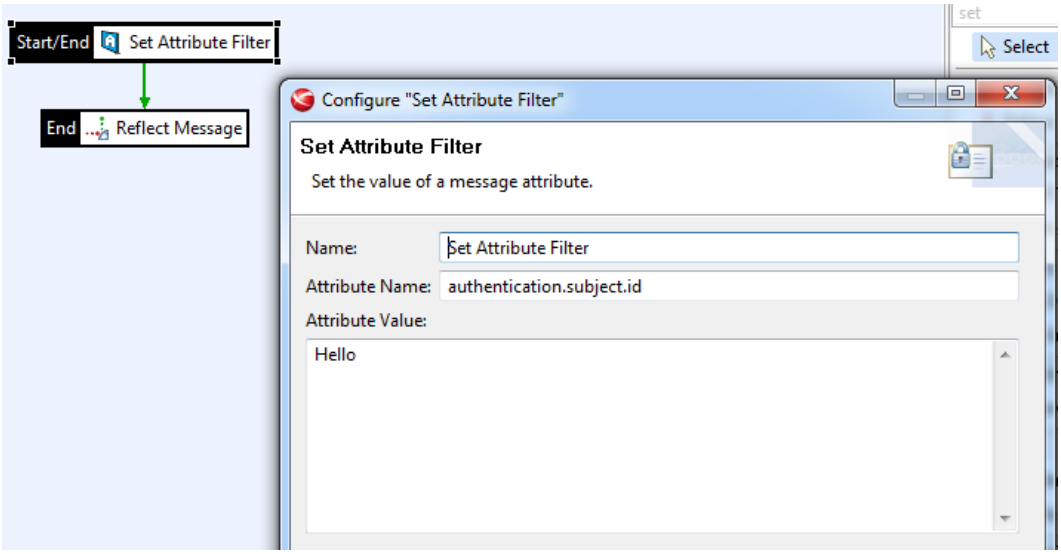
When you create an API in Policy Studio and register it in API Manager, you can use the API Gateway Manager web console to monitor the API at runtime.

When invoking a REST API developed in Policy Studio, by default, the authenticated subject displayed on the **Traffic** tab is blank:

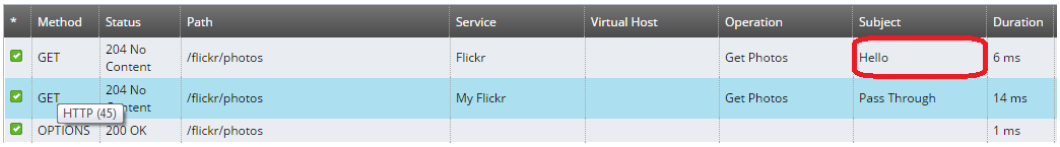


To override the default behavior, you must set the `${authentication.subject.id}` message attribute in one of the policies configured for the API (request, routing, or response).

The following shows example shows how to configure this using a **Set Attribute** filter:



When configured in a policy, the subject is displayed in API Gateway Manager as follows:



For details on using API Gateway Manager, see the *API Gateway Administrator Guide*.

Connect to a UDDI registry

Overview

This topic explains how to configure a connection to a UDDI registry in the **Registry Connection Details** dialog. It explains how to configure connections to UDDI v2 and UDDI v3 registries, and how to secure a connection over SSL.

Configure a registry connection

Configure the following fields in the **Registry Connection Details** dialog:

Registry Name:
Enter the display name for the UDDI registry.

UDDI v2:
Select this option to use UDDI v2.

UDDI v3:
Select this option to use UDDI v3.

Inquiry URL:
Enter the URL on which to search the UDDI registry (for example, `http://HOSTNAME:PORT/uddi/inquiry`).

Publish URL:
Enter the URL on which to publish to the UDDI registry, if required (for example, `http://HOSTNAME:PORT/uddi/publishing`).

Security URL (UDDI v3):
For UDDI v3 only, enter the URL for the security service, if required (for example, `http://HOSTNAME:PORT/uddi/security.wsdl`).

Note For UDDI v3, the **Inquiry URL**, **Publish URL**, and **Security URL** specify the URLs of the WSDL for the inquiry, publishing, and security web services that the registry exposes. These fields can use the same URL if the WSDL for each service is at the same URL.

For example, a WSDL file at `http://HOSTNAME:PORT/uddi/uddi_v3_registry.wsdl` can contain three URLs:

- `http://HOSTNAME:PORT/uddi/inquiry`
- `http://HOSTNAME:PORT/uddi/publishing`
- `http://HOSTNAME:PORT/uddi/security`

These are the service endpoint URLs that Policy Studio uses to browse and publish to the registry. These URLs are not set in the connection dialog, but discovered using the WSDL. However, for UDDI v2, WSDL is *not* used to discover the service endpoints, so you must enter these URLs directly in the connection dialog.

Max Rows:
Enter the maximum number of entries returned by a search. Defaults to `20`.

Registry Authentication:
The registry authentication settings are as follows:

- **Type**

  This optional field applies to UDDI v2 only. The only supported authentication type is `UDDI_GET_AUTHTOKEN`.

- **Username**

  Enter the user name required to authenticate to the registry, if required.

- **Password**

  Enter the password for this user, if required.

The user name and password apply to UDDI v2 and v3. These are generally required for publishing, but depend on the configuration on the registry side.

HTTP Proxy:
The HTTP proxy settings apply to UDDI v2 and v3:

- **Proxy Host**

  If the UDDI registry location entered above requires a connection to be made through an HTTP proxy, enter the host name of the proxy.

- **Proxy Port**

  If a proxy is required, enter the port on which the proxy server is listening.

- **Username**

  If the proxy has been configured to only accept authenticated requests, Policy Studio sends this user name and password to the proxy using HTTP Basic authentication.

- **Password**

  Enter the password to use with the user name specified in the field above.

HTTPS Proxy:
The HTTPS proxy settings apply to UDDI v2 and v3:

- **SSL Proxy Host**

  If the **Inquiry URL** or **Publish URL** uses the HTTPS protocol, the SSL proxy host entered is used instead of the HTTP proxy entered above. In this case, the HTTP proxy settings are not used.

- **Proxy Port**

  Enter the port that the SSL proxy is listening on.

Secure a connection to a UDDI registry

You can communicate with the UDDI registry over SSL. All communication may not need to be over SSL (for example, you may wish publish over SSL, and perform inquiry calls without SSL). For UDDI v2 and v3, you can use a mix of `http` and `https` URLs for WSDL and service address locations.

You can specify some or all of the **Inquiry URL**, **Publish URL**, and **Security URL** settings as `https` URLs. For example, with UDDI v3, you could use a single URL like the following:

```
https://HOSTNAME:PORT/uddi/wsdl/uddi_v3_registry.wsdl
```

If any URLs (WSDL or service address location) use `https` , you must configure the Policy Studio so that it trusts the registry SSL certificate.

Configure Policy Studio to trust a registry certificate

For an SSL connection, you must configure the registry server certificate as a trusted certificate. Assuming mutual authentication is not required, the simplest way to configure an SSL connection between Policy Studio and UDDI registry is to add the registry certificate to the Policy Studio default

truststore (the `cacerts` file). You can do this by performing the following steps in Policy Studio:

1. Select the **Environment Configuration** > **Certificates and Keys** > **Certificates** node in the Policy Studio tree.
2. Click **Create/Import**, and click **Import Certificate**.
3. Browse to the UDDI registry SSL certificate file, and click **Open**.
4. Click **Use Subject** on the right of the **Alias Name** field, and click **OK**. The registry SSL certificate is now imported into the certificate store, and must be added to the Java keystore.
5. Click **Keystore** on the **Certificate** window.
6. Click **Browse** next to the **Keystore** field.
7. Browse to the following file:

```
INSTALL_DIR/policystudio/jre/lib/security/cacerts
```

8. Click **Open**, and enter the **Keystore password**.
9. Click **Add to Keystore**.
10. Browse to the registry SSL certificate imported earlier, select it, and click **OK**.
11. Restart Policy Studio. You should now be able to connect to the registry over SSL.

Configure mutual SSL authentication

If mutual SSL authentication is required (if Policy Studio must authenticate to the registry), Policy Studio must have an SSL private key and certificate. In this case, you should create a keystore containing the Policy Studio key and certificate. You must configure Policy Studio to load this file. For example, edit the `INSTALL_DIR/policystudio/policystudio.ini` file, and add the following arguments:

```
-Djavax.net.ssl.keyStore=/home/axway/osr-client.jks
-Djavax.net.ssl.keyStorePassword=changeit
```

This example shows an `osr-client.jks` keystore file used with Oracle Service Registry (OSR), which is the UDDI registry provided by Oracle.

Note You can also use Policy Studio to create a new keystore (`.jks`) file. Click **New keystore** instead of browsing to the `cacerts` file as described earlier.

Retrieve WSDL files from a UDDI registry

Overview

You can use WSDL files to register web services in the **Web Service Repository** and to add WSDL documents and XML schemas to the global cache. Policy Studio can retrieve a WSDL file from the file system, from a URL, or from a UDDI registry. This topic explains how to retrieve a WSDL file from a UDDI registry. For details on how to register WSDL files, see *Manage web services* on page 92. For details on how to publish WSDL files, see *Publish WSDL files to a UDDI registry* on page 144.

You can also browse a UDDI registry in Policy Studio directly without registering a WSDL file. Under the **APIs** node in the tree, right-click the **Web Service Repository** node, and select **Browse UDDI Registry**.

UDDI concepts

Universal Description, Discovery and Integration (UDDI) is an OASIS-led initiative that enables businesses to publish and discover web services on the Internet. A business publishes services that it provides to a public XML-based registry so that other businesses can dynamically look up the registry and discover these services. Enough information is published to the registry to enable other businesses to find services and communicate with them. In addition, businesses can also publish services to a private or semi-private registry for internal use.

A business registration in a UDDI registry includes the following components:

- **Green Pages**:
  Contains technical information about the services exposed by the business
- **Yellow Pages**:
  Categorizes the services according to standard taxonomies and categorization systems
- **White Pages**:
  Gives general information about the business, such as name, address, and contact information

You can search the UDDI registry according to a whole range of search criteria, which is of key importance to Policy Studio. You can search the registry to retrieve the WSDL file for a particular service. Policy Studio can then use this WSDL file to create a policy for the service, or to extract a schema from the WSDL to check the format of messages attempting to use the operations exposed by the Web service.

For a more detailed description of UDDI, see the UDDI specification. In the meantime, the next section gives high-level definitions of some of the terms displayed in the Policy Studio interface.

UDDI definitions

Because UDDI terminology is used in Policy Studio windows, such as the **Import WSDL** wizard, the following list of definitions explains some common UDDI terms. For more detailed explanations, see the UDDI specification.

businessEntity
This represents all known information about a particular business (for example, name, description, and contact information). A `businessEntity` can contain a number of `businessService` entities. A `businessEntity` may have an `identifierBag`, which is a list of name-value pairs for identifiers, such as Data Universal Numbering System (DUNS) numbers or taxonomy identifiers. A `businessEntity` may also have a `categoryBag`, which is a list of name-value pairs used to tag the `businessEntity` with classification information such as industry, product, or geographic codes. There is no mapping for a WSDL item to a `businessEntity`. When a WSDL file is published, you must specify a `businessEntity` for the `businessService`.

businessService
A `businessService` represents a logical service classification, and is used to describe a web service provided by a business. It contains descriptive information in business terms outlining the type of technical services found in each `businessService` element. A `businessService` may have a `categoryBag`, and may contain a number of `bindingTemplate` entities. In the WSDL to UDDI mapping, a `businessService` represents a `wsdl:service`. A `businessService` has a `businessEntity` as its parent in the UDDI registry.

bindingTemplate
A `bindingTemplate` contains pointers to the technical descriptions and the access point URL of the web service, but does not contain the details of the service specification. A `bindingTemplate` may contain references to a number of `tModel` entities, which do include details of the service specification. In the WSDL to UDDI mapping, a `bindingTemplate` represents a `wsdl:port`.

tModel
A `tModel` is a web service type definition, which is used to categorize a service type. A `tModel` consists of a key, a name, a description, and a URL. `tModel` s are referred to by other entities in the registry. The primary role of the `tModel` is to represent a technical specification (for example, WSDL file). A specification designer can establish a unique technical identity in a UDDI registry by registering information about the specification in a `tModel`. Other parties can express the availability of web services that are compliant with a specification by including a reference to the `tModel` in their `bindingTemplate` data.

This approach facilitates searching for registered web services that are compatible with a particular specification. `tModel`s are also used in `identifierBag` and `categoryBag` structures to define organizational identity and various classifications. In this way, a `tModel` reference represents a relationship between the keyed name-value pairs to the super-name, or namespace in which the name-value pairs are meaningful. A `tModel` may have an `identifierBag` and a `categoryBag`. In the WSDL to UDDI mapping, a `tModel` represents a `wsdl:binding` or `wsdl:portType`.

Identifier
The purpose of identifiers in a UDDI registry is to enable others to find the published information using more formal identifiers such as DUNS numbers, Global Location Numbers (GLN), tax identifiers, or any other kind of organizational identifiers, regardless of whether these are private or shared.

The following are identification systems used commonly in UDDI registries:

Identification System	Name	tModel Key
Dun and Bradstreet D-U-N-S Number	`dnb-com:D-U-N-S`	`uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823`
Thomas Registry Suppliers	`thomasregister-com:supplierID`	`uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039`

Category
Entities in the registry may be categorized according to categorization system defined in a `tModel` (for example, geographical region). The `businessEntity`, `businessService`, and `tModel` types have an optional `categoryBag`. This is a collection of categories, each of which has a name, value, and `tModel` key.

The following are categorization systems used commonly in UDDI registries:

Categorization System	Name	tModel Key
UDDI Type Taxonomy	`uddi-org:types`	`uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4`
North American Industry Classification System (NAICS) 1997 Release	`ntis-gov:naics:1997`	`uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2`

Example tModel mapping for WSDL portType

The following shows an example `tModel` mapped for a WSDL `portType`:

```
<tModel tModelKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3">
  <name>
    StockQuotePortType
  </name>
  <overviewDoc>
```

```
    <overviewURL>
      http://location/sample.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824"
keyName="portType namespace"
      keyValue="http://example.com/stockquote/" />
    <keyedReference tModelKey="uuid:6e090afa-33e5-36eb-81b7-1ca18373f457"
keyName="WSDL type"
      keyValue="portType" />
  </categoryBag>
</tModel>
```

In this example, the `tModel` name is the same as the local name of the WSDL `portType` (in this case, `StockQuotePortType`), and the `overviewURL` links to the WSDL file. The `categoryBag` specifies the WSDL namespace, and shows that the `tModel` is for a `portType`.

Configure a registry connection

You first need to select the UDDI registry to search for WSDL files. Complete the following fields to select or add a UDDI registry:

Select Registry:
Select an existing UDDI registry to browse for WSDL files from the **Registry** drop-down list. To configure the location of a new UDDI registry, click **Add**. Similarly, to edit an existing UDDI registry location, click **Edit**. Then configure the fields in the **Registry Connection Details** dialog. For more details, see *Connect to a UDDI registry* on page 129.

Select Locale:
You can select an optional language locale from this list. The default is `No Locale`.

WSDL search

When you have configured a UDDI registry connection, you can search the registry using a variety of different search options on the **Search** tab. **WSDL Search** is the default option. This enables you to search for the UDDI entries that the WSDL file is mapped to. You can also do this using the **Advanced Search** option. The following different types of WSDL searches are available:

WSDL portType (UDDI tModel):
Searches for a `uddi:tModel` that corresponds to a `wsdl:portType`. You can enter optional search criteria for specific categories in the `uddi:tModel` (for example, **Namespace of portType**).

WSDL binding (UDDI tModel):
Searches for a `uddi:tModel` that corresponds to a `wsdl:binding`. You can enter optional search criteria for specific categories in the `uddi:tModel` (for example, **Name of binding**, or **Binding Transport Type**).

WSDL service (UDDI businessService):
Searches for a `uddi:businessService` that corresponds to a `wsdl:service`. You can enter optional search criteria for specific categories in the `uddi:businessService` (for example, **Namespace of service**).

WSDL port (UDDI bindingTemplate):
Searches for a `uddi:bindingTemplate` that corresponds to a `wsdl:port`.This search is more complex because a `serviceKey` is required to find a `uddi:bindingTemplate`. This means that at least two queries are carried out, first to find the `uddi:businessService`, and another to find the `uddi:bindingTemplate`.

For example, a `bindingTemplate` contains a reference to the `tModel` for the `wsdl:portType`. You can use the `tModel` key to find all implementations (`bindingTemplate`s) for that `wsdl:portType`. The search looks for `businessService`s that have `bindingTemplate`s that refer to the `tModel` for the `wsdl:portType`. Then with the `serviceKey`, you can find the `bindingTemplate` that refers to the `tModel` for the `wsdl:portType`.

In all cases, click **Next** to start the WSDL search. The **Search Results** tree shows the `tModel` URIs as top-level nodes. These URIs are all WSDL URIs, and you can use these to generate policies on import by selecting the URI, and clicking the **Finish** button.

You can click any of the nodes in the tree to display detailed properties about that node in the table below the **Search Results** tree. The properties listed depend on the type of the node that is selected. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

Quick search

The **Quick Search** option enables you to search the UDDI registry for a specific `tModel` name or category.

tModel Name:
You can enter a **tModel Name** for a fine-grained search. This is a partial or full name pattern with wildcard searching as specified by the *SQL-92 LIKE* specification. The wildcard characters are percent `%`, and underscore `_`, where an underscore matches any single character and a percent matches zero or more characters.

Categories:
You can select one of the following options to search by:

wsdlSpec	Search for `tModel`s classified as `wsdlSpec` (`uddi-org:types` category set to `wsdlSpec`). This is the default.

Reusable WS-Policy Expressions	Search for `tModel`s classified as reusable WS-Policy Expressions.
All	Search for all `tModel`s.

Click **Next** to start the search. The **Search Results** tree shows the `tModel` URIs as top-level nodes. These URIs are all WSDL URIs, and you can use these to generate policies on import by selecting the URI, and clicking the **Finish** button.

You can click any of the nodes in the tree to display detailed properties about that node in the table below the **Search Results** tree. The properties listed depend on the type of the node that is selected. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

Name search

The **Name Search** option enables you to search for a `businessEntity`, `businessService`, or `tModel` by name. In the **Select Registry Data Type**, select one of the following UDDI entity levels to search for:

- **businessEntity**
- **businessService**
- **tModel**

You can enter a name in the **Name** field to narrow the search. You can also use wildcards in the name. The name applies to a `businessEntity`, `businessService`, or `tModel`, depending on which registry entity type has been selected. If no name is entered, all entities of the selected type are retrieved.

Click the **Search** button to start the search. The search results display the matching entities in the tree. For example, if you enter `MyTestBusiness` for **Name**, and select **businessEntity**, this searches for a `businessEntity` with the name `MyTestBusiness`. Child nodes of the matching `businessEntity` nodes are also shown. `tModel`s are displayed in the results if any child nodes of the `businessEntity` refer to `tModel`s. Only referred to `tModel`s are displayed. The same applies if you search for a `businessService`. If you select `tModel`, and search for `tModel`s, only `tModel`s are displayed.

Note The `tModel` URIs shown in the resulting tree may not all be categorized as `wsdlSpec` according to the `uddi-org:types` categorization system. You can choose to load any of these URIs as a WSDL file, but you are warned if it is not categorized as `wsdlSpec`.

As before, you can click any node in the results tree to display properties about that node in the table. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

UDDI v3 name searches

By default, a UDDI v3 name search is an exact match. To perform a search using wildcards (for example, `%`, `_`, and so on), you must select the **approximateMatch** find qualifier in the **Advanced Options** tab. This applies to anywhere you enter a name for search purposes (for example, **Name Search**, **Quick Search**, and **Advanced Search**).

Advanced search

The **Advanced Search** option enables you to search the UDDI registry using any combination of **Names**, **Keys**, **tModels**, **Discovery URLs**, **Categories**, and **Identifiers**. You can also specify the entity level to search for in the tree. All of these options combine to provide a very powerful search facility. You can specify search criteria for any of the categories listed above by right-clicking the folder node in the **Enter Search Criteria** tree, and selecting the **Add** menu option. You can enter more than one search criteria of the same type (for example, two **Key** search criteria).

Note The `tModel` URIs shown in the resulting tree may not all be categorized as `wsdlSpec` according to the `uddi-org:types` categorization system. You can choose to load any of these URIs as a WSDL file, but you are warned if it is not categorized as `wsdlSpec`.

The following options enable you to add a search criteria for each of the types listed in the **Enter Search Criteria** tree. All search criteria are configured by right-clicking the folder node, and selecting the **Add** menu option.

Names:
Enter a name to be used in the search in the **Name** field in the **Name Search Criterion** dialog. For example, the name could be the **businessEntity** name. The name is a partial or full name pattern with wildcards allowed as specified by the *SQL-92 LIKE* specification. The wildcard characters are percent `%`, and underscore `_`, where an underscore matches any single character and a percent matches zero or more characters. A name search criterion can be used for `businessEntity`, `businessService`, and `tModel` level searches.

Keys:
In the **Key Search Criterion** dialog, you can specify a key to search the registry for in the **Key** field. The key value is a Universally Unique Identifier (UUID) value for a registry object. You can use the **Key Search Criterion** on all levels of searches. If one or more keys are specified with no other search criteria, the keys are interpreted as the keys of the selected type of registry object and used for a direct lookup, instead of a find/search operation. For example, if you enter `key1` and `key2`, and select the `businessService` entity type, the search retrieves the `businessService` object with key `key1`, and another `businessService` with key `key2`.

If you enter a key with other search criteria, a key criterion is interpreted as follows:

- For a businessService entity lookup, the key is the `businessKey` of the services.
- For a `bindingTemplate` entity lookup, the key is the `serviceKey` of the binding templates.
- Not applicable for any other object type.

tModels:
You can enter a key in the **tModel Key** field on the **tModel Search Criterion** window. The key entered should correspond to the UUID of the `tModel` associated with the type of object you are searching for. A `tModel` search criterion may be used for `businessEntity`, `businessService`, and `bindingTemplate` level searches.

Discovery URLs:
Enter a URL in the **Discovery URL** field on the **Discovery URL Search Criterion** dialog. The **Use Type** field is optional, but can be used to further fine-grain the search by type. You can use a Discovery URL search criterion for `businessEntity` level searches only.

Categories:
Select a previously configured categorization system from the **Type** drop-down list in the **Category Search Criterion** dialog. This prepopulated with a list of common categorization systems. You can add a new categorization system using the **Add** button.

In the **Add/Edit Category** dialog, enter a **Name**, **Description**, and **UUID** for the new category type in the fields provided. When the categorization system has been selected or added, enter a value to search for in the **Value** field. The **Name** field is optional.

Identifiers:
Select a previously configured identification system from the **Type** drop-down list in the **Identifier Search Criterion** dialog. This is prepopulated with well-known identification systems. To add a new identification system, click the **Add** button.

In the **Add/Edit Identifier** dialog, enter a **Name**, **Description**, and **UUID** for the new identifier in the fields provided.

Select Registry Data Type:
Select one of the following UDDI entity levels to search for:

- **businessEntity**
- **businessService**
- **bindingTemplate**
- **tModel**

The search also displays child nodes of the matching nodes. `tModel`s are also returned if they are referred to.

Advanced options

This tab enables you to configure various aspects of the search conditions specified on the previous tabs. The following options are available:

UDDI Find Qualifier	Description
andAllKeys	By default, identifier search criteria are ORed together. This setting ensures that they are ANDed instead. This is already the default for `categoryBag` and `tModelBag`.
approximateMatch (v3)	This applies to a UDDI v3 registry only. Specifies wildcard searching as defined by the `uddi-org:approximatematch:SQL99 tModel`, which means approximate matching where percent sign (`%`) indicates any number of characters, and underscore (`_`) indicates any single character. The backslash character (`\`) is an escape character for the percent sign, underscore and backslash characters. This option adjusts the matching behavior for `name`, `keyValue`, and `keyName` (where applicable).
binarySort (v3)	This applies to a UDDI v3 registry only. Enables greater speed in sorting, and causes a binary sort by name, as represented in Unicode codepoints.
bindingSubset (v3)	This applies to a UDDI v3 registry only. Specifies that the search uses only `categoryBag` elements from contained `bindingTemplate` elements in the registered data, and ignores any entries found in the `categoryBag` that are not direct descendents of registered `businessEntity` or `businessService` elements.
caseInsensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that that the matching for `name`, `keyValue`, and `keyName` (where applicable) should be performed without regard to case.
caseInsensitiveSort (v3)	This applies to a UDDI v3 registry only. Specifies that the result set should be sorted without regard to case. This overrides the default case sensitive sorting behavior.
caseSensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that that the matching for `name`, `keyValue`, and `keyName` (where applicable) should be case sensitive. This is the default behavior.
caseSensitiveSort (v3)	This applies to a UDDI v3 registry only. Specifies that the result set should be sorted with regard to case. This is the default behavior.

UDDI Find Qualifier	Description
combineCategoryBags	Makes the `categoryBag` entries of a `businessEntity` behave as if all `categoryBag` s found at the `businessEntity` level and in all contained or referenced `businessService` s are combined. Searching for a category yields a positive match on a registered business if any of the `categoryBag`s contained in a `businessEntity` (including the `categoryBag`s in contained or referenced `businessService`s) contain the filter criteria.
diacriticInsensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that matching for `name`, `keyValue`, and `keyName` (where applicable) should be performed without regard to diacritics. Support for this qualifier by nodes is optional.
diacriticSensitiveMatch (v3)	This applies to a UDDI v3 registry only. Specifies that matching for `name`, `keyValue`, and `keyName` (where applicable) should be performed with regard to diacritics. This is the default behavior.
exactMatch (v3)	This applies to a UDDI v3 registry only. Specifies that only entries with `name`, `keyValue`, and `keyName` (where applicable) that exactly match the name argument passed in, after normalization, are returned. This qualifier is sensitive to case and diacritics where applicable. This is the default behavior.
exactNameMatch (v2)	This applies to a UDDI v2 registry only. Specifies that the name entered as part of the search criteria must exactly match the name specified in the UDDI registry.
orAllKeys	By default, `tModel` and category search criteria are ANDed. This setting ORs these criteria instead.
orLikeKeys	When a bag container contains multiple `keyedReference` elements (`categoryBag` or `identifierBag`), any `keyedReference` filters from the same namespace (for example, with the same `tModelKey` value) are ORed together rather than ANDed. For example, this enables you to search for `any of these four values from this namespace, and any of these two values from this namespace`.

UDDI Find Qualifier	Description
serviceSubset	Causes the component of the search that involves categorization to use only the `categoryBag`s from directly contained or referenced `businessService`s in the registered data. The search results return only those businesses that match based on this modified behavior, in conjunction with any other search arguments provided.
signaturePresent (v3)	This applies to a UDDI v3 registry only. This restricts the result to entities that contain, or are contained in, an XML Digital `Signature` element. The `Signature` element should be verified by the client. This option, or the presence of a `Signature` element, should only be used to refine a search result, and should not be used as a verification mechanism by UDDI clients.
sortByDateAsc (v3)	This applies to a UDDI v3 registry only. Sorts the results alphabetically in order of ascending date.
sortByDateDsc (v3)	This applies to a UDDI v3 registry only. Sorts the results alphabetically in order of descending date.
sortByNameAsc	Sorts the results alphabetically in order of ascending name.
sortByNameDsc	Sorts the results alphabetically in order of descending name.
suppressProjectedServices (v3)	This applies to a UDDI v3 registry only. Specifies that service projections must not be returned when searching for services or businesses. This option is enabled by default when searching for a service without a `businessKey`.
UTS-10 (v3)	This applies to a UDDI v3 registry only. Specifies sorting of results based on the Unicode Collation Algorithm on elements normalized according to Unicode Normalization Form C. A sort is performed according to the Unicode Collation Element Table in conjunction with the Unicode Collation Algorithm on the name field, and normalized using Unicode Normalization Form C. Support for this qualifier by nodes is optional.

Publish

Click the **Publish** radio button to view the **Published UDDI Entities Tree View**. This enables you to manually publish UDDI entities to the specified UDDI registry (for example, `businessEntity`, `businessService`, `bindingTemplate`, and `tModel` entities). You

must already have the appropriate permissions to write to the UDDI registry.

Add a businessEntity

To add a business, perform the following steps:

1. Right-click the tree view, and select **Add businessEntity**.
2. In the **Business** dialog, enter a **Name** and **Description** for the business. Click **OK**.
3. You can right-click the new `businessEntity` node to add child UDDI entities in the tree (for example, `businessService`, `Category`, and `Identifier` entities).

Add a tModel

To add a `tModel` , perform the following steps:

1. Right-click the tree view, and select **Add tModel**.
2. In the **tModel** dialog, enter a **Name**, **Description**, and **Overview URL** for the `tModel` . For example, you can use the **Overview URL** to specify the location of a WSDL file. Click **OK**.
3. You can right-click the new `tModel` node to add child UDDI entities in the tree (for example, `Category` and `Identifier` entities).

As before, you can click any node in the results tree to display properties about that node in the table. You can also right-click a node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node). At any stage, you can click the **Clear** button on the right to clear the entire contents of the tree. This does not delete the contents of the registry.

For more details on UDDI entities such as `businessEntity` and `tModel`, see *UDDI definitions* on page 134. For details on how to publish web services automatically using a wizard, see *Publish WSDL files to a UDDI registry* on page 144.

Publish WSDL files to a UDDI registry

Overview

You can register web services in the **Web Service Repository** using Web Services Description Language (WSDL) files. Policy Studio can retrieve a WSDL file from the file system, from a URL, or from a UDDI registry. When you have registered a WSDL file in the web service repository, you can use the **Publish WSDL** wizard to publish the WSDL file to a UDDI registry. You can also use the **Find WSDL** wizard to search for the selected WSDL file in a UDDI registry. This topic explains how to perform both of these tasks.

For background information and an introduction to general UDDI concepts, see *Retrieve WSDL files from a UDDI registry* on page 133. For details on how to register WSDL files, see *Manage web services* on page 92.

Find WSDL files

You can search a UDDI registry to determine if a web service is already published in the registry. To search for a selected WSDL file in a specified UDDI registry, perform the following steps:

1. In the Policy Studio tree, expand the **APIs** > **Web Service Repository** node.
2. Right-click a WSDL node and select **Find in UDDI Registry** to launch the **Find WSDL** wizard.
3. In the **Find WSDL** dialog, select a UDDI registry from the list. You can add or edit a registry connection using the buttons provided. For details on configuring a registry connection, see *Connect to a UDDI registry* on page 129.
4. You can select an optional language **Locale** from the list. The default is `No Locale`.
5. Click **Next**. The **WSDL Found in UDDI Registry** window displays the result of the search in a tree. The **Node Counts** field shows the total numbers of each UDDI entity type returned from the search (`businessEntity`, `businessService`, `bindingTemplate`, and `tModel`).
6. You can right-click to edit a UDDI entity node in the tree, if necessary (for example, add a description, add a category or identifier node, or delete a duplicate node).
7. Click the **Refresh** button to run the search again.
8. Click **Finish**.

The **Find WSDL** wizard provides is a quick and easy way of finding a selected WSDL file published in a UDDI registry. For more fine-grained ways of searching a UDDI registry (for example, for specific WSDL or UDDI entities), see *Retrieve WSDL files from a UDDI registry* on page 133.

Publish WSDL files

To publish a WSDL file registered in the **Web Service Repository** to a UDDI registry, perform the following steps:

1. Expand the **API** > **Web Service Repository** tree node.
2. Right-click a WSDL node and select **Publish WSDL to UDDI Registry** to launch the **Publish WSDL Wizard**.
3. Perform the steps in the wizard as described in the next sections.

Step 1: Enter virtualized service address and WSDL URL for publishing in UDDI registry

When you register a WSDL file in the web service repository, API Gateway exposes a *virtualized* version of the web service. The host and port for the web service are changed dynamically to point to the machine running API Gateway. The client can then retrieve the WSDL for the virtualized web service from API Gateway, without knowing its real location.

This window enables you to optionally override the service address locations in the WSDL file with the virtualized addresses exposed by API Gateway. You can also override the WSDL URL published to the UDDI registry.

Complete the following fields:

Mapping of Service Addresses to Virtualized Service Addresses:
You can enter multiple virtual service address mappings for each service address specified in the selected WSDL file. If you do not enter a mapping, the original address location in the WSDL file is published to the UDDI registry. If one or more mappings are provided, corresponding UDDI `bindingTemplates` are published in the UDDI registry, each with a different access point (virtual service address). This enables you to publish the access points of a service when it is exposed on different ports/schemes using API Gateway.

When you launch the wizard, the mapping table is populated with a row for each `wsdl:service`, `wsdl:port`, `soap:address`, `soap12:address`, or `http:address` in the selected WSDL file. To modify an existing entry, select a row in the table, and click **Edit**. Alternatively, click **Add** to add an entry. In the **Virtualize Service Address** dialog, enter the virtualized service address. For example, if API Gateway is running on a machine named `roadrunner`, the new URL on which the web service is available to clients is:
`http://roadrunner:8080/TestServices/StockQuote.svc`.

WSDL URL:
You can enter a WSDL URL to be published to the UDDI registry in the corresponding `tModel overviewURL` fields. If you do not enter a URL, the WSDL URL in the **Original WSDL URL** field is used. For example, an endpoint service is at `http://coyote.qa.acmecorp.com/TestService/StockQuote.svc`. Assume this service is virtualized in API Gateway and exposed at `http://HOST:8080/TestService/StockQuote.svc`, where `HOST` is the machine on which API Gateway is running. The `http://HOST:8080/TestService/StockQuote.svc` URL is entered as the virtual service address, and `http://HOST:8080/TestService/StockQuote.svc?WSDL` is entered as the WSDL URL to publish.

Note If incorrect URLs are published, you can edit these in the UDDI tree in later steps in this wizard, or when browsing the registry.

Click **Next** when finished.

Step 2: View WSDL to UDDI mapping result

You can use this window to view the unpublished mapping of the WSDL file to a UDDI registry structure. You can also edit a specific mapping in the tree view. This window includes the following fields:

Mapping of WSDL to a UDDI Registry Structure:
The unpublished mappings from the WSDL file to the UDDI registry are displayed in the table. For example, this includes the relevant `businessService`, `bindingTemplate`, `tModel`, `Identifier`, `Category` mappings. You can select a tree node to display its values in the table below.

You can optionally edit the values for a specific mapping in the table (for example, update a value, or add a key or description for the selected UDDI entity). You can also right-click a tree node to edit it (for example, add a description, add a category or identifier node, or delete a duplicate node).

Retrieve service address from WSDL instead of bindingTemplate access point:
When selected, this ensures that the `bindingTemplate` access point does not contain the service port address, and is set to `WSDL` instead. This means that you must retrieve the WSDL to get the service access point. When selected, the `bindingTemplate` contains an additional `tModelInstanceInfo` that points to the `uddi:uddi.org.wsdl:address tModel`. This option is not selected by default.

Include WS-Policy as:
When selected, you can choose one of the following options to specify how WS-Policy statements in the WSDL file are included in the registry:

Remote Policy Expressions	Each WS-Policy URL in the WSDL that is associated with a mapped UDDI entity is accessed remotely. For example, a `businessService` is categorized with the `uddi:w3.org:ws-policy:v1.5:attachment:remotepolicyreference tModel` where the `keyValue` holds the remote WS-Policy URL. This is the default option.
Reusable Policy Expressions	Each WS-Policy URL in the WSDL that is associated with a mapped UDDI entity has a separate `tModel` published for it. Other UDDI entities (for example, `businessService`) can then refer to these `tModel`s. These are reusable because UDDI entities published in the future can also use these `tModel`s. You can do this in *Step 4: Select a duplicate publishing approach* on page 148, by selecting the **Reuse duplicate tModels** option.

Click **Next** when finished.

Step 3: Select a registry for publishing

Use this window to select a UDDI registry in which to publish the WSDL to UDDI mapping. Complete the following fields:

Select Registry:
Select an existing UDDI registry to browse for WSDL files from the **Registry** drop-down list. To configure the location of a new UDDI registry, click **Add**. Similarly, to edit an existing UDDI registry location, click **Edit**. For details on how to configure a UDDI connection, see *Connect to a UDDI registry* on page 129.

Select Locale:
You can select an optional language locale from this list. The default is `No Locale`.

Click **Next** when finished.

Step 4: Select a duplicate publishing approach

This window is displayed only if mapped WSDL entities already exist in the UDDI registry. Otherwise, the wizard skips to *Step 5: Create or search for business* on page 149. This window includes the following fields:

Select Duplicate Mappings:
The **Mapped WSDL to publish** pane on the left displays the unpublished WSDL mappings from *Step 2: View WSDL to UDDI mapping result* on page 147. The **Duplicates for WSDL mappings in UDDI registry** pane on the right displays the nodes already published in the registry. The **Node List** at the bottom right shows a breakdown of the duplicate nodes.

Edit Duplicate Mappings:
You can eliminate duplicate mappings by right-clicking a tree node in the right or left pane, and selecting edit to update a specific mapping in the dialog. Select the **Refresh** button on the right to run the search again, and view the updated **Node List**. Alternatively, you can configure the options in the next field.

Select Publishing Approach for Duplicate Entries:
Select one of the following options:

Reuse duplicate tModels	Publishes the selected entries from the tree on the left, and reuses the selected duplicate entries in the tree on the right. This is the default option. Some or all duplicate `tModel`s (for example, for `portType`, `binding`, and reusable WS-Policy expressions) that already exist in the registry can be reused. This means that a new `businessService` that points to existing `tModel`s is published. Any entries selected on the left are published, and any referred to `tModel`s on the left now point to selected duplicate `tModel`s on the right. By default, this option selects all `businessServices` on the left, and all duplicate `tModel`s on the right. If there is more than one duplicate `tModel`s, only the first is selected.

Overwrite duplicates	Publishes the selected entries from the tree on the left, and overwrites the selected duplicate entries in the tree on the right. When a UDDI entity is overwritten, its UUID key stays the same, but all the data associated with it is overwritten. This is not just a transfer of additions or differences. You can also overwrite some duplicates and create some new entries. By default, this option selects all `businessService`s and `tModel`s on the left and all duplicate `businessService`s and `tModel`s on the right. If there is more than one duplicate, only the first is selected. The default overwrites all selected duplicates and does not create any new UDDI entries, unless there is a new referred to `tModel` (for example, for a reusable WS-Policy expression).
Ignore duplicates	Publishes the selected entries from the tree on the left, and ignores all duplicates. You can proceed to publish the mapped WSDL to UDDI data. New UDDI entries are created for each item that is selected in the tree on the left.

Click **Next** when finished.

Note If you select duplicate `businessService`s in the tree, and select **Overwrite duplicates**, the wizard skips to *Step 6: Publish WSDL* on page 150 when you click **Next**.

Step 5: Create or search for business

Use this window to specify a `businessEntity` for the web service. You can create a new `businessEntity` or search for an existing one in the UDDI registry. Complete the following fields:

Create a new businessEntity:
This is the default option. Enter a **Name** and **Description** for the `businessEntity`, and click **Publish**.

Search for an existing businessEntity:
To search for an existing `businessEntity` name, perform the following steps:

1. Select the **Search for an existing businessEntity in the UDDI registry** option.
2. In the **Search** tab, ensure the **Name Search** option is selected.
3. Enter a **Name** option (for example, `Acme Corporation`).

Alternatively, you can select the **Advanced Search** option to search by different criteria such as **Keys**, **Categories**, or **tModels**. You can also select a range of search options on the **Advanced** tab (for example, **Exact match**, **Case sensitive**, or **Service subset**). For more details, see *Retrieve WSDL files from a UDDI registry* on page 133.

The **Node Counts** field shows the total numbers of each UDDI entity type returned from the search (`businessEntity`, `businessService`, `bindingTemplate`,and `tModel`).

Click **Next** when finished.

Step 6: Publish WSDL

Use this to publish the WSDL to the UDDI registry.

Selected businessEntity for Publishing:
This field displays the name and `tModel` key of the `businessEntity` to be published. Click the **Publish WSDL** button on the right.

Published WSDL:
This field displays the tree of the UDDI mapping for the WSDL file. You can right-click to edit or delete any nodes in the tree if necessary, and click **Refresh** to run the search again. Click **Publish WSDL** to publish your updates.

Click **Finish**.

Messaging 4

This section describes how to configure messaging services.

Configure messaging services

A *messaging system* is a loosely coupled, peer-to-peer facility where clients can send messages to, and receive messages from any other client. In a messaging system, a client sends a message to a messaging agent. The recipient of the message can then connect to the same agent and read the message. However, the sender and recipient of the message do not need to be available at the same time to communicate (for example, unlike HTTP). The sender and recipient need only know the name and address of the messaging agent to talk to.

Java Message Service (JMS) is an implementation of such a messaging system. It provides an API for creating, sending, receiving, and reading messages. Java-based applications can use it to connect to other messaging system implementations. A *JMS provider* can deliver messages synchronously or asynchronously, which means that the client can fire and forget messages or wait for a response before resuming processing. Furthermore, the JMS API ensures different levels of reliability in terms of message delivery. For example, it can ensure that the message is delivered once and only once, or at least once.

API Gateway uses the JMS API to connect to other messaging systems that expose a JMS interface (for example, Oracle WebLogic Server, IBM WebSphere MQ, Red Hat JBoss Messaging, Apache ActiveMQ, or Progress SonicMQ). As a consumer of a JMS queue or topic, the API Gateway can read XML messages and pass them into a policy for validation. These messages can then be routed on over HTTP or dropped on to another JMS queue or topic.

API Gateway also provides a default embedded Apache ActiveMQ service, which enables it to act as a JMS server. For example, this enables API Gateway to integrate external facing REST APIs and SOAP Web services with back-end systems and applications using reliable asynchronous messaging.

Prerequisites

API Gateway provides all the required third-party JAR files for IBM WebSphere MQ and Apache ActiveMQ (both embedded and external).

Note For other third-party JMS providers only, you must add the required third-party JAR files to the API Gateway classpath for messaging to function correctly. If the provider's implementation is platform-specific, copy the provider JAR files to `INSTALL_DIR/ext/PLATFORM`.
`INSTALL_DIR` is your API Gateway installation, and `PLATFORM` is the platform on which API Gateway is installed (`Linux.x86_64`). If the provider implementation is platform-independent, copy the JAR files to `INSTALL_DIR/ext/lib`.

Configure API Gateway messaging using the JMS wizard

You can use the **JMS Wizard** to configure an API Gateway instance to consume JMS messages from a JMS queue or topic. When a message has been consumed by the API Gateway, it can be sent to a configured policy where the full range of API Gateway message filters can run on the message. The message can then be routed onwards over HTTP or dropped back on to a JMS queue or topic. The **JMS Wizard** guides you through all of the necessary steps to configure messaging (for example, the JMS service, JMS session, and JMS consumer).

To launch the **JMS Wizard**, right-click the instance under the **Environment Configuration > Listeners** node in the Policy Studio tree, and select **Messaging System > JMS Wizard**. The wizard includes the following windows:

JMS Service Provider

The first window in the wizard enables you to configure connection details to the JMS provider that produces the JMS messages that are consumed by the API Gateway. For details on configuring the fields on this window, see *Configure a JMS service* on page 153. Click **Next** when finished.

JMS Session Configuration

The second window in the wizard enables you to configure settings such as **Remove message from source** for the JMS session that is established with the JMS provider. For details on configuring these settings, see *Configure a JMS session* on page 157. Click **Next** when finished.

JMS Consumer Configuration

The third window in the wizard enables you to configure JMS consumer settings. For details on configuring the fields on this window, see *Configure a JMS consumer* on page 158. Click **Finish** to complete.

Configure global JMS services in external connections

Alternatively, you can configure a global JMS service under the **Environment Configuration > External Connections** node in Policy Studio by right-clicking the **JMS Services** node, and selecting **Add a JMS Service**.

The configured global JMS services can then be used by the API Gateway to drop messages on to a JMS queue or topic, or to read messages from a JMS queue or topic (for example, using the **Send to JMS** or **Read from JMS** filter).

For more details, see *Configure a JMS service* on page 153.

Configure embedded Apache ActiveMQ in API Gateway settings

You can use the API Gateway server settings to configure the default embedded Apache ActiveMQ broker available in API Gateway, which enables it to act as a JMS service provider.

In the Policy Studio tree, select **Environment Configuration > Server Settings > Messaging > Embedded ActiveMQ**. For example, you can enable embedded ActiveMQ, and configure location and SSL security settings.

Note Apache ActiveMQ 5.14.3 restricts serializing object message types. For more details, see the *API Gateway Administrator Guide*.

Monitor messaging using API Gateway Manager

You can use the API Gateway Manager web console to monitor messaging at runtime. For example, you can create, delete, and view JMS topics, queues, and messages at runtime.

For more details, see the *API Gateway Administrator Guide*.

Configure a JMS service

You can configure a global JMS service under the **Environment Configuration** > **External Connections** node in Policy Studio by right-clicking the **JMS Services** node, and selecting **Add a JMS Service** . The details entered in the **JMS Service** dialog can then be used by the API Gateway to drop messages on to a JMS queue or topic, or to read messages from a JMS queue or topic. For more details, see the following filters:

- *Send to JMS* on page 161
- *Read from JMS* on page 166

Alternatively, you can configure a JMS service at the API Gateway instance level, and configure the API Gateway to consume a JMS queue or topic. Right-click the instance under the **Environment Configuration** > **Listeners** node in the Policy Studio, and select **JMS Wizard**.

General configuration

Configure the following fields on the **JMS Service** tab:

Name:

Enter a descriptive name for the JMS provider in the **Name** field.

Service type:

Select one of the following from the list:

- **Embedded Apache ActiveMQ**: The default Apache ActiveMQ service that is embedded in the API Gateway.
- **Apache ActiveMQ**: An external Apache ActiveMQ service that is not embedded in the API Gateway.
- **IBM MQ**: An IBM WebSphere MQ service. See *IBM WebSphere MQ settings* on page 154.
- **Standard JMS**: Other systems that support the JMS standard (for example, Oracle WebLogic Server, IBM MQSeries, JBoss Messaging, or Progress SonicMQ).

Apache ActiveMQ and Standard JMS settings

The following settings are displayed when you select a **Service Type** of Embedded Apache ActiveMQ, Apache ActiveMQ, or Standard JMS:

Provider URL:

Enter the URL of the JMS provider. For example, a URL for a JBoss application server might be `jnp://localhost:1099`. Defaults to `local` for Embedded Apache ActiveMQ.

Initial Context Factory:

API Gateway uses a connection factory to create a connection with a JMS provider. A connection factory encapsulates a set of connection configuration parameters that have been defined by the administrator. The following are some example default values:

- Embedded Apache ActiveMQ: `com.vordel.ama.jndi.InitialContextFactory`
- External Apache ActiveMQ: `com.vordel.jms.apache.activemq.InitialContextFactory`
- JBoss application server: `org.jnp.interfaces.NamingContextFactory`

Connection Factory:

Enter the name of the connection factory to use when connecting to the JMS provider. The name of the connection factory is vendor-specific. For example, the connection factory for the JBoss application server is `org.jnp.interfaces:javax.jnp`. Defaults to `connectionFactory` for embedded and external ActiveMQ.

IBM WebSphere MQ settings

The following settings are displayed when you select a **Service Type** of **IBM MQ**:

Host name:

Enter the host name of the JMS provider (for example, `localhost`).

Port number:

Enter the port number of the JMS provider (for example, `1414`).

Queue manager:

Enter the virtual queue manager name by which IBM WebSphere Application Server is known to WebSphere MQ (for example, `TEST_BUS`).

Channel:

Enter the IBM WebSphere MQ channel name on the WebSphere MQ system (for example, `MY_QM.TO.TEST_BUS`).

Initial Context Factory:

The API Gateway uses a connection factory to create a connection with a JMS provider. A connection factory encapsulates a set of connection configuration parameters that have been defined by the administrator. Defaults to `com.vordel.jms.ibm.mq.InitialContextFactory`.

Connection Factory:

Enter the name of the connection factory to use when connecting to the JMS provider. Defaults to `connectionFactory`.

Settings for all service types

The following optional settings are common to all service types:

Username:

If a user name is required to connect to this JMS provider, enter it here.

Password:

Enter the password for this user.

Custom Message Properties:

You can add JNDI context settings by clicking **Add**, and entering name and value properties in the fields.

For the **Embedded Apache ActiveMQ** service type, you can define Apache ActiveMQ URI parameters using JNDI properties. For example, see the following:

- http://activemq.apache.org/tcp-transport-reference.html
- http://activemq.apache.org/connection-configuration-uri.html
- http://activemq.apache.org/redelivery-policy.html
- http://activemq.apache.org/ssl-transport-reference.html
- http://activemq.apache.org/what-is-the-prefetch-limit-for.html

Configure advanced settings

You can configure the following options on the **Advanced Settings** tab:

JMS service settings

The advanced JMS service settings are as follows:

JMS Client ID:

Enter the client ID required by JMS durable topic subscriptions to consume messages from the service. For more details, see the following:

- *Configure a JMS consumer* on page 158
- *Read from JMS* on page 166

Max sessions for JMS filters:

Enter the maximum number of sessions that are created for the JMS filters (**Send To JMS** and **Read From JMS**) using this JMS service. The default value is 20.

Automatic reconnection:

Select whether a reconnection to the JMS server is performed when the configured JMS provider raises a connection error. This setting is selected by default.

Start first connection asynchronously:

Select whether the first connection attempt is detached from the API Gateway startup sequence. When this setting is selected, API Gateway starts even if the JMS connection cannot be established.

SSL settings

Note SSL settings are available only for the **IBM MQ** and external **Apache ActiveMQ** JMS service types.

You can configure the following SSL settings:

Cipher suite:

Click the browse button on the right, and select SSL cipher suites from the list of JSSE or IBM cipher suites in the dialog (for example, **SSL_RSA_WITH_RC4_128_MD5**).

Note When using an **IBM MQ** JMS service type, you can select only one SSL cipher suite. For more details, see your IBM WebSphere MQ documentation. When using an **Apache ActiveMQ** JMS service type, you can select multiple cipher suites.

Trusted certificates:

When a cipher suite is selected, you can select SSL trusted certificates and authorities from the list. The selected certificates are used to check the JMS server certificate.

Client certificate (SSL mutual authentication):

Click **Client Certificate** to select the SSL client certificate and key to use. This setting is required only for SSL mutual authentication.

Next steps

When the JMS service has been configured, you can configure the API Gateway to drop messages on to a queue or topic exposed by this service. You can do this when configuring a policy by selecting the service in the **Send to JMS** or **Read from JMS** filters. For more details, see the following:

- *Send to JMS* on page 161
- *Read from JMS* on page 166

You can also configure JMS sessions for the newly added JMS service at the API Gateway instance level. For more details, see *Configure a JMS session* on page 157.

Configure a JMS session

JMS services have JMS sessions, which can be shared by multiple JMS consumers, or used by a single JMS consumer only. To configure a JMS session, right-click an API Gateway instance under the **Environment Configuration** > **Listeners** node in the Policy Studio tree, and select **Messaging System** > **Add JMS Session**. Alternatively, you can configure a JMS session using **Messaging System** > **JMS Wizard**.

Note You must have first configured a JMS service before you can configure a JMS session. For more details, see *Configure a JMS service* on page 153.

JMS session configuration

The JMS session settings that are displayed on the **Session** tab depend on whether you selected **Add JMS Session** or **JMS Wizard**.

Add JMS session only

If you selected **Messaging System** > **Add JMS Session**, configure the following fields:

JMS service:

Click the browse button on the right, and select a preconfigured JMS service. To add a service, right-click **JMS Services**, and select **Add a JMS Service**. For more details, see *Configure a JMS service* on page 153.

Listener Count:

Specify the number of listeners permitted for this JMS session. Defaults to 1. If the volume of messages arriving at the queue is more than a single thread can process, you can increase the number of threads listening on the queue by increasing the listener count.

Common configuration

In both cases (**Add JMS Session** and **JMS Wizard**), configure the following fields:

Remove message from source:

Select one of the following options from the list:

- **Immediately when message is read**: Message is removed immediately after it is read.
- **Lazily which will allow for duplicate message**: Message is removed lazily, which allows possible duplicate messages and compatibility with previous versions of API Gateway.
- **When policy completes without error**: Message is removed if the configured policy completes, either with a succeess or failure. If the configured policy does not complete due to an error, the message is not removed. This option allows possible duplicate messages and compatibility with previous versions of API Gateway. This is the default option.
- **When policy completes and property below evaluates to true**: Message is removed if the message attribute configured in **Message removal property** evaluates to `true`. This attribute is set to `true` by default.

  Note After the configured policy executes, if a message is not removed, it is then rolled back. You may need to configure an error path for the messages to prevent a poison message loop.

Message removal property:

Enter the message attribute name used by the **When policy completes and selector below evaluates to true** option.

Monitoring options

The **Traffic Monitor** tab enables you to configure traffic monitoring settings for the JMS session. To override the system-level traffic monitoring settings, select **Override system-level settings**, and configure the relevant options. For more details, see the *API Gateway Administrator Guide*.

Next steps

When the JMS session has been configured, you can configure JMS consumers for the newly added JMS session at the API Gateway instance level. For more details, see *Configure a JMS consumer* on page 158.

Configure a JMS consumer

You can configure multiple JMS consumers under a single JMS session, which share that session. Alternatively, you can configure a single JMS consumer per JMS session. Consumers sharing a JMS session access that session serially. Each consumer blocks until a response (if any) is received.

Consumers with their own session do not encounter this problem, which may improve performance.

You can configure JMS consumers using the JMS Wizard, or by right-clicking an existing JMS session, and selecting **Add JMS Consumer**.

Note You must first configure a JMS service and a JMS session before you can configure a JMS consumer. For more details, see *Configure a JMS service* on page 153.

JMS Message source

On the **General** tab, configure the following fields in the **Message source** section:

Source type:

Select one of the following from the list:

- **Queue**
- **Topic**
- **JNDI lookup**

Defaults to **Queue**.

Source Name:

Enter the name of the JMS queue, JMS topic, or JNDI lookup to specify where you want read the messages from.

Note The source name to use depends on the configured **Source type**. For example, for IBM WebSphere MQ, this name may need to use a format of `queue://`. For details on source name requirements, please see the user documentation for your third-party JMS provider tool.

Selector:

Enter an SQL selector expression that specifies a response message. The expression entered specifies the messages that the consumer is interested in receiving. By using a selector, the task of filtering the messages is performed by the JMS provider instead of by the consumer.

The selector is a string that specifies an expression whose syntax is based on the SQL92 conditional expression syntax. The API Gateway instance only receives messages whose headers and properties match the selector. For example, the following expression gets the selected message from the queue:

```
JMSCorrelationID='${params.query.id}'
```

JMS consumer type

The **JMS consumer type** settings enable you to configure the following:

Durable subscription:

Select this setting to use a durable topic subscription to consume messages from the server. This option is available only for **Topic** and **JNDI lookup** source types.

Topic subscriber name:

Enter the JMS subscriber name used to identify the durable subscription.

Note The JMS service used must have a **JMS Client ID** configured. If a **JNDI lookup** source is configured, the name must not point to a topic.
Only one durable subscriber (described by the JMS client ID and subscriber name) can be active at a time.

Message processing

The **Message processing** settings include the following:

Extraction Method:

Specify how to extract the data from the JMS message from the drop-down list:

- Create a `content.body` attribute based on the SOAP over JMS draft specification (the default)
- Insert the JMS message directly into the attribute named below
- Populate the attribute below with the value inferred from message type to Java

Attribute Name:

The name of the API Gateway message attribute that holds the data extracted from the JMS message. Defaults to the `jms.message` message attribute.

Policy:

Select the appropriate policy from the list to run on the JMS message after it has been consumed by the API Gateway. This setting is required.

Send Response to Configured Destination:

Specifies whether the API Gateway sends a reply to the response queue named in the incoming message (in the `ReplyTo` header). This option is selected by default. Deselecting this option means that the API Gateway never sends a reply to the response queue named in the `ReplyTo` header.

Logging settings

The **Logging Settings** tab enables you to configure the logging level for all filters in policies executed on this JMS consumer, and to configure when message payloads are logged.

Transaction Audit Logging Level

You can configure the following settings on all filters executed on the JMS consumer:

Logging level	Description
Fatal	Logs Fatal log points that occur on all filters executed.
Failure	Logs Failure log points that occur on all filters executed. This is the default logging level.
Success	Logs Success log points that occur on all filters executed.

For details on logging levels, and configuring logging for a filter, see "Set transaction log level and log message" in the *API Gateway Policy Developer Filter Reference*.

Transaction Audit Payload Logging

You can configure the following settings for this JMS consumer:

Payload logging	Description
On receive request from client	Log the message payload when a request arrives from the client.
On send response to client	Log the message payload before the response is sent back to the client.
On send request to remote server	Log the message payload before the request is sent using any **Connection** or **Connect to URL** filters deployed in policies.
On receive response from remote server	Log the message payload when the response is received using any **Connection** or **Connect to URL** filters deployed in a policies.

For details on how to log message payloads at any point in a specific policy, see "Log message payload" in the *API Gateway Policy Developer Filter Reference*.

Send to JMS

The **Send to JMS** filter enables you to configure a JMS messaging system to which the API Gateway sends messages. You can configure various settings for the message request and response (for example, destination and message type, how the message system should respond, and so on).

API Gateway provides all the required third-party JAR files for IBM WebSphere MQ and Apache ActiveMQ (both embedded and external).

Note For other third-party JMS providers only, you must add the required third-party JAR files to the API Gateway classpath for messaging to function correctly. If the provider's implementation is platform-specific, copy the provider JAR files to `INSTALL_`

DIR/ext/PLATFORM.
INSTALL_DIR is your API Gateway installation, and PLATFORM is the platform on which API Gateway is installed (Linux.x86_64). If the provider implementation is platform-independent, copy the JAR files to INSTALL_DIR/ext/lib.

Request settings

The **Request** tab specifies the following properties of the request sent to the messaging system:

JMS Service:
Click the browse button on the right, and select an existing JMS service in the tree. To add a JMS Service, right-click the **JMS Services** tree node, and select **Add a JMS Service**. Alternatively, you can configure JMS services under the **Environment Configuration** > **External Connections** node in the Policy Studio tree. For more details, see *Configure messaging services* on page 151.

Destination type:
Select one of the following from the list:

- **Queue**
- **Topic**
- **JNDI lookup**

Defaults to **Queue**.

Destination:
Enter the name of the JMS queue, JMS topic, or JNDI lookup to specify where you want to drop the messages.

Note The destination name to use depends on the configured **Destination type**. For example, for IBM WebSphere MQ, this name may need to use a format of queue://. For details on destination name requirements, please see the user documentation for your third-party JMS provider tool.

Delivery Mode:
Select one of the following delivery modes:

- **Persistent**:
  Instructs the JMS provider to ensure that a message is not lost in transit if the JMS provider fails. A message sent with this delivery mode is logged to persistent storage when it is sent. This is the default mode.
- **Non-persistent**:
  Does not require the JMS provider to store the message. With this mode, the message may be lost if the JMS provider fails.

Priority Level:
You can use message priority levels to instruct the JMS provider to deliver urgent messages first. The ten levels of priority range from 0 (lowest) to 9 (highest). If you do not specify a priority level, the default level is 4. A JMS provider tries to deliver higher priority messages before lower priority ones but does not have to deliver messages in exact order of priority.

Time to Live:
By default, a message never expires. However, if a message becomes obsolete after a certain period, you may want to set an expiry time (in milliseconds). The default value is `0`, which means the message never expires.

Message ID:
Enter an identifier to be used as the unique identifier for the message. By default, the unique identifier is the ID assigned to the message by API Gateway (`${id}`). However, you can use a proprietary correlation system, perhaps using MIME message IDs instead of API Gateway message IDs.

Correlation ID:
Enter an identifier for the message that API Gateway uses to correlate response messages with the corresponding request messages. Usually, if `${id}` is specified in the **Message ID** field, it is also used here to correlate request messages with their correct response messages.

Message Type:
This list enables you to specify the type of data to be serialized and sent in the JMS message to the JMS provider. The option selected depends on what part of the message you want to send to the consumer. For example, to send the message body, select the option to format the body according to the rules defined in the SOAP over JMS recommendation. Alternatively, to serialize a list of name-value pairs to the JMS message, choose the option to create a `MapMessage`.

Select one of the following serialization options:

- **Use content.body attribute to create a message in the format specified in the SOAP over Java Message Service recommendation**:
  If this option is selected, messages are formatted according to the SOAP over JMS recommendation. This is the default option because in most cases the message body is routed to the messaging system. When this option is selected, a `javax.jms.BytesMessage` is created and a JMS property containing the content type `text/xml`) is set on the message.
- **Create a MapMessage from the java.lang.Map in the attribute named below**:
  Select this option to create a `javax.jms.MapMessage` from the API Gateway message attribute named below that consists of name-value pairs.
- **Create a BytesMessage from the attribute named below**:
  Select this option to create a `javax.jms.BytesMessage` from the API Gateway message attribute named below.
- **Create an ObjectMessage from the java.lang.Serializable in the attribute named below**:
  Select this option to create a `javax.jms.ObjectMessage` from the API Gateway message attribute named below.
- **Create a TextMessage from the attribute named below**:
  Select this option to create a `javax.jms.TextMessage` from the message attribute named below.
- **Use the javax.jms.Message stored in the attribute named below**:
  If a `javax.jms.Message` has already been stored in a message attribute, select this option, and enter the name of the attribute in the field below.

Attribute Name:
Enter the name of the API Gateway message attribute that holds the data that is to be serialized to a JMS message and sent over the wire to the JMS provider. The type of the attribute named here must correspond to that selected in the **Message Type** field above.

Custom Message Properties:
You can set custom properties for messages in addition to those provided by the header fields. Custom properties may be required to provide compatibility with other messaging systems. You can use message attribute selectors as property values. For example, you can create a property called `AuthNUser`, and set its value to `${authenticated.subject.id}`. Other applications can then filter on this property (for example, only consume messages where `AuthNUser` equals `admin`). To add a new property, click **Add**, and enter a name and value in the fields provided on the **Properties** dialog.

Use the following policy to change JMS request message:
This setting enables you to customize the JMS message before it is published to a JMS queue or topic. Click the browse button on the right, and select a configured policy in the dialog. The selected policy is then invoked before the JMS request is sent to the queuing system.

When the selected policy is invoked, the JMS request message is available on the white board in the `jms.outbound.message` message attribute. You can therefore call JMS API methods to manipulate the JMS request further. For example, you could configure a policy containing a **Scripting Language** filter that runs a script such as the following against the JMS message:

```
function invoke(msg) {
  var jmsMsg = msg.get("jms.outbound.message");
  jmsMsg.setIntProperty("My_JMS_Report", 123);
  return true;
}
```

Response settings

The **Response** tab specifies whether API Gateway uses asynchronous or synchronous communication when talking to the messaging system. For example, to use asynchronous communication, select the **Do not set response** option. If synchronous communication is required, you can select to read the response from a temporary queue or from a named queue or topic.

You can also specify whether API Gateway waits on a response message from a queue or topic from the messaging system. API Gateway sets the `JMSReplyTo` property on each message that it sends. The value of the `JMSReplyTo` property is the temporary queue, queue, or topic selected in this dialog. It is the responsibility of the application that consumes the message from the queue (JMS consumer) to send the message back to the destination specified in `JMSReplyTo`.

API Gateway sets the `JMSCorrelationID` property to the value of the **Correlation ID** field on the **Request** tab to correlate requests messages to their corresponding response messages. If you select to use a temporary queue or temporary topic, this is created when API Gateway starts up.

Configure how messaging system should respond:
Select where the response message is to be placed using one of the following options:

- **Do not set response**:
  Select this option if you do not expect or do not care about receiving a response from the JMS provider.
- **Use temporary queue**:
  Select this option to instruct the JMS provider to place the response message on a temporary queue. In this case, the temporary queue is created when API Gateway starts up. Only API Gateway can read from the temporary queue, but any application can write to it. API Gateway uses the value of the `JMSReplyTo` header to indicate the location where it reads responses from.
- **Use queue**:
  If you want the JMS provider to place response messages on a queue, select this option, and enter the queue name in the text box. This is used in the `JMSReplyTo` field of the response message.
- **Use topic**:
  If you want the JMS provider to place response messages on a topic, select this option, and enter the topic name in the text box. This is used in the `JMSReplyTo` field of the response message.
- **Use named queue or topic (JNDI)**:
  If you want the JMS provider to place response messages on a named queue or topic using JNDI lookup, select this option, and enter the JNDI name for the queue or topic in the text box. This is used in the `JMSReplyTo` field of the response message.

Wait for response:
If **Do not set response** is not selected, you can select whether API Gateway waits to receive a response:

- **Wait with timeout (ms)**:
  API Gateway waits a specific time period to receive a response before it times out. If API Gateway times out waiting for a response, the **Send to JMS** filter fails. Enter the timeout value in milliseconds. The default value of `10000` means that API Gateway waits for a response for 10 seconds. The accepted range of values is `10000` –`20000` ms.
- **Selector for response**:
  If **Wait with timeout (ms)** is selected, you can enter an SQL selector expression that specifies a response message. The expression entered specifies the messages that the consumer is interested in receiving. By using a selector, the task of filtering the messages is performed by the JMS provider instead of by the consumer.

  The selector is a string that specifies an expression whose syntax is based on the SQL92 conditional expression syntax. The API Gateway instance only receives messages whose headers and properties match the selector. For example:

  ```
  JMSCorrelationID='${params.query.id}'
  ```

Note The JMS consumer automatically returns the results of the invoked policy to the JMS destination specified in the `JMSReplyTo` header in the request. This means that you do not need to send a reply using the **Send to JMS** filter.
If the incoming JMS message contains a `JMSReplyTo` header, the queue or topic expects

a response. So when the JMS consumer policy completes, API Gateway sends a message to the `JMSReplyTo` source in reverse. For example, the consumer reads the JMS message, and populates an attribute with a value inferred from the message type to Java (for example, from `TextMessage` to `String`). When the policy completes, the consumer looks up this attribute and infers the JMS response message type based on the object type stored in the message.

Read from JMS

The **Read from JMS** filter enables you to configure a JMS messaging system from which the API Gateway reads messages. You can configure various settings for the JMS message source, message type, and processing options.

API Gateway provides all the required third-party JAR files for IBM WebSphere MQ and Apache ActiveMQ (both embedded and external).

Note For other third-party JMS providers only, you must add the required third-party JAR files to the API Gateway classpath for messaging to function correctly. If the provider's implementation is platform-specific, copy the provider JAR files to `INSTALL_DIR/ext/PLATFORM`.
`INSTALL_DIR` is your API Gateway installation, and `PLATFORM` is the platform on which API Gateway is installed (`Linux.x86_64`). If the provider implementation is platform-independent, copy the JAR files to `INSTALL_DIR/ext/lib`.

Message source

The **Message source** settings enable you to configure the following:

JMS Service:
Click the browse button on the right, and select an existing JMS service in the tree. To add a JMS Service, right-click the **JMS Services** tree node, and select **Add a JMS Service**. Alternatively, you can configure JMS services under the **Environment Configuration** > **External Connections** node in the Policy Studio tree. For more details, see *Configure messaging services* on page 151.

Source type:
Select one of the following from the list:

- **Queue**
- **Topic**
- **JNDI lookup**

Defaults to **Queue**.

Source Name:
Enter the name of the JMS queue, JMS topic, or JNDI lookup to specify where you want read the messages from.

Note The source name to use depends on the configured **Source type**. For example, for IBM WebSphere MQ, this name may need to use a format of `queue://`. For details on source name requirements, please see the user documentation for your third-party JMS provider tool.

Selector:
Enter an SQL selector expression that specifies a response message. The expression entered specifies the messages that the consumer is interested in receiving. By using a selector, the task of filtering the messages is performed by the JMS provider instead of by the consumer.

The selector is a string that specifies an expression whose syntax is based on the SQL92 conditional expression syntax. The API Gateway instance only receives messages whose headers and properties match the selector. For example, the following expression gets the selected message from the queue:

```
JMSCorrelationID='${params.query.id}'
```

Read timeout (ms):
Enter the timeout after which the **Read from JMS** filter fails. The accepted range of values is 1–20000 ms. Defaults to `1000` ms.

JMS consumer type

The **JMS consumer type** settings enable you to configure the following:

Durable subscription:
Create or use a durable topic subscription to consume messages from the server. This option is only available for **Topic** and **JNDI lookup** source types.

Note This is only available with a **Topic** source and the JMS service used must have a client ID configured. If a **JNDI lookup** source is configured, the name must not point to a topic.

Topic subscriber name:
Enter the JMS subscriber name used to identify the durable subscription.

Message processing

The **JMS consumer type** settings enable you to configure the following:

Extraction Method:
Specify how to extract the data from the JMS message from the list:

- Insert the JMS message directly into the attribute named below (this is the default)
- Populate the attribute below with the value inferred from message type to Java

Attribute Name:
The name of the API Gateway message attribute that holds the data extracted from the JMS message. Defaults to the `jms.message` message attribute.

Policy:
Select the appropriate policy to run on the JMS message after it has been consumed by the API Gateway.

Send Response to Configured Destination:
Specifies whether the API Gateway sends a reply to the response queue named in the incoming message (in the `ReplyTo` header). This option is selected by default. Deselecting this option means that the API Gateway never sends a reply to the response queue named in the `ReplyTo` header.

General configuration 5

This section describes some general configuration settings for API Gateway.

Manage Policy Studio projects

Overview

This topic explains how work with Policy Studio projects when developing API Gateway policies and configuration. For example, this includes basic tasks such as creating, opening, saving, and closing projects. It also includes tasks such as deploying, importing, and exporting configuration from a project, changing a project passphrase, and adding APIs and web services to a project.

Create a new project

To create a new project in Policy Studio, select **File** > **New Project** from the main menu, or click **New Project** on the welcome page. Follow the steps in the **New Project** wizard.

For more details, see *Create a Policy Studio project* on page 31.

Open an existing project

To open an existing project in Policy Studio, select **File** > **Open Project** from the main menu, or click **Open Project** on the welcome page, and enter the project details in the dialog. Alternatively, click a link to the existing project in the **Recent Projects** pane on the welcome page.

For more details, see *Manage API Gateway connections* on page 172.

Save changes to a project

When a project is loaded in Policy Studio, you can select **File** > **Save** from the main menu to save the changes to the current configuration editor.

Alternatively, select **File** > **Save All** to save the changes to all open unsaved configuration editors.

Deploy changes to a project

When a project is loaded in Policy Studio, you can select **Tasks** > **Deploy** from the main menu to deploy saved changes to a running API Gateway instance at any time. Alternatively, click the **Deploy** button in the toolbar.

For more details, see *Deploy API Gateway configuration* on page 210.

Add an API to a project

When a project is loaded in Policy Studio, you can select **Tasks** > **Add REST API** from the main menu to add an API to the project.

For more details, see *Develop REST APIs in Policy Studio* on page 121.

Virtualize a web service

When a project is loaded in Policy Studio, you can select **Tasks** > **Virtualize a Service** from the main menu to use the API Gateway to virtualize a web service.

For more details, see *Register and secure web services* on page 75.

Change the project passphrase

You can use the `projchangepass` command to change the encryption passphrase for a Policy Studio project. For more information on `projchangepass`, see "Automate processes for continuous integration" in the *API Gateway DevOps Deployment Guide*.

Note It is important to distinguish between the passphrase used by a project on the local file system and the passphrase used by an API Gateway group configuration on a running API Gateway instance. For details on specifying a different passphrase for runtime, see *Deploy API Gateway configuration* on page 210.

For more details on configuring encryption passphrases, see the *API Gateway Administrator Guide*.

Export configuration packages

When a project is loaded in Policy Studio, you call select **File** > **Export** to save the project as a configuration package. Select one of the following from the menu:

- **Deployment package**:
  Saves the project as a `.fed` file that contains all API Gateway configuration. This includes policies, listeners, external connections, users, certificates, and environment settings.
- **Policy package**:
  Saves the project as a `.pol` file that contains users, certificates, and environment settings.
- **Environment package**:
  Saves the project as a `.env` file that contains policies, listeners, external connections, and environment settings.

When you have saved a configuration package, you can use it to create a Policy Studio project in another environment. For more details, see *Create a Policy Studio project* on page 31.

For more details on configuration packages to promote configuration between environments, see the *API Gateway DevOps Deployment Guide*.

Import configuration into a project

When a project is loaded in Policy Studio, you can select **File** > **Import** > **Configuration Fragment** from the main menu to import configuration into the project. This feature enables you to import XML-based configuration previously exported from Policy Studio. For more details, see *Import API Gateway configuration fragment* on page 187.

You can also select **File** > **Import** > **Custom filters** to import custom filters to be added to the Policy Studio filter palette. For more details, see the *API Gateway Developer Guide*.

Manage multiple projects

You can work with multiple projects in Policy Studio. When a project is loaded in Policy Studio, you can click the Show List icon (**>>**) in the toolbar to display a list of currently open views (for example, other open projects and the welcome page). Select an item in the list to switch between views. The toolbar also displays the number of available views beside the Show List icon in the toolbar.

Close a project

To close the project currently open in Policy Studio, select **File** > **Close Project** from the main menu. Alternatively, click the **X** icon next to the project name in the toolbar.

Manage API Gateway connections

Overview

You can use Policy Studio to manage API Gateway server instances. You can open a server connection when creating an project based on an API Gateway instance, when opening an existing project, or when deploying a configuration.

Create a new project based on an API Gateway instance

When you select **File** > **New Project** to create a new project, you can select to use an API Gateway instance as the starting point for the project configuration. For details on configuring the server connection for the new project, see *Create a Policy Studio project* on page 31.

Note You must first create an API Gateway project before you can connect to a server.

Open an existing project

You can open an existing project using **File** > **Open Project** in the main menu, or by clicking **Open Project** on the welcome page or clicking a link to the existing project in the **Recent Projects** pane.

When you select **File** > **Open Project** or click **Open Project** on the welcome page, you can configure the following connection settings in the **Open project** window:

Location:
Enter or browse to the full path to the project in the file system (for example, `C:\Users\john\apiprojects\my_test_project`). You can select from the connection history of existing projects in the list.

Project Name:
If a valid project location is selected, Policy Studio completes this field with the project name configured in the `.project` file in the specified project **Location**.

Passphrase:
Enter the encryption passphrase if one has been configured. For more details, see *Manage Policy Studio projects* on page 169.

Clear History:
Click this button to clear the connection history in the **Recent Projects** pane on the welcome page.

Open a connection when deploying

When you select **Deploy** in the toolbar to deploy updated API Gateway configuration to a server, you must first open the server connection. For details on configuring the server connection when deploying, see *Deploy API Gateway configuration* on page 210.

Note You must first create an API Gateway project before you can deploy configuration.

Unlock a server connection

You can also unlock an existing connection to a server. This is for emergency use if you have changed configuration, and this results in you being locked out from the **Management Services** on port `8090`. In this case, you have incorrectly configured the authentication filter in the **Protect Management Interfaces** policy.

For example, if you created and deployed an LDAP connection without specifying the correct associated user accounts, and are now unable to connect to the Admin Node Manager.

To unlock a server connection, perform the following steps:

1. Download all the files in the server's `conf/fed` directory to the machine on which Policy Studio is installed.
2. Start Policy Studio.
3. Create a new project from existing configuration based on the `conf/fed` directory that you downloaded from the server in step 1 (for details, see *Create a Policy Studio project* on page 31).
4. Change the configuration details as required (for example, specify the correct user account details for the LDAP connection under the **Environment Configuration** > **External Connections** node).
5. Upload the files back to the server's `conf/fed` directory.
6. Restart the Admin Node Manager.

For more details on **Management Services**, see *Configure HTTP services* on page 275.

Global configuration

For convenience, Policy Studio displays various global configuration options. For example, it includes libraries of users, X.509 certificates, and schemas that can be added globally and then referenced in filters and policies. This avoids the need to reconfigure details over and over again (for example, each time a schema or certificate is used).

The following global configuration options are available in Policy Studio, each of which are discussed briefly in the sections below:

API Gateway settings

You can configure the underlying configuration settings for API Gateway using the **Environment Configuration > Server Settings** node in the Policy Studio tree. This includes the following settings:

- API Manager
- General
- Logging
- Messaging
- Monitoring
- Security

For more details, see the *API Gateway Administrator Guide*.

Web service repository

The easiest way to secure a web service with API Gateway is to import the Web Services Description Language (WSDL) file for the service using Policy Studio. This creates a **Service Handler** for the web service, which is used to control and validate requests to the web service and responses from the web service.

The WSDL file is also added to the web service repository, making sure to update the URL of the web service to point at the machine on which API Gateway is running instead of that on which the web service is running. Consumers of the web service can then query API Gateway for the WSDL file for the web service. The consumer then knows to route messages to API Gateway instead of attempting to route directly to the web service, which most likely is not available on a public IP address.

The web service repository offers a very simple way of securing a web service with minimal impact on consumers of that service. Because of this, the web service repository should be used as the primary method of setting up policies within Policy Studio. For more information on using the repository to register a web service, see *Register and secure web services* on page 75.

API Gateway instances

A single running instance of API Gateway enables you to configure at least two interfaces: one for public traffic, and a second for listening for and serving configuration data. The configuration interface should rarely need to be updated. However, you might need to add several HTTP interfaces. For example, an HTTP interface and an SSL-enabled HTTPS interface.

Furthermore, you can add features such as the following at the API Gateway instance level:

- Remote hosts to control connection settings to a server
- SMTP interfaces to configure email relay
- File transfer services for FTP, FTPS, and SFTP
- Policy execution schedulers to run policies at regular time intervals
- JMS listeners to listen for JMS messages
- Packet sniffers to inspect packets at the network level for logging and monitoring
- FTP pollers to retrieve files to be processed by polling a remote file server
- Directory scanners to scan messages dumped to the file system

Because API Gateway can read messages from HTTP, SMTP, FTP, JMS, or a directory, this enables it to perform protocol translation. For example, API Gateway can read a message from a JMS queue, and then route it on over HTTP to a web service. Similarly, API Gateway can read XML messages that have been put into a directory on the file system using FTP, and send them to a JMS messaging system, or route them over HTTP to a back-end system.

For more information on configuring API Gateway instances, see *Configure API Gateway instances* on page 264.

Policies

A policy is made up of a sequence of modular, reusable message filters, each of which processes the message in a particular way. There are many categories of filters available, including authentication, authorization, content filtering, routing, and many more. For example, a typical policy might contain an authentication filter, followed by several content-based filters (for example, Schema Validation, Threatening Content, Message Size, XML Complexity, and so on), and provided all configured filters run successfully, the message is routed on to the configured destination.

A policy can be thought of as a network of message filters. A message can traverse different paths through the network depending on what filters succeed or fail. This enables you to configure policies that, for example, route messages that pass one Schema Validation filter to one back-end system, and route messages that pass a different Schema Validation filter to a different system.

You can use *policy containers* to help manage your policies. These are typically used to group together a number of similar policies (for example, all authentication policies) or to act as an umbrella around several policies that relate to a particular policy (for example, all policies for the `getQuote` web service). A number of useful policies that ship with API Gateway are found in the **Policy Library** policy container. This container is prepopulated with policies to return various types of faults to the client and policies to block certain types of threatening content, among others. You can also add your own policies to this container, and create your own policy containers as necessary to suit your own requirements.

Certificates and keys

API Gateway must be able to trust X.509 certificates to establish SSL connections with external servers, validate XML Signatures, encrypt XML segments for certain recipients, and for other such cryptographic operations. Similarly, a private key is required to carry out certain other cryptographic operations, such as message signing and decrypting data.

The **Certificate Store** contains all the certificates and keys that are considered to be trusted by the API Gateway. Certificates can be imported into or created by the certificate store. You can also assign a private key to the public key stored in a certificate, by importing the private key, or by generating one using the provided interface.

For more information on importing and creating certificates and keys, see *Manage X.509 certificates and keys* on page 221.

API Gateway user store

Users are mainly used for authentication purposes in API Gateway. In this context, the **User Store** acts as a repository for user information against which users can be authenticated. You can also store user attributes for each user or user group. For example, you can then use these attributes when generating SAML attribute assertions on behalf of the user.

Manage API Gateway users on page 233 contains more details on how to create users, user groups, and attributes.

System alerts

API Gateway can send system alerts to various error reporting systems in the case of a policy error (for example, when a request is blocked by a policy). For more details on how to configure API Gateway to send these alerts, and for details of all of the alert destinations that you can configure, see *Configure system alerts* on page 235.

External connections

API Gateway can leverage your existing identity management infrastructure and avoid maintaining separate silos of user information. For example, if you already have a database full of user credentials, API Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, API Gateway can authorize users, look up user attributes, and validate certificates against third-party identity management servers.

You can add each connection to an external system as a global **External Connection** in Policy Studio so that it can be reused across all filters and policies. For example, if you create a policy that authenticates users against an LDAP directory and then validates an XML signature by retrieving a public key from the same LDAP directory, it makes sense to create a global external connection for that LDAP directory. You can then select the LDAP connection in both the authentication and XML signature verification filters, rather than having to reconfigure it in both filters.

For example, you can use the external connections interface to configure global connections such as the following:

- Authentication repository profiles
- Database connections
- ICAP servers
- JMS services
- Kerberos services
- LDAP connections
- Proxy servers
- Radius clients
- SiteMinder connections
- TIBCO connections
- Tivoli connections
- XKMS connections

You can also use external connections to configure a group of related URLs. This is most useful to round-robin between a number of related URLs to ensure high availability. When API Gateway is configured to use a *URL Connection Set* (instead of just a single URL), it round-robins between the URLs in the set.

For more information on configuring external connections and connection sets, see *External connections* on page 359.

Caches

You can configure API Gateway to cache responses from a back-end web service. For example, if API Gateway receives two successive identical requests it can (if configured) take the response for this request from the cache instead of routing the request on to the web service and asking it to generate the response again.

As a result, excess traffic is diverted from the web service making it more responsive to requests for other services. API Gateway is saved the processing effort of routing identical requests unnecessarily to the web service, and the client benefits from the far shorter response time.

You can configure local caches for each running instance of API Gateway. If you have deployed multiple API Gateways throughout your network, you can configure a distributed cache where cache events on one cache are replicated across all others. For example, if a response message is cached at one instance of API Gateway, it is added to all other caches.

For more details on how to configure API Gateway to use local and distributed caches, see *Global caches* on page 245.

Black list and White list

The **White list** is a global library of regular expressions that can be used across several different filters. For example, the **Validate HTTP Headers**, **Validate Query String**, and **Validate Message Attributes** filters all use regular expressions from the **White list** to ensure that various parts of the request contain expected content.

The **White list** is prepopulated with regular expressions that can be used to identify common data formats, such as alphanumeric characters, dates, email addresses, IP addresses, and so on. For example, if a particular HTTP header is expected to contain an email address, the **Email Address** expression from the library can be run against the HTTP header to ensure that it contains an email address as expected. This is yet another way that the API Gateway can ensure that only the correct data reaches the web service.

While the **White list** contains regular expressions to identify valid data, the **Black list** contains regular expressions that are used to identify common attack signatures. For example, this includes expressions to scan for SQL injection attacks, buffer overflow attacks, ASCII control characters, DTD entity expansion attacks, and many more.

You can run various parts of the request message against the regular expressions contained in the **Black list** library. For example, the HTTP headers, request query string, and message (MIME) parts can be scanned for SQL injection attacks by selecting the SQL-type expressions from the **Black list** .The **Threatening Content** filter also uses regular expressions from the **Black list** to identify attack signatures in request messages.

For more details on running regular expressions, see the following filters in the *API Gateway Policy Developer Filter Reference*:

- **Validate HTTP Headers**
- **Validate Query String**
- **Validate Message Attributes**
- **Threatening Content**

WSDL and XML schema document bundles

The WSDL documents and XML schemas that API Gateway can use to validate incoming requests against are stored in a global cache. The **Schema Validation** filter validates the format of an incoming message against a schema from the cache. This ensures that only messages of the correct format are processed by the target system.

In the Policy Studio navigation tree, you can access the global cache of WSDL documents or XML schema documents by selecting **Resources > WSDL Document Bundles** or **Resources > XML Schema Document Bundles**. Select a child node to view its contents. To add a schema, right-click the **XML Schema Document Bundles** node, and select **Add Schema**. For more details on adding XML schemas to the cache see *Manage WSDL and XML schema documents* on page 98.

When you have imported your XML schemas, see "Schema validation" in the *API Gateway Policy Developer Filter Reference* for instructions on how to validate XML messages against the schemas in the cache.

Scripts

The **Scripts** library contains the JavaScript and Groovy scripts that API Gateway can use to interact with the message as it is processed. For example, you use these scripts with the **Scripting Filter** to get, set, and evaluate specific message attributes.

In the Policy Studio navigation tree, you can access the global scripts library by selecting **Resources > Scripts**. Select a child node to view or edit its contents. To add a script, right-click the **Scripts** node, and select **Add Script**.

For more details on using the **Scripts Library** dialog to add scripts, and on configuring API Gateway to use scripts, see *Scripting language filter* on page 425.

Stylesheets

The **Stylesheets** library contains the XSLT style sheets that API Gateway can use to transform incoming request messages. The **XSLT Transformation** filter enables you convert the contents of a message using these style sheets. For example, an incoming XML message that adheres to a specific XML schema can be converted to an XML message that adheres to a different schema before it is sent to the destination web service.

In the Policy Studio navigation tree, you can access the global style sheet library by selecting **Resources > Stylesheets**. Select a child node to view or edit its contents. To add a style sheet, right-click the **Stylesheets** node, and select **Add Stylesheet**.

For more details on using the **Stylesheet Library** dialog to add style sheets, and on configuring API Gateway to use XSLT style sheets, see "Transform with XSLT" in the *API Gateway Policy Developer Filter Reference*.

References

References can occur between API Gateway configurations items (for example, a policy might include a reference to an external connection to a database). You can view references between configuration items in Policy Studio by right-clicking an item, and selecting **Show All References**. References are displayed in a tab at the bottom of the window.

The **Show All References** option is enabled only for items that have references to other items. For an example in a default API Gateway installation, right-click **Environment Configuration > External Connections > LDAP Connections > Sample Active Directory Connection**, and select **Show all References**. Showing all references is useful for impact analysis (for example, before upgrading or migrating), and is a general navigation aid.

Policy Studio preferences

The **Preferences** dialog enables you to configure a range of settings for Policy Studio. For example, you can configure the level at which Policy Studio traces diagnostic output. To change the Policy Studio settings, select **Window > Preferences** from the main menu. Each of the available settings is discussed in the following sections.

Note When finished your updates, remember to click **Apply** at the bottom of the window, and to click **Deploy** in the toolbar.

Auto-mapping

These settings relate to the auto-mapping feature in Visual Mapper. For more information, see the *API Gateway Visual Mapper User Guide*.

Environmentalization

Environmentalization refers to configuring environment-specific settings for a particular target environment (for example, users, certificates, and external connections for a development environment). You can enable Policy Studio to display settings that have been environmentalized by selecting the **Allow environmentalization of fields** option.

When this option is selected, you can environmentalize a selected field (for example, database URL) by clicking the globe icon to the right of the field. Alternatively, press **Ctrl-E**. When you have selected settings to be environmentalized, the field is disabled, and the globe icon is displayed on the right. You can manage settings that have been environmentalized under the **Environment Settings** node in the Policy Studio tree. For more details, see the *API Gateway DevOps Deployment Guide*.

FIPS mode

The **FIPS Mode** setting enables you to enable Federal Information Processing Standards (FIPS) mode for Policy Studio. This enables FIPS-certified cryptographic modules and ensures that all cryptographic operations (for example, SSL) are performed by these modules. To enable FIPS mode, select **Enable FIPS mode in Axway Policy Studio**.

Note To run in this mode, you must install API Gateway with a FIPS-compliant mode license.

For more details on FIPS, see the *API Management Security Guide*.

Policy colors

The **Policy Colors** settings enable you to customize the look-and-feel of the Policy Canvas in the Policy Studio. For example, you can change the colors of the following components:

- **Policy Background**:
  Changes the background color of the Policy Canvas.
- **Missing Attribute**:
  You can right-click the Policy Canvas, and select **Show All Attributes** from the context menu. When this is selected, each filter displays the list of required and generated message attributes that are relevant for that filter. If a required attribute has not been generated by a previous filter in the policy, the attribute is highlighted in a different color (red by default). You can change this color by selecting an appropriate color using this setting.
- **Success Path**:
  You can change the color of the Success Path link using this setting.
- **Failure Path**:
  Similarly, you can change the color of the Failure Path link here.
- **Show Link Labels**:
  If this option is selected, a Success Path is labeled with the letter S, while a Failure Path is labeled F.

Proxy settings

You can specify global proxy settings that apply only when downloading WSDL, XSD, and XSLT files from Policy Studio. These include the following settings:

Proxy Setting	Description
Host	Host name or IP address of the proxy server.
Port	Port number on which to connect to the proxy server.

Proxy Setting	Description
Username	Optional user name when connecting to the proxy server.
Password	Optional password when connecting to the proxy server.

Tip You can also specify individual proxy servers under the **Environment Configuration** > **External Connections** node in the Policy Studio tree. These are different from the global proxy settings in the **Preferences** because you can specify these proxy servers at the filter level (in the **Connection** and **Connect To URL** filters). For more details, see *Configure proxy servers* on page 409.

Runtime dependencies

The **Runtime Dependencies** setting enables you to add JAR files to the Policy Studio classpath. For example, if you write a custom API Gateway filter, you must add its JAR file, and any third-party JAR files that it uses, to the **Runtime Dependencies** list.

Click **Add** to select a JAR file to add to the list of dependencies, and click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.

Note You must restart Policy Studio and the server for these changes to take effect. You should restart Policy Studio using the `policystudio -clean` command.

Custom filter dependencies

If your custom filter introduces a dependency on a new third-party library, you must first check if the required library is already available under the following directory and sub-directories:

```
INSTALL_DIR/apigateway/system/lib
```

Note Any JAR file that you add under the following directories will be pushed ahead of `apigateway/system` JAR files on the CLASSPATH:

- `INSTALL_DIR/apigateway/ext/lib`
- `INSTALL_DIR/apigateway/groups/`*`GROUP_ID`*`/`*`INSTANCE_ID`*`/ext/lib`

For example, API Gateway ships with specific versions of several Apache Commons JARs. Introducing conflicting versions of these JARs could adversely affect the ability of the API Gateway and Node Manager to function correctly.

SSL settings

The **SSL Settings** enable you to specify what action is taken when an unrecognized server certificate is presented to the client. This allows Policy Studio to connect to SSL services without a requirement to add a certificate to its JVM certificate store.

Configure one of the following options:

Prompt User	When you try to connect to SSL services, you are prompted with a dialog. If you choose to trust this particular server certificate displayed in the dialog, it is stored locally, and you are not prompted again.
Trust All	All server certificates are trusted.
Keystore	Enter or browse to the location of the **Keystore** that contains the authentication credentials sent to a remote host for mutual SSL, and enter the appropriate **Keystore Password**.

Status bar

The **Show Status Bar** setting enables you to specify whether the applications status bar is displayed at the bottom of the Policy Studio window. For example, this status bar displays details such as the currently selected tree node on the left, and details such as the heap size on the right. You can also use the status bar to run garbage collection by clicking the trash icon on the right. This status bar is enabled by default.

Team development

The **Enable Team Development** setting enables team development project templates and project dependencies in Policy Studio. This setting is disabled by default.

Trace level

You can set the level at which Policy Studio logs diagnostic output by selecting the appropriate level from the **Tracing Level** drop-down list. Diagnostic output is written to a file in the `/logs` directory of your Policy Studio installation. You can select **Window** > **Show View** > **Console** in the main menu to view the trace output in the **Console** window at the bottom of the screen. The default trace level is `INFO`.

You can also configure the **Maximum number of files** output to the `/logs` directory. Defaults to 10 files.

WS-I settings

Before importing a WSDL file that contains the definition of a web service into the web service repository, you can test the WSDL file for compliance with the Web Service Interoperability (WS-I) Basic Profile. The WS-I Basic Profile contains a number of *Test Assertions* that describe rules for writing WSDL files for maximum interoperability with other WSDL authors, consumers, and other related tools.

The WS-I settings are described as follows:

WS-I Setting	Description
WS-I Tool Location	Use the **Browse** button to specify the full path to the Java version of the WS-Interoperability testing tools (for example, `C:\Program Files\WSI_Test_Java_Final_1.1\wsi-test-tools`).The WS-I testing tools are used to check a WSDL file for WS-I compliance. You can download them from www.ws-i.org.
Results Type	Select the type of WS-I test results that you wish to view in the generated report from the drop-down list. You can select from `all`, `onlyFailed`, `notPassed`, or `notInfo`.
Message Entry	Specify whether message entries should be included in the report using the check box (selected by default).
Failure Message	Specify whether the failure message defined for each test assertion should be included in the report using the check box (selected by default).
Assertion Description	Specify whether the description of each test assertion should be included in the report using the check box (unselected by default).
Verbose Output	Specify whether verbose output is displayed in the Policy Studio console window using the check box (unselected by default). To view the console window, select **Window** > **Show Console** from the Policy Studio main menu.

Note On Linux, when you download WS-I Testing Tools v1.1, you must run `dos2unix` on `/java/bin/Analyzer.sh` and `/java/bin/setenv.sh`, because both files do not have executable privileges set and have Windows line endings, so the shell interpreter is unable to use them.

For details on running the WS-I testing tools, see *Manage WSDL and XML schema documents* on page 98.

XML settings

The **XML** settings enable you to configure a range of options that affect how XML files are treated in the Policy Studio.

XML Files
This includes the following options:

Creating or saving files	Specifies a line delimiter (for example, `Mac`, `Unix`, `Windows`, or `No translation`).

Creating files	Specifies a file suffix (`xml`), and the type of encoding (for example, `ISO 10646/Unicode(UTF-8)`).
Validating files	Configures whether to warn when no grammar is specified.

Source
This includes the following options:

Formatting	Specifies a range of formatting options (for example, line width, line breaks, and indentation).
Content assist	Specifies whether to make suggestions and which strategy to use (for example, `Lax` or `Strict`).
Grammar constraints	Specifies whether to use inferred grammar in the absence of DTD/Schema.

Syntax Coloring
These settings enable you to associate specific colors with specific XML syntax elements (for example, attribute names, comment delimiters, or processing instruction content).

Policy Studio viewing options

Overview

You can filter the Policy Studio navigation tree on the left of the window to display specified tree nodes only. You can click the **Options** link at the bottom of the tree to display additional viewing options. These enable you to configure whether management services and tree node configuration types are displayed in the tree. Finally, you can configure how the Policy Studio policy filter palette is displayed on the right of the window when editing policies.

Filter the tree

To filter the tree by a specific node name, enter the name in the text box above the tree. When you enter a name (for example, `SOAP Schema`), the tree is filtered automatically, and all occurrences are displayed in the tree.

Filtering the tree is especially useful in cases where many policies have been configured in the Policy Studio, and you wish to find a specific tree node (for example, a schema filter named **Check against SOAP Schema**).

Configure viewing options

When you click the **Options** link at the bottom left of the navigation tree, you can configure the following viewing option:

Show Types:
Select this option to show the **Type** column in the Policy Studio navigation tree. This shows the type of each node in the tree (for example, *HTTP Service* or *Remote Host*). This option is not selected by default. When this option is selected, you can use the **Filter by type** setting.

Configure the policy filter palette

When editing policies, you can configure how the Policy Studio policy filter palette is displayed on the right of the window. Right-click the filter palette, and select from the following options:

Layout:
Specifies how the filters are displayed in each category in the palette. By default, the filters are displayed in a list. Select one of the following options from the context menu:

- Columns
- List
- Icons Only
- Details

Customize:
The **Customize Palette** dialog enables you to customize each of the items displayed in the filter palette. Select a node in the tree on the left to display what can be customized on the right. For example, you can edit a filter name and description, specify whether it is hidden, and add tags to help searches. In addition, you can use the buttons above the tree to add or delete new category drawers or separators. You can also move a selected category drawer up or down in the palette.

Settings:
The **Palette Settings** dialog enables you to customize settings such as fonts, layout, and category drawer options (for example, close each drawer automatically when there is not enough room on the window).

Restore Palette Defaults:
Restores all the palette settings from a default API Gateway installation.

Import API Gateway configuration fragment

Overview

You can import XML-based configuration data into your API Gateway configuration (for example, policies, certificates, and users). For example, this is useful in a development environment if you wish to share and test configuration with other developers.

If you have XML-based configuration data that was previously exported from one API Gateway installation, by importing into another API Gateway installation, you can share API Gateway configuration in a development environment. This also enables you to manage differences and references between configuration components.

For details on exporting configuration data, see *Export API Gateway configuration* on page 189.

Note The recommended way to export configuration between different environments is to use configuration packages. Select **File** > **Export** from the main menu. For more details, see *Manage Policy Studio projects* on page 169.

Import configuration fragment

To import previously-exported API Gateway configuration data, perform the following steps:

1. Click the **Import Configuration Fragment** button in the Policy Studio toolbar.
2. Browse to the location of the XML file that contains the previously exported configuration data that you wish to import.
3. Select the XML file, and click **Open**.
4. If a passphrase was set on the configuration from which the data was previously exported, enter it in the **Enter Passphrase** dialog, and click **OK**.
5. In the **Import Configuration** dialog, all configuration items are selected for import by default. If you do not wish to import specific items, deselect them in the tree. For more details, see *View differences* on page 188.
6. Click **OK** to import the selected configuration items.
7. The selected configuration items are imported into your API Gateway configuration and displayed in the Policy Studio tree. For example, any imported policies and containers are displayed under the **Policies** node.

Note Be careful when deselecting configuration nodes for import. Deselecting certain nodes might make the imported configuration inconsistent by removing supporting configuration.

View differences

The **Import Configuration** dialog displays the differences between the existing stored configuration data (destination) and the configuration data to be imported (source). Differences are displayed in the tree as follows:

Addition	Exists in the source Configuration being imported but not in the destination Configuration. Displayed as a green plus icon.
Deletion	Exists in the destination Configuration but not in the source Configuration being imported. Displayed as a red minus icon.
Conflict	Exists in both Configurations but is not the same. Displayed as a yellow warning icon.

If you select a particular node in the **Import Configuration** tree, the **Differences Details** panel at the bottom of the screen shows details for this Configuration entity (for example, added or removed fields). In the case of conflicts, changed fields are highlighted.

Some Configuration entities also contain references to other entities. In this case, an icon is displayed for the field in the **Difference Details** panel. If you double-click a row with an icon, you can drill down to view further **Difference Details** dialogs for those entities.

What is imported

When configuration data is imported, some configuration items are imported in their entirety. For example, if the contents of a particular policy are different, the entire policy is replaced (new filters are added, missing filters are removed, and conflicting filters are overwritten). In addition, if a complex filter differs in its children, child items are removed and added as required (for example, WS Filter, Web service, User, and so on).

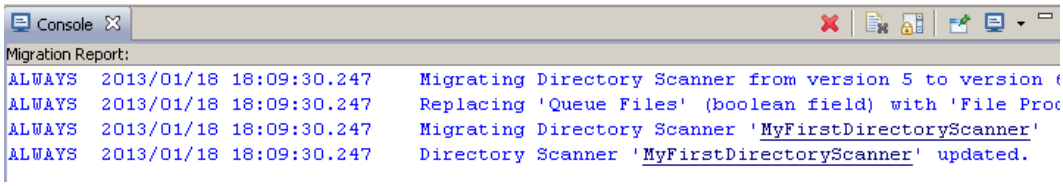
Other imports are additive only. For example, importing a single certificate does not remove the certificates already in the destination Certificate store. All references to other policies are also maintained during import.

Note Although importing some configuration items removes child items by default, you can deselect child nodes to keep existing child items. However, you should take care to avoid inconsistencies. The default selection applies in most cases.

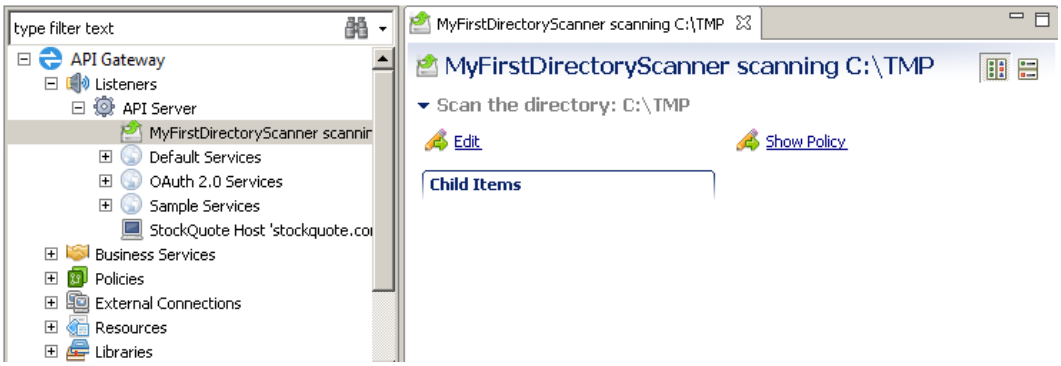
Upgrade configuration from an earlier version

When you import configuration created using an earlier version of API Gateway, the configuration is automatically upgraded to the current API Gateway version configuration. This results in the migration of the configuration entities present in the `.xml` file that is being imported.

The **Migration Report** trace console at the bottom of the window displays the migration report output that is generated when the configuration is upgraded. For example:



The **Migration Report** console also displays links that navigate to the appropriate upgraded configuration entity. For example, the following window is displayed when the `MyFirstDirectoryScanner` link is clicked:



For more details on upgrading, see the *API Gateway Upgrade Guide*.

Export API Gateway configuration

Overview

You can export API Gateway configuration data by right-clicking a Policy Studio tree node (for example, policy or policy container), and selecting the relevant export menu option (for example, **Export Policy**). The configuration is exported to an XML file, which you can then import into a different API Gateway configuration.

For example, this is useful in a development environment if you wish to share and test configuration with other developers. By exporting configuration data from one API Gateway installation, and importing into another API Gateway installation, you can effectively share your API Gateway configuration in a development environment. This also enables you to manage differences and references between configuration components.

For details on importing configuration data, see *Import API Gateway configuration fragment* on page 187.

Note For details on migrating API Gateway configuration between development, testing, and production environments, see the *API Gateway DevOps Deployment Guide*.

What is exported

You can export API Gateway configuration items by right-clicking a node in the Policy Studio tree. For example, this includes the following types of Policy Studio tree nodes:

- Policies
- Policy containers
- Schemas
- Alerts
- Caches
- Regular expressions (White list)
- Attacks (Black list)
- Users
- Certificates and Keys
- Relative paths
- Remote hosts
- Database Connections
- Server Settings

In addition, you can also export configuration items that are associated with the selected tree node. For example, this includes referenced policies, MIME types, regular expressions, schemas, and remote hosts. For details on exporting additional configuration items, see the next section.

Export configuration items

To export API Gateway configuration items, perform the following steps:

1. Right-click a Policy Studio tree node (for example, policy or policy container), and select the relevant menu option (for example, **Export Policy**).
2. The first window in the export wizard is a read-only window that displays the configuration items to be exported. The **Exporting** tree displays the selected tree node (in this case, policy), which is exported by default. **The following configuration items will also be exported** tree includes additional referenced items that are also exported by default along with the policy (for example, MIME types, regular expressions, and schemas).
3. You can click **Finish** if this selection suits your requirements. Otherwise, click **Next** to refine the selection.
4. In the next window, you can select optional configuration items for export. The **Additional configuration items that may be exported** tree on the left includes dependent items that are not exported by default. For example, these include the following:

- Outbound references: Configuration items directly referenced out from the export set to other configuration stores (for example, certificates, users, or external connections).
- Inbound references: Configuration items in other configuration stores that directly reference items in the export set.
- Associated configuration directly related to the export set (for example, remote hosts or relative paths).

5. To add an item for export, select it in the **Additional configuration that may be exported** tree on the left, and click **Add**.
6. To remove an item for export, select it in the **Additional configuration that will be exported** tree on the right, and click **Remove**.

   Note The original set of items in the **Additional configuration that will be exported** tree cannot be removed. Only items added from the **Additional configuration that may be exported** tree can be removed.

7. By default, items displayed in the **Additional configuration that may be exported** tree are scoped to direct references to the export set (inbound, outbound, and associated). You can select **Display additional configuration that depends on items to be exported** to recursively add references to this tree when additional configuration items are added to the export set.
8. Click **OK** to export the selected configuration.

Referenced Policies

When exporting a policy or policy container, by default, any policies referenced by the policy are included for export and displayed in the **Additional configuration that will be exported** list.

Compliance validation tools

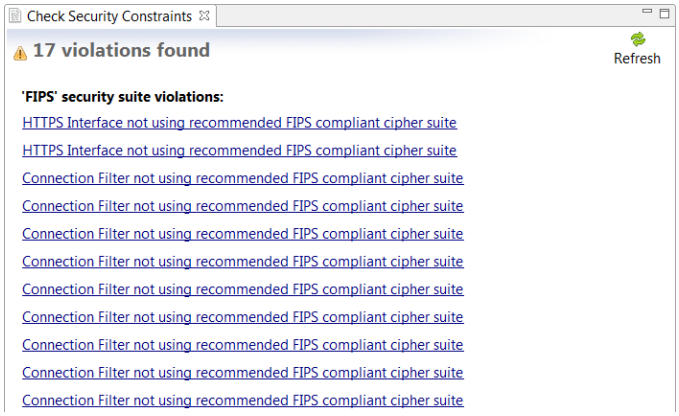
Overview

Policy Studio contains validation tools for checking the compliance of your configuration against the following security standards:

- Federal Information Processing Standards (FIPS) 140-2
- National Institute of Standards and Technology (NIST) Suite B
- National Institute of Standards and Technology (NIST) Suite B Top Secret

Validate FIPS compliance

You can run the FIPS Compliance Validation Tool to check that a configuration is FIPS compliant. This tool is available from the **Tools > Check Security Constraints > FIPS** menu option. When you run the tool, it generates a list of violations, for example:



Click the link for a violation to go to the part of the configuration that is not compliant. For guidance on compliant settings for filters, see the *API Management Security Guide*.

Validate Suite B compliance

You can run the Suite B Compliance Validation Tool to check that a configuration is Suite B compliant. This tool is available from the **Tools > Check Security Constraints > SuiteB** menu option. When you run the tool, it generates a list of violations. Click the link for a violation to go to the part of the configuration that is not compliant. For guidance on compliant settings for filters, see the *API Management Security Guide*.

Validate Suite B Top Secret compliance

You can run the Suite B Top Secret Compliance Validation Tool to check that a configuration is Suite B compliant. This tool is available from the **Tools > Check Security Constraints > SuiteBTS** menu option. When you run the tool, it generates a list of violations. Click the link for a violation to go to the part of the configuration that is not compliant. For guidance on compliant settings for filters, see the *API Management Security Guide*.

Upgrade log analysis

Policy Studio provides graphical features to help you detect and analyze upgrade issues. You can perform the following in Policy Studio:

- If `sysupgrade` has identified issues during the upgrade (for example, web service configuration issues), you can use the **Tools > Upgrade Log Analysis** option in Policy

Studio to analyze the `upgrade.log` file. See *Manual upgrade log analysis* on page 193.

- If you create a new project in Policy Studio from an existing configuration, or if you import a configuration fragment, an automatic upgrade is triggered and the **Upgrade Log Analysis** window opens. This allows you to view the log and resolve any issues. See *Automatic upgrade log analysis* on page 194.

This topic describes both cases with examples of how to resolve critical upgrade issues in Policy Studio.

Manual upgrade log analysis

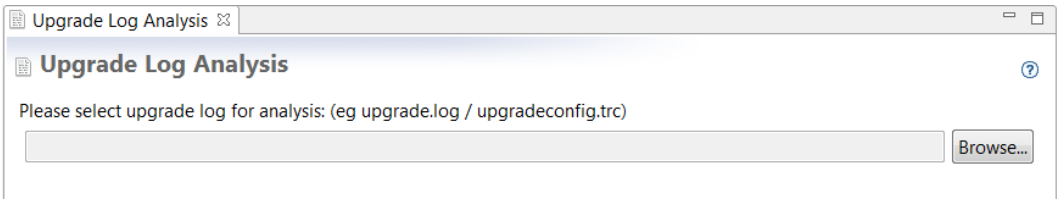
The `sysupgrade upgrade` command might generate errors or warnings that need to be resolved in Policy Studio. This includes, for example, issues with web service or REST API configuration, or with OAuth credentials. For more details, see *Example critical/major issues and recommended solutions* on page 196.

When prompted to perform upgrade log analysis in Policy Studio, perform the following steps:

1. Open the problematic API Gateway configuration in Policy Studio. For example, in Policy Studio version 7.6.2, create a new project using **From .fed file** or **From .pol and .env files**, and select the correct files under the following directory:

```
/opt/Axway-
7.6.2/apigateway/upgrade/bin/out/upgrade/esgroups/groups/group-2/
```

2. Select **Tools** > **Upgrade Log Analysis** from the main menu.



3. Select the `upgrade.log` file generated by `sysupgrade`, for example:

```
/opt/Axway-7.6.2/apigateway/upgrade/bin/out/log/upgrade.log
```

4. Click **Open**.
5. If the log file contains data on multiple groups, select the group configuration to analyze in the dialog.

   Note The `upgrade.log` file generated by `sysupgrade` can contain upgrade information for multiple groups, whereas the automatic upgrades triggered by Policy Studio are specific to the project upgraded.

6. In the **Upgrade Log Analysis** pane, click **View Log**. For more details, see *Example upgrade log analysis* on page 194.

Automatic upgrade log analysis

An automatic upgrade can be triggered in Policy Studio when you create a new API Gateway project from an existing:

- API Gateway configuration package (`.fed`, `.pol` or `.env` file)
- API Gateway configuration directory

You can also trigger an automatic upgrade by importing an API Gateway configuration fragment (**File** > **Import Configuration Fragment**).

In either case, in the **Upgrade Log Analysis** pane, click **View Log**. For more details, see *Example upgrade log analysis* on page 194.

Policy Studio-triggered upgrade logs are stored under the `policystudio/trace` directory and are prefixed with the project name or configuration fragment name, for example:

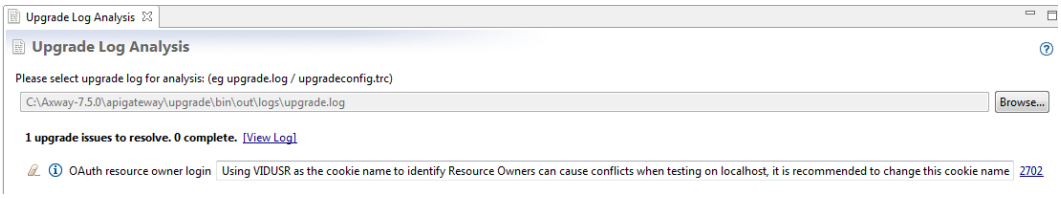
- `webservice_fragment_upgrade_20160317125324.log`
- `my_project_upgrade_20160417125319.log`

For more details on creating API Gateway projects and importing configuration fragments, see *Create a Policy Studio project* on page 31 and *Import API Gateway configuration fragment* on page 187.

Example upgrade log analysis

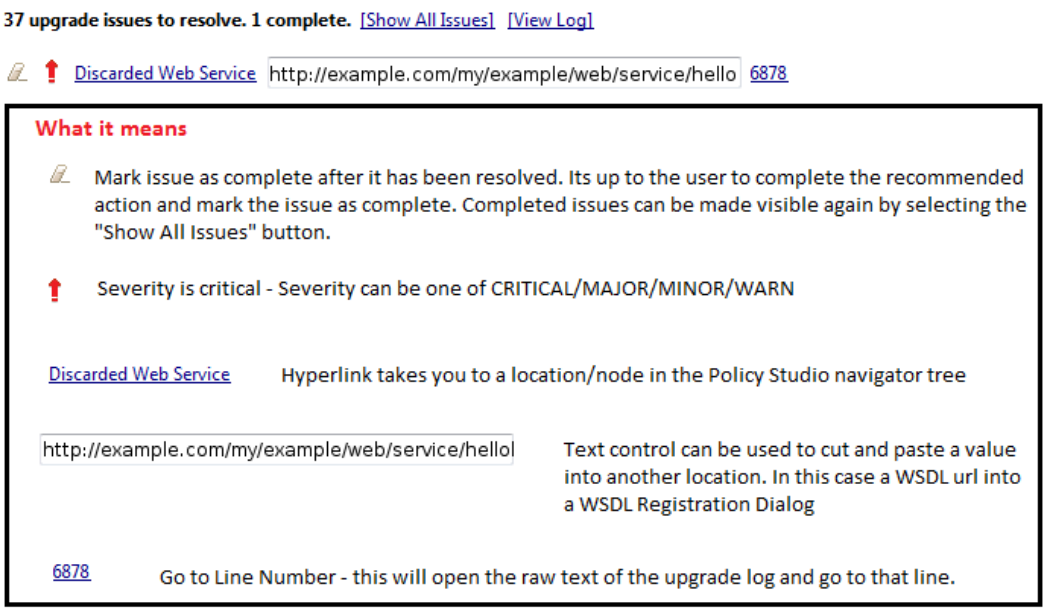
When upgrading an earlier version API Gateway configuration to version 7.6.2, the **Upgrade Log Analysis** pane on the right lists the upgrade issues that need to be resolved.

For example, when upgrading a version 7.3.1 API Gateway configuration, the following example shows an OAuth-specific issue and its recommended solution.



You can click the eraser icon on the left of each issue to mark the issue as complete when it has been resolved.

The following example shows a critical web service issue encountered when upgrading a version 7.2.2 configuration:



Tip To see this informational window in Policy Studio, hover over an issue.

In this case, the web service could not be upgraded, and has been discarded. You must reimport the WSDL into your new API Gateway7.6.2 installation using Policy Studio. For more details, see *Example critical/major issues and recommended solutions* on page 196.

Upgrade indicators in Policy Studio tree

You can view the critical upgrade issues for web services (for example, the **Discarded Web Services**, or WSDL files using an **Old Repository Format**) as text in the Policy Studio tree on the left. If you close the **Upgrade Log Analysis** view or ignore the issues, the indicator text is no longer displayed.

View more details in upgrade log

To view more details on a specific upgrade issue in the right pane, click **View Log** to display the relevant entry in the upgrade log file.

Restore completed issues

All completed issues are persisted to disk. This means that you can shut down Policy Studio or the **Upgrade Log Analysis** view and continue later. To restore completed issues to the view at any time, click **Show All Issues** in the right pane.

Example critical/major issues and recommended solutions

This section describes some critical and major upgrade issues displayed by Policy Studio and provides recommended solutions.

Discarded web service

Problem: The WSDL file for the web service could not be upgraded.

Solution: You must re-register the WDSL in the web service repository. Perform the following steps:

1. Copy the original WSDL URL displayed in Policy Studio to the clipboard.
2. Click the **Discarded web service** hyperlink to locate the web service group it was originally registered in.
3. On that web service group, right-click and select **Register Web Service**.
4. In the WSDL registration wizard, paste the URL in the **WSDL URL** field.
5. Click **Next** and complete the wizard to register the WSDL.
6. Click the **Mark this item as complete** icon in the **Upgrade Log Analysis** window.

Old repository format

Problem: The WSDL file for the web service uses an old web service repository format (versions earlier than API Gateway 7.3.0).

Solution: You must re-register the WDSL in the web service repository. Perform the following steps:

1. Click the **Old repository format** hyperlink to locate the web service to be re-registered.
2. Right-click the web service, select **Resynchronize Web Service**, and select **Yes**.
3. Copy the original WSDL URL to the clipboard, and select **Cancel** from the wizard.
4. Right-click the web service node, select **Delete**, and select **Yes**.
5. On that web service group, right-click, and select **Register Web Service**.
6. In the WSDL registration wizard, paste the URL in the **WSDL URL** field.
7. Click **Next** and complete the wizard to register the WSDL.
8. Click the **Mark this item as complete** icon in the **Upgrade Log Analysis** window.

Web Service error: Failed to find binding for operation

Problem: The WSDL file the web service has no binding for a particular operation.

Solution: You must re-register the WDSL in the web service repository. Perform the steps listed in *Old repository format* on page 196.

OAuth Client Application Registry

Problem: A new mechanism for authenticating OAuth client application users in the REST servlet was introduced in API Gateway version 7.4.0. This issue only occurs if you are upgrading from versions earlier than API Gateway 7.4.0.

Solution: You must configure OAuth authentication settings in Policy Studio. Click **Server Settings** in the Policy Studio tree and select **Security** > **Client Application Registry**. In the **Circuit for Authentication** field, select a policy for authenticating users. This policy must authenticate the user against a user store of your choice and set the message attributes `authentication.subject.role` and `user.email`.The message attribute `authentication.subject.role` must be set to either `admin` or `resourceowner`. For more information on the Client Application Registry settings, see the *API Gateway OAuth User Guide*.

A sample policy for authenticating users is provided in `apigateway/samples/oauth/authzserver/sampleUserAuthnPolicy.xml`. This sample policy authenticates the user credentials against the local user store and sets the two required attributes on the message whiteboard, which are used to complete authentication and authorization process. If you use the sample policy, you must manually add a role (for example, `role=admin`) for the `regadmin` and `sampleuser` users. To add the role in Policy Studio, select **Environment Configuration > Users and Groups > Users** and add the role on the **Attributes** tab for each user. For more information on adding users, see the *API Gateway Administrator Guide*.

Migrated resolver for web service

Problem: In API Gateway 7.6.2, `WebService` resolvers are migrated to path resolvers during upgrade, because the `WebService` resolver had issues if the web service was renamed.

Solution: The migration to the path resolver removes this issue. The upgrade log contains the following minor warning if this migration affects a web service:

```
StockQuoteService Migrated resolver on path '/svcpath'. Confirm path is correct.
```

In some cases, the migration of the `WebService` resolver results in a conflict with another path resolver. In this case, the path resolver is renamed to avoid conflict. You are alerted to this with the following critical warning:

```
StockQuoteService Migrated resolver on path '/svcpath' was renamed to '/*** Path
Conflict ‘/svcpath’' to avoid conflict. Please update appropriately.
```

You must enter a new unique path.

Discarded Contivo filter

Problem: The **Contivo** filter is not supported in API Gateway 7.5.1.

Solution: We recommend that you replace this filter with the **Execute Data Map** filter.

Example warnings and recommended solutions

This section describes some warnings displayed by Policy Studio and provides recommended solutions.

Fastest scripting language not used

Problem: A warning appears if you are using a **Scripting Language** filter in your old installation with the **Language** field set to `JavaScript (Rhino engine JRE7 and earlier)`.

Solution: Change the **Language** of the filter to `JavaScript` and ensure that the JavaScript syntax in the script conforms with Nashorn engine syntax. Alternatively, you can ignore the warning, meaning that the script continues to work but not with the optimal performance.

Cassandra connection details

Problem: A warning about the Cassandra connection details appears when you are upgrading from API Gateway versions earlier than 7.5.1 to 7.6.2.

If you upgrade your entity store configuration using `sysupgrade`, you can set the Cassandra connection details for the 7.6.2 installation using the `upgrade` command options `--cass_host`, `--cass_port`, `--cass_username` and `--cass_password`. If you have done this correctly, you can ignore the warning.

If you upgrade your entity store configuration in Policy Studio, the Cassandra settings are set to the default settings (for example, `localhost`).

Solution: After the upgrade, ensure that the Cassandra connection details are correct.

ActiveMQ path update

Problem: A warning appears if ActiveMQ is enabled in your old installation, reminding you to update the **Shared Directory** field if necessary. Upgrading your entity store in Policy Studio does not automatically update the **Shared Directory** field or backup the data. This warning indicates you might need to take some actions.

If you upgrade your entity store configuration using `sysupgrade`, and the **Shared Directory** field is an absolute path under the old installation directory (for example, `/opt/Axway/7.3.1/apigateway/activemq`), `sysupgrade` automatically updates the directory to be the equivalent directory under the new installation directory (for example, `/opt/Axway/7.6.2/apigateway/activemq`), and copies the data over to the new installation.

If you upgrade your entity store configuration using `sysupgrade`, and the **Shared Directory** field is an absolute path that is unrelated to the installation directory (for example, `/mynetworkdrive/activemq`), `sysupgrade` backs up the data, but does not update the **Shared Directory** field.

Solution: After the upgrade, ensure that the path defined in the **Shared Directory** field for ActiveMQ is not an absolute path that can be accessed by both the old and new installations of API Gateway. You can ignore the warning if the field points to a directory that is not shared between the old and new installations, for example, any non-absolute path.

Default SSL ciphers/options not used

Problem: If an API Gateway group or Node Manager configuration contains an SSL interface that has old SSL settings (default ciphers and SSLv2/v3 allowed), a warning appears suggesting that you reconfigure the SSL settings. Similar warnings are generated for system ports and custom ports.

Solution: Reconfigure the affected HTTPS listeners to use the recommended ciphers and protocol settings. Alternatively, you can ignore these warnings.

Amazon Web Services filters

Problem: A warning appears if an Amazon Web Services filter without a setting for either **AWS Credential** or **Client setting**s is found.

Solution: During the upgrade, dummy settings are added to affected AWS filters. A warning message indicates the affected filters. Replace the dummy settings with valid values, so the filters work correctly.

API Gateway KPS collection

Problem: A warning appears when configuration from an old installation does not contain the KPS schema required by the OAuth Client Application Registry. This warning is only relevant if the configuration is imported directly through Policy Studio and has not been upgraded using the `sysupgrade` command.

Solution: You can import the required schema from the configuration fragment in `samples/oauth/authzserver/APIServerKPSDefinition.xml`.

KPS table PortalExternalClientStore missing

Problem: A warning appears when configuration from an old installation does not contain the KPS table PortalExternalClientStore. This table is required by API Manager. This warning is only relevant if the configuration is imported directly through Policy Studio and has not been upgraded using the `sysupgrade` command.

Solution: You can import the required schema from the configuration fragment in `samples/oauth/authzserver/APIServerKPSDefinition.xml`.

OAuth resource owner login

Problem: This warning appears if an old installation is configured to use the same cookie name for both OAuth client and OAuth server configurations. It is a minor issue that only affects test environments where client and server share the same host name (for example, `localhost`).

Solution: Change the cookie name for OAuth logins on either the client or the server.

OAuth services interface

Problem: This warning appears when multiple listeners implementing OAuth services are found in the old configuration. Upgrade can only successfully upgrade canonical listeners that follow the naming construct of `OAuth 2.0 Services`.

Solution: Ensure that the Servlet and Path definitions of additional OAuth listeners match those of the default listener `OAuth 2.0 Services`. If the standard listener definition is not in the configuration, it can be imported from `samples/oauth/authzserver/config.xml`.

OAuth KPS definitions missing

Problem: A warning appears when configuration from an old installation does not contain the KPS schema required for OAuth token storage. This warning is only relevant if the configuration is imported directly through Policy Studio and has not been upgraded using the `sysupgrade` command.

Solution: You can import the required schema from the configuration fragment in `samples/oauth/authzserver/OAuthTokenKPSDefinition.xml`.

OAuth authorizations table missing

Problem: A warning appears when configuration from an old installation does not contain the KPS schema required for OAuth authorization storage.This warning is only relevant if the configuration is imported directly through Policy Studio and has not been upgraded using the `sysupgrade` command.

Solution: You can import the required schema from the configuration fragment in `samples/oauth/authzserver/OAuthTokenKPSDefinition.xml`.

Oracle Security Service Module settings (10g)

Overview

An Oracle Security Service Module (SSM) integrates a secured application (in this case, the API Gateway) with an Oracle Entitlements Server (OES) 10g so that security administration (for example, roles, resources, and policies) is delegated to the Oracle Entitlements Server 10g. An SSM must be installed on the machine hosting the application to be secured by the Oracle Entitlements Server 10g. The SSM runs in-process with the secured application, which improves performance and on-the-wire security.

In Policy Studio, select the **Environment Configuration** > **Server Settings** node in the tree, and select **Security**> **Security Service Module** in the right pane. The **Security Service Module** settings enable you to configure the API Gateway to act as a Java SSM. For more details on Oracle Entitlements Server 10g and SSMs, see the Oracle Entitlements Server website.

Note Oracle SSM is required only for integration with Oracle OES 10g. Oracle SSM is not required for integration with Oracle OES 11g. OES 10g was previously known as BEA AquaLogic Enterprise Security (ALES). Some items, such as schema objects, paths, and so on, may still use the ALES name.

Prerequisites

Before configuring the settings on the **Security Service Module** tab, you must perform the following prerequisite tasks.

Test the SSM installation

Because the API Gateway is running a Java SSM internally, it is recommended that the example Java SSM client that ships with the OES installation is set up and configured. This example can be found in the following directory:

```
/ales32-ssm/java-ssm/examples/JavaAPIExample
```

Follow the instructions in the README file in this directory to test the installation. When the testing of the `JavaAPIExample` is complete, all the configuration files for an SSM instance are located in the `/ales32-ssm/java-ssm/SSM-Name` directory, where `SSM-Name` is the name of the SSM setup when testing the example.

Configure the API Gateway classpath

The API Gateway classpath must be updated to include the JARs and configuration files for the SSM instance. The `jvm.xml` file must be updated so that various environment variables and the `SSM-Name` are updated to reflect the installation of the Java SSM. At minimum, the following must be updated in `jvm.xml`:

```
<Environment name="BEA_HOME" value="/opt/apps/bea" >
<Environment name="INSTANCE_NAME" value="SSM-Name" >
```

For example, to modify the classpath, place the following `jvm.xml` in the `conf` directory of the API Gateway installation:

```
<!--Additional JVM settings to run with Oracle Entitlements Server BEA_HOME must be
```

```
set to the location
  where the SSM is installed -->
<ConfigurationFragment>
  <!-- Environment variables -->
  <!-- change these to match location where SSM has been installed and configured --
>
  <Environment name="BEA_HOME" value="/opt/apps/bea" />
  <Environment name="ALES_SHARED_HOME" value="$BEA_HOME/ales32-shared" />
  <!-- Name of the SSM running in the API Gateway, replace the "SSM-Name" with the
name of the SSM for
    the API Gateway -->
  <Environment name="INSTANCE_NAME" value="SSM-Name" />
  <Environment name="INSTANCE_HOME" value="$BEA_HOME/ales32-ssm/java-ssm/instance/
$INSTANCE_NAME" />
  <Environment name="PDP_PROXY" value="$INSTANCE_HOME/pdpproxy" />
  <!-- Location of the Java SSM libraries -->
  <!-- <ClassDir name="$BEA_HOME" /> -->
  <ClassDir name="$BEA_HOME/ales32-ssm/java-ssm/lib" />
  <ClassDir name="$BEA_HOME/ales32-ssm/java-ssm/lib/providers/ales" />
  <!-- Add location of the SSM configuration to classpath -->
  <ClassPath name="$INSTANCE_HOME/config/" />
  <!-- Additional JVM parameters based on the %JAVA-OPTIONS% of set-env script in
SSM instance running
    in API Gateway $BEA_HOME/ales32-ssm/java-ssm/instance/ssm-name/config -->
  <VMArg name="-Dwles.scm.port=7005" />
  <VMArg name="-Dwles.arme.port=8000" />
  <VMArg name="-Dwles.config.signer=Oracle Entitlements Serverdemo.oracle.com" />
  <VMArg name="-Dlog4j.configuration=file:$INSTANCE_HOME/config/log4j.properties" />
  <VMArg name="-Dlog4j.ignoreTCL=true" />
  <VMArg name="-Dwles.ssl.passwordFile=$ALES_SHARED_HOME/keys/password.xml" />
  <VMArg name="-Dwles.ssl.passwordKeyFile=$ALES_SHARED_HOME/keys/password.key" />
  <VMArg name="-Dwles.ssl.identityKeyStore=$ALES_SHARED_HOME/keys/identity.jceks" />
  <VMArg name="-Dwles.ssl.identityKeyAlias=wles-ssm" />
  <VMArg name="-Dwles.ssl.identityKeyPasswordAlias=wles-ssm" />
  <VMArg name="-Dwles.ssl.trustedCAKeyStore=$ALES_SHARED_HOME/keys/trust.jks" />
  <VMArg name="-Dwles.ssl.trustedPeerKeyStore=$ALES_SHARED_HOME/keys/peer.jks" />
  <VMArg name="-Djava.io.tmpdir=$INSTANCE_HOME/work/jar_temp" />
  <VMArg name="-Darme.configuration=$INSTANCE_HOME/config/WLESarme.properties" />
  <VMArg name="-Dales.blm.home=$INSTANCE_HOME" />
  <VMArg name="-Dkodo.Log=log4j" />
  <VMArg name="-Dwles.scm.useSSL=true" />
  <VMArg name="-Dwles.providers.dir=$BEA_HOME/ales32-ssm/java-ssm/lib/providers" />
  <VMArg name="-Dpdp.configuration.properties.location=$PDP_PROXY/
PDPProxyConfiguration.properties" />
</ConfigurationFragment>
```

Centralize all trace output

Oracle's Java SSM uses log4j to output any diagnostics. You can also add these messages to the API Gateway trace output by adding the log4j that ships with the API Gateway to the following file:

```
/ales32-ssm/java-ssm/SSM-NAME/conf/log4j.properties
```

Then the `log4j.rootCategory=WARN, A1, ASIlogFile` line includes a new appender called `VordelTrace` as follows:

```
log4j.rootCategory=WARN, A1, ASIlogFile, VordelTrace
```

Add the configuration for this new appender by adding the following line to the file:

```
log4j.appender.VordelTrace=com.vordel.trace.VordelTraceAppender
```

You can now start the API Gateway so that it runs with the Java SSM classpath and the centralized trace output.

Further information

For more details on configuring and testing SSMs, see the *Oracle SSM Installation and Configuration Guide*.

Settings

On the **Security Service Module** settings window, configure the following fields on the **Settings** tab:

Enable Oracle Security Service Module:
Select whether to enable the API Gateway instance to act as an SSM. This setting is disabled by default.

Application Configuration Name:
Enter the Application Configuration name for the SSM instance. This is the name of your runtime application used by OES (for example, for monitoring purposes).

Configuration Name:
Enter the OES Configuration name for the SSM instance to be stored in the OES Configuration Repository. Configuration names share the same name as their Policy Domain names.

Application Configuration Properties:
Click **Add** to specify optional configuration properties as name-value pairs. Enter a **Name** and **Value** in the **Properties** dialog. Repeat to specify multiple properties.

Policy Domain Name:
Enter the OES Policy Domain name for the SSM instance. Policy Domains contain policy definitions (target resource, permission set, and policy). Policy Domain names share the same name as their Configuration names.

Name authority definition settings

Configure the following field on the **Name Authority Definition** tab:

Name Authority Definition File:
Click the **Browse** button at the bottom right to configure the Name Authority Definition file for the SSM. This is an XML file that specifies the naming authority definition required for the API Gateway. For example, a specified XML file named `apigatewayNameAuthorityDefinition.xml` file should contain the following settings:

```
<AuthorityConfig>
  <AuthorityDefinition name="apigatewayResource" delimiters="/\">
    <Attribute name="protocol" type="MULTI_TOKEN" authority="URLBASE" />
  </AuthorityDefinition>
  <AuthorityDefinition name="apigatewayAction" delimiters="/">
    <Attribute name="action" type="SINGLE_VALUE_TERMINAL" />
  </AuthorityDefinition>
</AuthorityConfig>
```

Kerberos configuration

The **Kerberos Configuration** under **Server settings > Security > Kerberos** in the node tree enables you to configure instance-wide Kerberos settings on API Gateway and to upload a Kerberos configuration file to API Gateway. This configuration file contains information on the location of the Kerberos Key Distribution Center (KDC), as well as which encryption algorithms, encryption keys, and domain realms to use.

You can also configure trace options for the various APIs used by the Kerberos system, such as the Generic Security Services (GSS) and Simple and Protected GSS-API Negotiation (SPNEGO) APIs.

Linux platforms ship with a native implementation of the GSS library, which API Gateway can leverage. You can specify the location of the GSS library in this configuration window.

For more details on different Kerberos setups with API Gateway, see *API Gateway Kerberos Integration Guide*.

Kerberos configuration file – krb5.conf

The Kerberos configuration file (`krb5.conf`) defines the location of the Kerberos KDC, supported encryption algorithms, and default realms in the Kerberos system. Both Kerberos clients and Kerberos services that are configured for API Gateway use this file.

Kerberos clients need to know the location of the KDC so that they can obtain a Ticket Granting Ticket (TGT). They also need to know what encryption algorithms to use and what realm they

belong to. Kerberos services do not need to call the KDC to request a TGT, but they still require the information on supported encryption algorithms and default realms contained in the `krb5.conf` file.

A Kerberos client or service identifies the realm it belongs to because the realm is appended to its Kerberos principal name after the `@` symbol. Alternatively, if the realm is not specified in the principal name, the Kerberos client or service assumes the realm to be the `default_realm` specified in the `krb5.conf` file. The file specifies only one `default_realm`, but you can specify a number of additional named realms. The `default_realm` setting is in the `[libdefaults]` section of the `krb5.conf` file. It points to a realm in the `[realms]` section. This setting is not required. ## (what does this mean? which setting is not required?)

The text input field in the Kerberos configuration window displays a default configuration for `krb5.conf`. You can type and modify the configuration as needed, and then click **OK** to upload it to your API Gateway configuration. Alternatively, if you have an existing `krb5.conf` file that you want to use, select **Load File** and open to the configuration file. The contents of the file are displayed in the text area, and you can edit and upload it to API Gateway.

Note Refer to your Kerberos documentation for more information on the settings that can be configured in the `krb5.conf` file.

Advanced settings

You can configure various tracing options for the underlying Kerberos API using the check boxes on the **Advanced settings** tab. Trace output is always written to the `/trace` directory of your API Gateway installation.

- **Kerberos Debug Trace**– Enables extra tracing from the Kerberos API layer.
- **SPNEGO Debug Trace** – Switches on extra tracing from the SPNEGO API layer.
- **Extra Debug at Login**– Provides extra tracing information during login to the Kerberos KDC.

Native GSS library

The Generic Security Services API (GSS-API) is an API for accessing security services, including Kerberos. Implementations of the GSS-API ship with the Linux platforms and can be leveraged by API Gateway when it is installed on these platforms. The fields on this tab allow you to configure various aspects of the GSS-API implementation for your target platform.

Use Native GSS Library:
Select this to use the operating system's native GSS implementation. This option only applies to API Gateway installations on the Linux platforms.

Note These are instance-wide settings. If you select **Use Native GSS Library**, it is used for all Kerberos operations, and all Kerberos clients and services must be configured to load their credentials natively.
If the native library is used, the following features are not supported:

- The SPNEGO mechanism
- The WS-Trust for SPNEGO standard (requires the SPNEGO mechanism)
- The SPNEGO over HTTP standard (requires the SPNEGO mechanism)
- Signing and encrypting using the Kerberos session keys

It is possible to use the KERBEROS mechanism with the SPNEGO over HTTP standard, but this would be non-standard.

Native GSS Library Location:
If you have opted to use the native GSS library, enter the location of the GSS library in the field provided, for example, `/usr/lib/libgssapi.so`. On Linux, the library is called `libgssapi.so.`.

Note This setting is only required when this library is in a non-default location.

Native GSS Trace:
Use this option to enable debug tracing for the native GSS library.

Sun Access Manager settings

Note This feature has been deprecated and will be removed in a future release. See Oracle Access Manager filters.

The **Access Manager** settings window enables you to configure how the access manager policy agents embedded in API Gateway's Sun Access Manager filters connect to Sun Access Manager. These settings also enable you to determine how and where these agents trace and log runtime information.

The access manager settings are available from the **Environment Configuration > Server Settings** node in the Policy Studio tree. Select the **Security > Access Manager** tab at the bottom of the window. For a more detailed explanation of any of the settings described in this topic, see the Sun Access Manager documentation.

Connection settings

The following configuration fields are available in this section:

Naming URL:
This property represents the URL where the access manager filters can retrieve the URLs of access manager internal services. This field sets the `com.iplanet.am.naming.url` property.

User name:
Specify the user name to read configuration data from the access manager. This sets the `com.sun.identity.agents.app.username` property.

Password:
Enter the password for this user. This setting configures the `com.iplanet.am.service.password` property.

Output settings

The fields in this section are used to configure the output from the access manager policy agent that is embedded into API Gateway's Sun Access Manager integration filters. In most cases, the default options should be sufficient.

Debug level:
This setting specifies the level that the access manager agent writes debug information at. Possible values are `message`, `warning`, `error`, and `off`. This setting corresponds to the `com.iplanet.services.debug.level` property.

Debug directory:
Specify the location of the directory where debug output is written to. This configures the `com.iplanet.services.debug.directory` property.

Log file name:
Enter the name of the log file where policy decision information is written to. This sets the `com.sun.identity.agents.server.log.file.name` property.

Logging level:
Select the logging level for entries in the log file from the following available levels: `NONE`, `ALLOW`, `DENY`, `BOTH`, or `DECISION`. The value selected sets the value of the `com.sun.identity.agents.logging.level` property.

General settings

The following general settings are available in this section:

Cache time:
The value entered can be used to determine how long (in minutes) configuration data is cached before it is fetched again from the access manager server. This setting corresponds to the `com.sun.identity.sm.cacheTime` property.

Polling interval:
The polling interval entered represents the time (in minutes) after which the user management cache is updated. This field sets the value of the `com.iplanet.am.sdk.remote.pollingTime` property.

Additional properties

This section enables you to specify any additional properties used by the Sun Access Manager Client SDK as name-value pairs. Click the **Add** button to enter the name and value of the required property. For more information on available client SDK properties, see the Sun Access Manager documentation.

Manage deployments 6

This section describes how to manage API Gateway deployments.

Manage API Gateway deployments

You can use Policy Studio to deploy configuration to API Gateway instances running in groups in an API Gateway domain. Policy Studio enables you to edit API Gateway configuration and then deploy it to the server instance, where it can be reloaded later. You can deploy modified configuration to multiple API Gateway instances in a group managed by an Admin Node Manager.

The API Gateway Manager web console also enables you to deploy configuration packages to API Gateway instances running in groups in a domain, to create groups and API Gateway instances, and to manage administrator users. In this way, Policy Studio and the API Gateway Manager enable policy developers and administrators to centrally manage the policies that are enforced at all nodes throughout the network.

In addition, Policy Studio enables you to compare and merge differences between versions of the same policy. Policies can be merged, and deployed to any running instance that is managed by Policy Studio. One of the most powerful uses of this centralized management capability is in transitioning from a staging environment to a production environment. For example, policies can be developed and tested on the staging environment, and when ready, they can be deployed to all instances deployed in the production environment.

Create a project in Policy Studio

Note Before starting Policy Studio, you should first ensure that the Admin Node Manager and the server instance that you wish to deploy to have been started.

To create a new Policy Studio project, select **File** > **New Project**, and follow the steps in the wizard. For more details, see *Create a Policy Studio project* on page 31.

Alternatively, if a project has already been created, select **File** > **Open Project** in the main menu, or click **Open Project** on the landing page. For more details, see *Manage API Gateway connections* on page 172.

Edit a project configuration in Policy Studio

When you create or open a Policy Studio project and make a server connection, this loads the project configuration and displays it in the following format:

ProjectName [ServerInstanceType]

For example:

MyDevProject [API Gateway]

When a project configuration is loaded, its services are displayed in the Policy Studio tree on the left. Expand one of the top-level nodes in the tree to display additional details (for example, **APIs**, **Policies**, **Resources**, or **Environment Configuration**).

When editing a project configuration, you can deploy updates using the **Deploy** button in the toolbar (alternatively, press **F6**). For more details, see *Deploy API Gateway configuration* on page 210.

Deploy to a server in Policy Studio

To deploy to a running API Gateway instance in a group, click **Deploy** in the toolbar, and follow the steps in the wizard. For more details, see *Deploy API Gateway configuration* on page 210.

Tip You must connect to the Admin Node Manager server to deploy API Gateway configuration or manage multiple API Gateway instances in your network.

Manage deployments in API Gateway Manager

In the web-based API Gateway Manager tool, the **TOPOLOGY** section on the **Dashboard** tab enables you to create API Gateway groups and instances, and to deploy configuration packages to running servers in API Gateway groups.

For details on how to access the API Gateway Manager, see *Start the API Gateway tools* on page 30.

Compare and merge configurations in Policy Studio

You can compare and merge differences between the currently loaded API Gateway configuration with a configuration stored in a deployment package (.fed file). Click the **Compare** button on the Policy Studio toolbar to select a `.fed` file to compare the current configuration against. The results are displayed in the **Compare/Merge** tab.

For example, you can view the differences made to particular fields in an Authentication filter that occurs in both configurations. When a difference is located, you can merge the differences, and thereby update the fields in the Authentication filter in the current configuration with the field values for the same Authentication filter in the deployment package.

For more details, see *Compare and merge API Gateway configurations* on page 214.

Manage administrator users in API Gateway Manager

You can add new administrator users to enable role-based access to the API Gateway configuration managed by Policy Studio and API Gateway Manager. The default administrator user has access to all API Gateway features in Policy Studio and API Gateway Manager, and can view and modify all API Gateway configurations.

To add or remove administrator users, click the **Settings > Admin Users** tab in the API Gateway Manager. For more details, see *Manage admin users* on page 216.

For more details on role-based access, see the *API Gateway Administrator Guide*.

Configure policies in Policy Studio

You can use Policy Studio to manage the configuration of your policies, which can then be deployed to running instances of Axway API Gateways.

For details on configuring the full range of message filters (for example, for Authentication, Authorization, or Content Filtering), see the other sections in this guide.

Deploy API Gateway configuration

You can edit API Gateway configuration in a Policy Studio project, and deploy to specified API Gateway instances running in an API Gateway group. You can deploy projects based on existing configuration, configuration packages, factory configuration, or a running API Gateway instance.

Policy Studio also enables you to create configuration packages (`.fed`, `.pol`, or `.env` files), and to deploy projects based on configuration packages to API Gateway instances.

You can also deploy API Gateway configuration packages in the API Gateway Manager web console. Alternatively, you can use the `managedomain` script to create and deploy deployment packages (`.fed` files) on the command line.

Deploy configuration in Policy Studio

You can deploy updates to a currently loaded configuration when editing the configuration in Policy Studio. To deploy a currently loaded configuration, perform the following steps:

1. Click the **Deploy** button on the right in the toolbar.
2. In the **Open Connection** dialog, in the **Saved Session**s section, select the server session to use from the list. You can edit a session name by entering a new name and clicking **Save**. You can also click the appropriate button to **Add**, **Clone**, or **Remove** saved sessions.

3. In the **Connection Details** section, configure the following:
   - **Host**:
     Enter the server host to connect to. The default is `localhost`.
   - **Port**:
     Enter the port to connect on. The default Admin Node Manager port is `8090`.
   - **User name**:
     The deployment service is protected by HTTP basic authentication. Enter the administrator user name to use to authenticate to the server. For more details, see *Manage admin users* on page 216.
   - **Password**:
     Enter the password for the administrator user.
4. Click **Advanced** to enter the **URL** of the deployment service exposed by the server. This setting is optional. The default Admin Node Manager URL is `https://localhost:8090/api`.
5. Click **Next** to configure deployment options.

   If an advisory warning has been configured, you must click **Next** again. For more details, see the *API Gateway Administrator Guide*.
6. In the **Select the servers(s) you wish to deploy to** section, select an API Gateway group from the **Group** list, and select the server instance(s) in the box below.

   If the server uses a different API Gateway encryption passphrase for its environment, click **Advanced**, select **The target server uses a different passphrase**, and enter the **Passphrase** used by the target server.
7. Click **Next**, and wait for the deployment to complete.
8. Click **Finish**.

Note You must connect to the Admin Node Manager server to deploy API Gateway configuration or manage multiple API Gateway instances in your network.

View deployment results in Policy Studio

When you click **Deploy**, the **Deployment Results** screen is displayed, and deployment to each server occurs sequentially. Feedback is provided using icons in the **Task** column, and text in the **Status** column. When the configuration has deployed, click **Finish**.

Cancel deployments

You can cancel deployments by clicking the **Cancel** button. Feedback is provided in the **Status** column. You cannot cancel a deployment when it has started. The wizard performs the cancellation at the end of the current deployment, with all remaining deployments being canceled.

Deployment errors

Client-side and server-side errors can occur. Client-side errors are displayed in the **System Trace** in the **Console** view. If any server-side deployment errors occur during the deployment process, you can review these in the **Deployment Error Log** view. This is displayed at the bottom of the screen when you click **Finish**, and lists any errors that occur for each instance. The corresponding Console **Deployment Log** is also available in the **Console** view.

Redeploy

When you have deployed a configuration to one or more instances, you can click back through the wizard to change your selections and redeploy, without needing to exit and relaunch the wizard.

API Gateway configuration packages

You can deploy configuration based on API Gateway configuration packages in Policy Studio and in the API Gateway Manager web console. API Gateway includes the following types of configuration package:

- A *deployment package* is a `.fed` file that contains all API Gateway configuration. This includes policies, listeners, external connections, users, certificates, and environment settings.
- A *policy package* is a `.pol` file that contains policies, listeners, external connections, and environment settings.
- An *environment package* is an `.env` file that contains users, certificates, and environment settings. The content of the `.fed` file is equivalent to the combined contents of the `.pol` and `.env` files.
- A *package property* is a name-value pair that applies to a specific configuration package (`.fed`, `.pol`, or `.env`). Specifying a property associates metadata with the configuration in that package. For example, the **Name** property with a value of `Default Factory Configuration` is associated with a default installation.

For more details on configuration packages and properties, see the *API Gateway DevOps Deployment Guide*.

Create a configuration package in Policy Studio

You can create an API Gateway configuration package for a currently loaded project configuration. To create a package (`.fed`, `.pol`, or `.env`), perform the following steps:

1. In the main menu, select **File > Save Package** followed by the appropriate option:
   - **Deployment Package** (`.fed`)
   - **Policy Package** (`.pol`)
   - **Environment Package** (`.env`)
2. Enter a file name, and click **Save**.

Configure package properties in Policy Studio

You can view or modify API Gateway configuration package properties for a currently loaded project configuration. To view and modify configuration properties, perform the following steps:

1. In the Policy Studio tree, and select **Environment Configuration > Package Properties > Policy** or **Environment**.
2. If you wish to create any additional properties (for example, **Department**), click the green (+) button on the right, and enter a property value (for example, `Engineering`).
3. If you wish to remove a property, click the red (x) button on the right of the property.
4. Click **Save** at the top right of the screen.

Deploy packages in Policy Studio

You can use the Policy Studio to create projects based on configuration packages. For more details, see *Create a Policy Studio project* on page 31.

You can deploy the configuration as normal using the **Deploy** button in the toolbar. For more details, see *Deploy configuration in Policy Studio* on page 210.

Deploy packages in API Gateway Manager

You can also use the API Gateway Manager web console to deploy configuration packages to a group of API Gateway instances. This functionality is available on the default **Dashboard** tab.

For more details, see the *API Gateway Administrator Guide*.

Deploy packages on the command line

You can create and deploy a deployment package (`.fed`) using the `managedomain --menu` command in the following directory:

```
INSTALL_DIR/apigateway/posix/bin
```

The deployment options for the `managedomain --menu` command are as follows:

```
18) Deploy to a group
19) List deployment information
20) Create deployment archive
21) Download deployment archive
22) Update deployment archive properties
```

For more details, see the *API Gateway Administrator Guide*.

Compare and merge API Gateway configurations

Overview

In the Policy Studio, you can compare the currently loaded API Gateway configuration with a configuration stored in a deployment package (`.fed` file). You can also merge any differences between the configurations.

Differences between configurations are identified as additions, deletions, or conflicts. When merging configurations, you can choose which differences to merge.

Note The currently loaded configuration can only be compared with a configuration stored in a deployment package (`.fed`) or in a server configuration file (`.xml`). You cannot compare against a policy package (`.pol`) or environment package (`.env`). For more information on configuration packages, see *Deploy API Gateway configuration* on page 210.

Compare and merge configurations

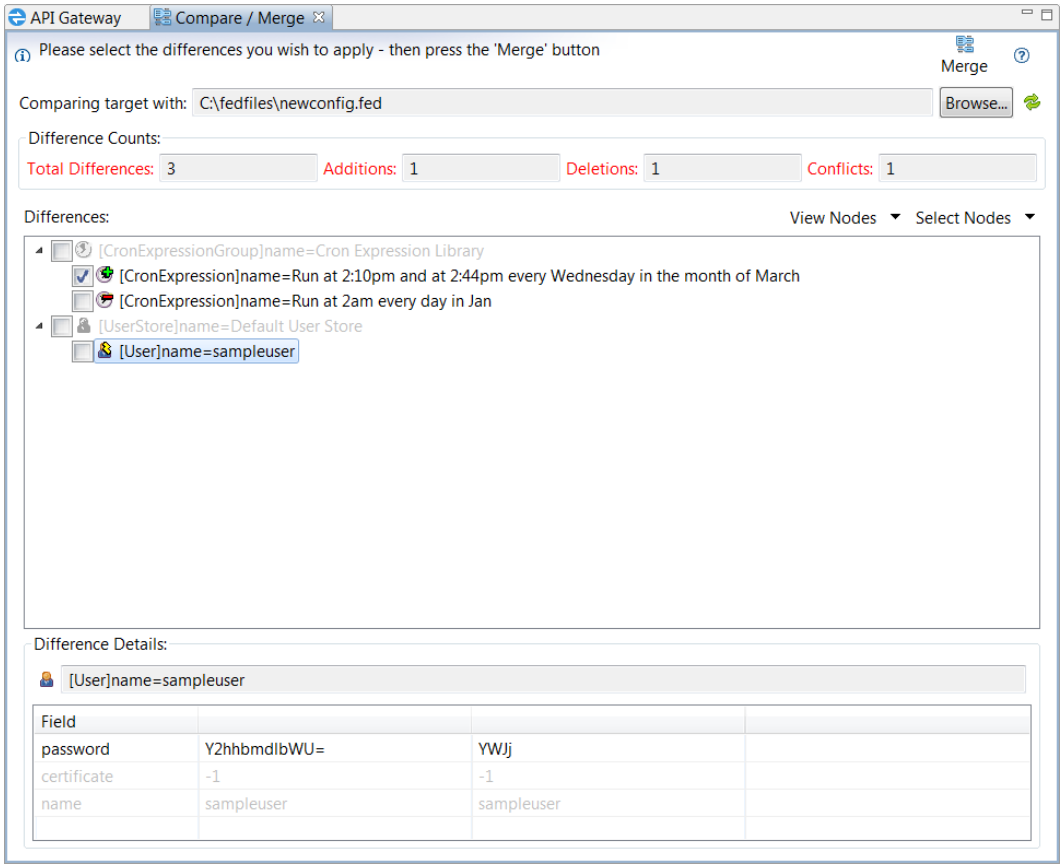
To compare the currently loaded configuration against the configuration in a `.fed` file, follow these steps:

1. Click the **Compare** button on the Policy Studio toolbar.
2. In the **Comparing target with** field, click the **Browse** button to choose a `.fed` file to compare the configuration with.
3. Enter the passphrase for the configuration, if one has been set, and click **OK** .The configurations are compared and the results are displayed in a tree view. Only entities with differences are shown.
4. To see detailed differences, click a configuration entity in the tree. The differences for that entity are displayed in the **Difference Details** pane.
5. To merge differences into the currently loaded configuration, select the check box next to each difference to be merged, and click the **Merge** button at the top right of the window.

Note If you modify the currently loaded configuration after the **Compare and Merge** tab is opened, click the **Refresh** button to refresh the comparison and show any new differences.

Comparison results

The following figure shows the result of a comparison:



The **Difference Counts** pane shows the number of differences in total, the number of additions, the number of deletions, and the number of conflicts.

The **Differences** tree view shows all of the differences in the configuration entities:

- Entities with green plus icon are additions. These entities exist in the `.fed` file but not in the currently loaded configuration.
- Entities with a red minus icon are deletions. These entities exist in the currently loaded configuration but not in the `.fed` file.
- Entities with a yellow warning icon are conflicts. These entities exist in both configurations but are not the same.

The **Difference Details** pane shows the values of the fields in each configuration when you click on an entity in the tree view. The second column shows the values of the fields in the `.fed` file, and the third column shows the values of the fields in the currently loaded configuration. The fields that are different in each configuration are highlighted.

In the preceding figure:

- The cron expression `Run at 2am every day in Jan` is a deletion.
- The cron expression `Run at 2:10pm and at 2:44pm every Wednesday in the month of March` is an addition.
- The user `sampleuser` is a conflict, because the password field has a different value in each configuration.

Some configuration entities contain references to other entities. In this case, an icon is displayed for the field in the **Difference Details** pane. Double-click a row with an icon to view the differences for those entities.

Filter differences

To filter nodes from the **Differences** tree view based on their type, click **View Nodes**, and select from the following options:

- **Additions**
- **Deletions**
- **Conflicts**

All differences are shown by default.

Select differences for merging

To select nodes for merging from the **Differences** tree view based on their type, click **View Nodes**, and select from the following options:

- **Additions**
- **Deletions**
- **Conflicts**

Additions are selected by default.

Manage admin users

When logging into the Policy Studio or API Gateway Manager, you must enter the user credentials stored in the local admin user store to connect to the API Gateway server instance. Admin users are responsible for managing API Gateway instances using the API Gateway management APIs. To manage admin users, click the **Settings** > **Admin Users** tab in the API Gateway Manager.

Note Admin users provide access to the API Gateway configuration management features available in the Policy Studio and API Gateway Manager. However, *API Gateway users* provide access to the messages and services protected by the API Gateway. For more details, see *Manage API Gateway users* on page 233.

Admin user privileges

After installation, a single admin user is defined in the API Gateway Manager with a user name of `admin`. Admin user rights in the system include the following:

- Add another admin user
- Delete another admin user
- Update an admin user
- Reset admin user passwords

Note An admin user *cannot* delete itself.

Remove the default admin user

If you need to remove the default admin user, perform the following steps:

1. Add another admin user.
2. Log in as the new admin user.
3. Delete the default admin user.

The **Admin Users** tab displays all existing admin users. You can use this tab to add, update, and delete admin users. These tasks are explained in the sections that follow.

Admin user roles

The API Gateway uses Role-Based Access Control (RBAC) to restrict access to authorized users based on their assigned roles in a domain. Using this model, permissions to perform specific system operations are assigned to specific roles only. This simplifies system administration because users do not need to be assigned permissions directly, but instead acquire them through their assigned roles.

For example, the default admin user (`admin`) has the following user roles:

- `Policy Developer`
- `API Server Administrator`
- `KPS Administrator`

API Gateway user roles and privileges

User roles have specific tools and privileges assigned to them. These define who can use which tools to perform what tasks. The user roles provided with the API Gateway assign the following privileges to admin users with these roles:

Role	Tool	Privileges
`API Server Administrator`	API Gateway Manager	Read/write access to API Gateway Manager.
`API Server Operator`	API Gateway Manager	Read-only access to API Gateway Manager.
`Deployer`	Deployment scripts	Deploy a new configuration.
`KPS Administrator`	API Gateway Manager	Perform create, read, update, delete (CRUD) operations on data in a Key Property Store (KPS).
`Policy Developer`	Policy Studio	Download, edit, deploy, version, and tag a configuration.

Note A single admin user typically has multiple roles. For example, in a development environment, a policy developer admin user would typically have the following roles:

- `Policy Developer`
- `API Server Administrator`

Add a new admin user

Complete the following steps to add a new admin user to the system:

1. Click the **Settings** > **Admin Users** tab in API Gateway Manager.
2. Click the **Create** button.
3. In the **Create New Admin User** dialog, enter a name for the user in the **Username** field.
4. Enter a user password in the **Password** field.
5. Re-enter the user password in the **Confirm Password** field.
6. Select roles for the user from the list of available roles (for example, `Policy Developer` and `API Server Administrator`).
7. Click **Create**.

Remove an admin user

To remove an admin user, select it in the **Username** list, and click **Delete**. The admin user is removed from the list and from the local admin user store.

Reset an admin user password

You can reset an admin user password as follows:

1. Select the admin user in the **Username** list.
2. Click the **Edit** button.
3. Enter and confirm the new password in the **Password** and **Confirm Password** fields.
4. Click **OK**.

Manage admin user roles

You can manage the roles that are assigned to specific admin users as follows:

1. Select the admin user in the **Username** list.
2. Click the **Edit** button.
3. Select the user roles to enable for this admin user in the dialog (for example, `Policy Developer` and/or `API Server Administrator`).
4. Click **OK**.

Edit API Gateway user roles

To add or delete specific API Gateway roles, you must edit the available roles in the `adminUsers.json` and `acl.json` files in the `conf` directory of your API Gateway installation.

For more details on role-based access, see the *API Gateway Administrator Guide*.

Configure a password policy for admin users

To configure the password policy that applies to admin user passwords, perform the following steps:

1. Click the **Settings** > **Admin Users** tab in API Gateway Manager.
2. Select **Password Policy enabled** to enable the password policy rules on this page. This is not selected by default.

3. Configure the following in **PASSWORD RULES**:
    - **Password must not be equal to the account name**: The password cannot be identical to the admin user name. This is selected by default.
    - **Password must not be the reverse of the account name**: The password cannot be the reverse of the admin user name. This is selected by default.
    - **Password must not contain the account name**: The password cannot contain the admin user name. This is selected by default.
    - **Minimum password length**: The password must be the specified minimum length. Defaults to 4 characters. If no value is specified, this rule is disabled.
    - **Password history length**: Enter the number of previous passwords to be compared. Leave this field empty to disable this rule.
    - **Minimum character differences from last password**: Enter the minimum number of different characters from the last password. Leave this field empty to disable this rule.
    - **Password lifetime (days)**: Enter how long the password is valid for in days. Leave this field empty to disable password expiry.
4. Configure the following in **PASSWORD COMPOSITION RULES**:
    - **Minimum uppercase characters**: Defaults to 1 uppercase alphabetic character.
    - **Minimum lowercase characters**: Defaults to 1 lowercase alphabetic character.
    - **Minimum numeric characters**: Defaults to 1 numeric character.
    - **Minimum special characters**: Defaults to 1 special character (`~!@#$%^&*()-_=+\[{}];:"",< >/?`).

      If no value is specified in these fields, these rules are disabled.
5. Click **Apply** when finished.

Environment configuration 7

This section describes how to configure your API Gateway environment. For example, this includes X.509 certificates and keys, users, caches, system alerts, and so on. For details on API Gateway listeners, see *Configure API Gateway instances* on page 264. See also *External connections* on page 359.

Manage X.509 certificates and keys

For API Gateway to trust X.509 certificates issued by a specific Certificate Authority (CA), you must import that CA's certificate into the API Gateway's trusted certificate store. For example, if API Gateway is to trust secure communications (SSL connections or XML Signature) from an external SAML Policy Decision Point (PDP), you must import the PDP certificate, or the issuing CA certificate into the API Gateway certificate store.

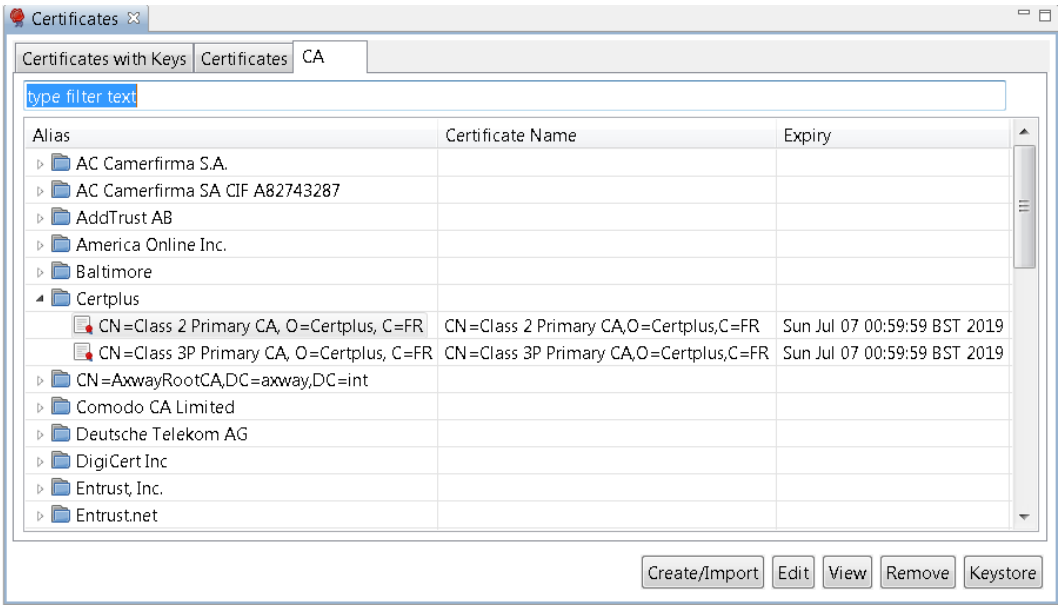
In addition to importing CA certificates, you can import and create server certificates and private keys in the certificate store. These can be stored locally or on an external Hardware Security Module (HSM). You can also import and create public-private key pairs. For example, these can be used with the Secure Shell (SSH) File Transfer Protocol (SFTP) or with Pretty Good Privacy (PGP).

View certificates and keys

To view the certificates and keys stored in the certificate store, select **Environment Configuration > Certificates and Keys > >Certificates** in the tree. Certificates and keys are listed on the following tabs in the **Certificates** window:

- **Certificates with Keys**: Server certificates with associated private keys
- **Certificates**: Server certificates without any associated private keys
- **CA**: Certificate Authority certificates with associated public keys

You can search for a specific certificate or key by entering a search string in the text box at the top of each tab, which automatically filters the tree.



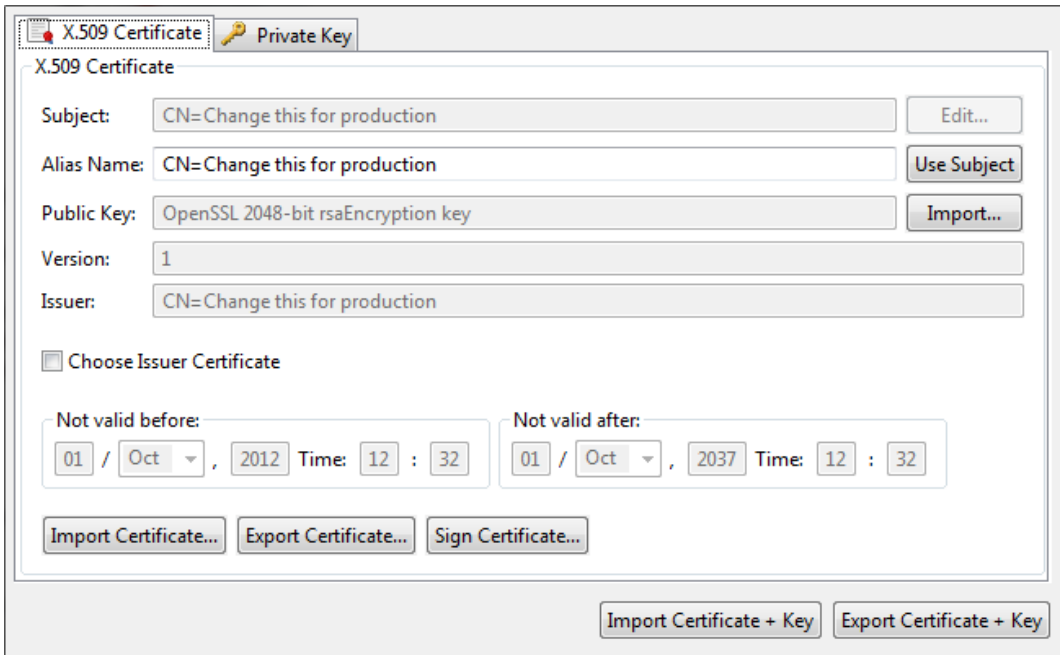
Certificate management options

The following options are available at the bottom right of the window:

- **Create/Import**: Click to create or import a new certificate and private key. For details, see *Configure an X.509 certificate* on page 222.
- **Edit**: Select a certificate, and click to edit its existing settings.
- **View**: Select a certificate, and click to view more detailed information.
- **Remove**: Select a certificate, and click to remove the certificate from the certificate store.
- **Keystore**: Click this to export or import certificates to or from a Java keystore. For details, see *Manage certificates in Java keystores* on page 233.

Configure an X.509 certificate

To create a certificate and private key, click **Create/Import**. The **Configure Certificate and Private Key** dialog is displayed. This section explains how to use the **X.509 Certificate** tab on this dialog.



Create a certificate

Configure the following settings to create a certificate:

- **Subject**:
  Click **Edit** to configure the *Distinguished Name* (DName) of the subject.
- **Alias Name**:
  This mandatory field enables you to specify a friendly name (or alias) for the certificate. Alternatively, you can click **Use Subject** to add the DName of the certificate in the text box instead of a certificate alias.
- **Public Key**:
  Click **Import** to import the subject's public key (usually from a PEM or DER-encoded file).
- **Version**:
  This read-only field displays the X.509 version of the certificate.
- **Issuer**:
  This read-only field displays the distinguished name of the CA that issued the certificate.
- **Choose Issuer Certificate**:
  Select to explicitly specify an issuer certificate for this certificate (for example, to avoid a potential clash or expiry issue with another certificate using the same intermediary certificate). You can then click the browse button on the right to select an issuer certificate. This setting is not selected by default.
- **Not valid before**:
  Select a date to define the start of the validity period of the certificate.
- **Not valid after**:
  Select a date to define the end of the validity period of the certificate.

- **Sign Certificate**:
  You must click this button to sign the certificate. The certificate can be self-signed, or signed by the private key belonging to a trusted CA whose key pair is stored in the certificate store.

Import certificates

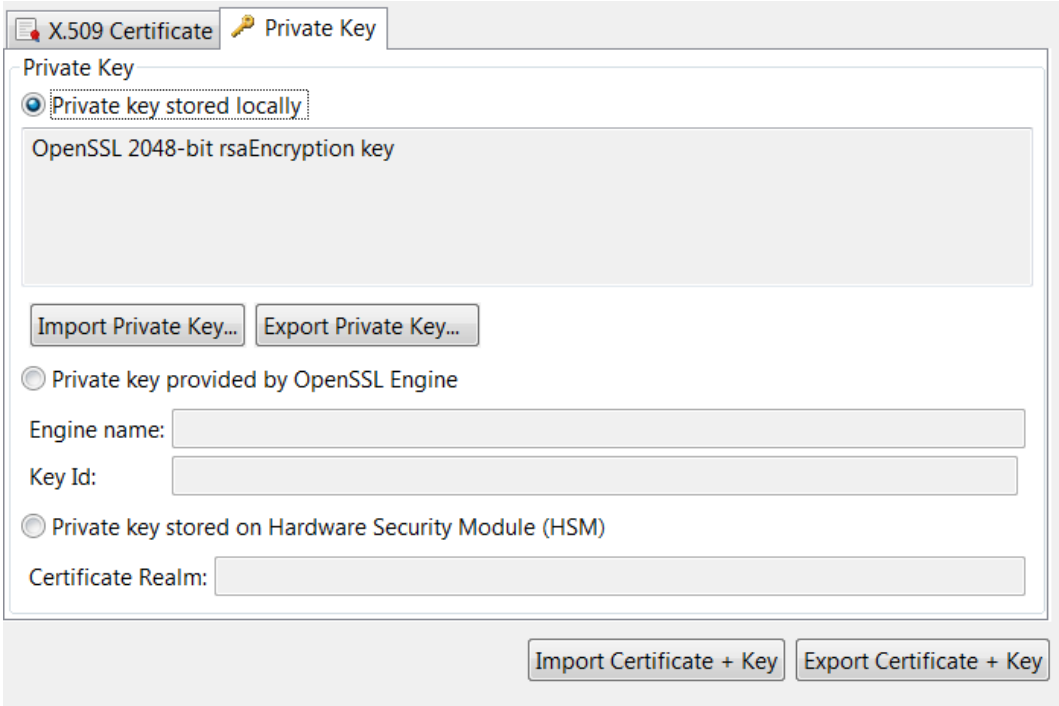
You can use the following buttons to import or export certificates into the certificate store:

- **Import Certificate**:
  Click to import a certificate (for example, from a `.pem` or `.der` file).
- **Export Certificate**:
  Click to export the certificate (for example, to a `.pem` or `.der` file).

Configure a private key

Use the **Private Key**tab to configure details of the private key. By default, private keys are stored locally (for example, in the API Gateway certificate store). They can also be provided by an OpenSSL engine, or stored on a Hardware Security Module (HSM) if required.

API Gateway supports PKCS#11-compatible HSM devices. For example, this includes Thales nShield Solo, SafeNet Luna SA, and so on.



Private key stored locally

If the private key is stored in the API Gateway certificate store, select **Private key stored locally**. The following options are available for keys stored locally:

- **Private key stored locally**:
  This read-only field displays details of the private key.
- **Import Private Key**:
  Click to import the subject's private key (usually from a PEM or DER-encoded file).
- **Export Private Key**:
  Click to export the subject's private key to a PEM or DER-encoded file.

Private key provided by OpenSSL engine

If the private key that corresponds to the public key in the certificate is provided by an OpenSSL engine, select **Private key provided by OpenSSL Engine**.

Configure the following fields to associate a key provided by the OpenSSL engine with the current certificate:

- **Engine name**:
  Enter the name of the OpenSSL engine to use to interface to an HSM. All vendor implementations of the OpenSSL Engine API are identified by a unique name. See your vendor's OpenSSL engine implementation or HSM documentation to find out the name of the engine.
- **Key Id**:
  Enter the key ID used to uniquely identify a specific private key from all others stored on an HSM. When you complete this dialog, the private key is associated with the certificate that you are currently editing. Private keys are identified by their key ID by default.

Private key stored on external HSM

If the private key that corresponds to the public key stored in the certificate resides on an external HSM, select **Private key stored on Hardware Security Module (HSM)**, and enter the name of the **Certificate Realm**.

Note To use the API Gateway's PKCS#11 engine to access objects in an external HSM, the corresponding HSM provider and certificate realms must also be configured. For more details, see *Configure HSMs and certificate realms* on page 225.

Configure HSMs and certificate realms

Certificate realms are abstractions of private keys and public key certificates, which mean that policy developers do not need to enter HSM-specific configuration such as slots and key labels. Instead, if

a private key exists on an HSM, the developer can configure the certificate to show that its private key uses a specific certificate realm, which is simply an alias for a private key (for example, `JMS Keys`).

For example, on the host machine, an administrator could configure the `JMS Keys` certificate realm, and create a *keystore* for the realm, which requires specific knowledge about the HSM (for example, PIN, slot, and private key label). The certificate realm is the alias name, while the keystore is the actual private keystore for the realm.

Manage HSMs with keystoreadmin

The `keystoreadmin` script enables you to perform the following tasks:

- Register an HSM provider
- List registered HSM providers
- Create a certificate realm
- List certificate realms

For example, if a policy developer is using JMS, and wants to indicate that private keys exist on an HSM, they could indicate that the certificate is using the `JMS Keys` certificate realm. On each instance using the configuration, it is the responsibility of the administrator to create the `JMS Keys` certificate realm.

For more details, enter `keystoreadmin` in the following directory, and perform the instructions at the command prompt:

```
INSTALL_DIR/apigateway/posix/bin
```

Use keystoreadmin in interactive mode

When you enter `keystoreadmin` without arguments, this displays an interactive menu with the following options:

Option	Description	When to use
1	`Change group or instance`	When registering HSMs or configuring certificate realms, you must choose the local group and instance to configure.
2	`List registered HSM providers`	Display the HSMs that are currently registered.

Option	Description	When to use
3	`Register an HSM provider`	Before creating certificate realms, you must first register the HSM. This option guides you through the steps. The HSM must be installed, configured, and active, and you must know the full path to the HSM device driver (PKCS#11). You give the HSM an alias (for example, LunaSA), which you use later when registering certificate realms.
4	`List Certificate Realms`	List configured certificate realms and associated keystores.
5	`Create a Certificate Realm`	Create a keystore and assign it to a certificate realm.

Step 1–Register an HSM provider

You must first register an HSM provider as follows:

1. Open a command prompt in the API Gateway `bin` directory (for example, `INSTALL_DIR/apigateway/posix/bin`).
2. Enter the `keystoreadmin` command.
3. Select option `3) Register an HSM provider`.
4. If prompted, select the appropriate API Gateway group or instance.
5. You are prompted for a provider alias name. The alias is local only. For example, if registering a LunaSA HSM, you might enter the `LunaSA` alias.
6. For convenience, `keystoreadmin` searches for supported HSM drivers. If found, it shows the list of supported drivers. If none are found, this does not mean the driver does not exist. You must see your HSM documentation for the location of the drivers. For example:

```
Choose from one of the following:1) /LunaSA/cryptoki.dll o)
Other q) Quit
```

7. If successful, `keystoreadmin` loads the driver and displays its details. For example:

```
Registering HSM provider...
Initializing HSM...
Crypto Version:2.20
Manufacter Id:SafeNet, Inc.
Library Description:Chrystoki
Library Version:5.1 Device registered.
```

Step 2–Create a certificate realm and associated keystore

To create a certificate realm and associated keystore, perform the following steps:

1. Open a command prompt in the API Gateway `bin` directory (for example, `INSTALL_DIR/apigateway/posix/bin`).
2. Enter the `keystoreadmin` command.
3. Select option `5) Create a Certificate Realm`.
4. You are prompted to enter a certificate realm name. This certificate realm name is used in when configuring the private key of the corresponding X.509 certificate. The realm name is case sensitive (for example, `JMS Keys`).
5. The registered HSMs are listed. For example, select option `1) HSM`.
6. The command connects to the selected HSM, and a list of available slots is displayed. Select the slot containing the private key to use for the certificate realm (for example, select slot `1`).
7. You are prompted to input the PIN passphrase for the slot. The passphrase will not echo any output.
8. When you enter the correct PIN passphrase for the slot, this displays a list of private keys. Choose the key to use for the certificate realm. For example:

```
Choose from one of the following:
  1) server1_priv
  2) jms_priv
  q) Quit

Select option:2
```

9. You are prompted for a file name for the keystore. For example:

```
Certificate realm filename [jms keys.ks]:Successfully created the
certificate realm:JMS KeysPress any key to continue...
```

10. The keystore is output to the API Gateway instance directory. For example:

```
apigateway/groups/group-2/instance-1/conf/certrealms/jms keys.ks
```

Note Each API Gateway instance must have its certificate realm configured. When finished creating certificate realms, you must restart the API Gateway instance for the changes to take effect.

Step 3–Start API Gateway when using an HSM

When API Gateway is configured to use certificate realms, these realms are initialized on startup, and a connection to the corresponding HSM is established. This requires the PIN passphrase for the specific HSM slots. At startup, you can manually enter the required HSM slot PIN passphrase, or you can automate this instead.

Start API Gateway with manually entered PIN passphrase

When API Gateway is configured to use an HSM, API Gateway stops all processing, prompts for the HSM slot PIN passphrase, and waits indefinitely for input. For example:

```
INFO    07/Jan/2015:16:31:54 Initializing certificate realm 'JMS Keys'...
Enter passphrase for Certificate Realm, "JMS Keys":
```

API Gateway does not reprompt if the PIN passphrase is incorrect. It logs the error and continues, while any services that use the certificate realm cannot use the HSM.

Start API Gateway with automatic PIN passphrase

You can configure API Gateway to start and initialize the HSM by invoking a command script on the operating system to obtain the HSM slot PIN passphrase. This enables API Gateway for automatic startup without manually entering the PIN passphrase.

To configure an automatic PIN passphrase, perform the following steps:

1. Edit the API Gateway instance's `vpkcs11.xml` configuration file. For example:

   ```
   apigateway/groups/group-2/instance-1/conf/vpkcs11.xml
   ```

2. Add a `PASSPHRASE_EXEC` command that contains the full path to the script that executes and obtains the passphrase. The script should write the passphrase to stdout, and should have the necessary operating system file and execute protection settings to prevent unauthorized access to the PIN passphrase. The following example shows a `vpkcs11.xml` file that invokes the `hsmpin.sh` to echo the passphrase:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <ConfigurationFragment provider="cryptov">

       <Engine name="vpkcs11" defaultFor="">
           <EngineCommand when="preInit" name="REALMS_DIR"
                value="$VINSTDIR/conf/certrealms" />
           <EngineCommand when="preInit" name="PASSPHRASE_EXEC"
                value=""$VDISTDIR/hsmpin.sh"" />
       </Engine>
   </ConfigurationFragment>
   ```

3. API Gateway provides the certificate realm as an argument to the script, so you can use the same script to initialize multiple realms. The following examples show scripts that write a PIN of `1234` to stdout when initializing the `JMS Keys` certificate realm:

```
#!/bin/shcase $1 in"JMS Keys")echo 1234;;esac
```

Configure SSH key pairs

To configure public-private key pairs in the certificate store, select **Environment Configuration > Certificates and Keys > Key Pairs**. The **Key Pairs** window enables you to add, edit, or delete OpenSSH public-private key pairs, which are required for the Secure Shell (SSH) File Transfer Protocol (SFTP).

Add a key pair

To add a public-private key pair, click **Add** on the right, and configure the following settings in the dialog:

- **Alias**:
  Enter a unique name for the key pair.
- **Algorithm**:
  Enter the algorithm used to generate the key pair. Defaults to `RSA`.
- **Load**:
  Click to select the public key or private key files to use. The **Fingerprint** field is auto-populated when you load a public key.

Note The keys must be OpenSSH keys. RSA keys are supported, but DSA keys are not supported. The keys must not be passphrase protected.

Edit a key pair

To edit a public-private key pair, select a key pair alias in the table, and click **Edit** on the right. For example, you can load a different public key and private key. Alternatively, double-click a key pair alias in the table to edit it.

You can delete a selected key pair from the certificate store by clicking **Remove** on the right. Alternatively, click **Remove All**.

Manage OpenSSH keys

You can use the `ssh-keygen` command provided on Linux to manage OpenSSH keys. For example:

- The following command creates an OpenSSH key:

```
ssh-keygen -t rsa
```

- The following command converts an `ssh.com` key to an OpenSSH key:

```
ssh-keygen -i -f ssh.com.key > open.ssh.key
```

- The following command removes a passphrase (enter the old passphrase, and enter nothing for the new passphrase):

```
ssh-keygen -p
```

- The following command outputs the key fingerprint:

```
ssh-keygen -lf ssh_host_rsa_key.pub
```

Configure PGP key pairs

To configure Pretty Good Privacy (PGP) key pairs in the certificate store, select **Environment Configuration > Certificates and Keys > PGP Key Pairs**. The **PGP Key Pairs** window enables you to add, edit, or delete PGP public-private key pairs.

Add a PGP key pair

To add a PGP public-private key pair, click the **Add** on the right, and configure the following settings in the dialog:

- **Alias**:
  Enter a unique name for the PGP key pair.
- **Load**:
  Click **Load** to select the public key and private key files to use.

Note The PGP keys added must not be passphrase protected.

Edit a PGP key pair

To edit a PGP key pair, select a key pair alias in the table, and click **Edit** on the right. For example, you can load a different public key and private key. Alternatively, double-click a key pair alias in the table to edit it.

You can delete a selected PGP key pair from the certificate store by clicking **Remove** on the right. Alternatively, click **Remove All**.

Manage PGP keys

You can use the freely available GNU Privacy Guard (GnuPG) tool to manage PGP key files (available from http://www.gnupg.org/). For example:

- The following command creates a PGP key:

  ```
  gpg --gen-key
  ```

  For more details, see http://www.seas.upenn.edu/cets/answers/pgp_keys.html
- The following command enables you to view the PGP key:

  ```
  gpg -a --export
  ```
- The following command exports a public key to a file:

  ```
  gpg --export -u 'UserName ' -a -o public.key
  ```
- The following command exports a private key to a file:

  ```
  gpg --export-secret-keys -u 'UserName ' -a -o
  private.key
  ```
- The following command lists the private keys:

  ```
  gpg --list-secret-keys
  ```

Global import and export options

This section describes global import and export options available when managing certificates and keys.

Import and export certificates and keys

The following global configuration options apply to both the **X.509 Certificate** and **Private Key** tabs:

- **Import Certificate + Key**:
  Use this option to import a certificate and a key (for example, from a `.p12` file).
- **Export Certificate + Key**:
  Use this option to export a certificate and a key (for example, to a `.p12` file).

Click **OK** when you have finished configuring the certificate and private key.

Manage certificates in Java keystores

You can also export a certificate to a Java keystore. You can do this by clicking **Keystore** on the main **Certificates** window. Click the browse button at beside the **Keystore** field at the top right to open an existing keystore, or click **New Keystore** to create a new keystore. Choose the name and location of the keystore file, and enter a passphrase for this keystore when prompted. Click **Export to Keystore**, and select a certificate to export.

Similarly, you can import certificates and keys from a Java keystore into the certificate store. To do this, click **Keystore** on the main **Certificates** window. On the **Keystore** window, browse to the location of the keystore by clicking the browse button beside the **Keystore** field. The certificates/keys in the keystore are listed in the table.

To import any of these keys to the certificate store, select the box next to the certificate or key to import, and click **Import to Trusted certificate store**. If the key is protected by a password, you are prompted for this password.

You can also use the **Keystore** window to view and remove existing entries in the keystore. You can also add keys to the keystore and to create a new keystore. Use the appropriate button to perform any of these tasks.

Further information

For more details on supported security features, see the *API Management Security Guide*.

Manage API Gateway users

By default, the API Gateway user store contains the configuration data for managing API Gateway user information. The API Gateway user store is typically used in a development environment, and is useful for demonstration purposes.

In a production environment, user information may be stored in existing user Identity Management repositories such as Microsoft Active Directory, Oracle Access Manager, CA SiteMinder, and so on. For more details, see the relevant *API Gateway Integration Guide*.

Note API Gateway users provide access to the messages and services protected by API Gateway. However, *admin users* provide access to the API Gateway configuration management features available in Policy Studio, Configuration Studio, and API Gateway Manager.

API Gateway users

API Gateway users specify the user identity in the API Gateway user store. This includes details such as the user name, password, and X.509 certificate. API Gateway users must be a member of at least one user group. In addition, users can specify optional attributes, and inherit attributes at the group level.

To view all existing users, select the **Environment Configuration > Users and Groups > Users** node in the tree. The users are listed in the table on the main panel. You can find a specific user by entering a search string in the **Filter** field.

Add API Gateway users

You can create API Gateway users on the **Users** page. Click the **Add** button on the right.

To specify the new user details, complete the following fields on the **General** tab:

- **User Name**:
  Enter a name for the new user.
- **Password**:
  Enter a password for the new user.
- **Confirm Password**:
  Re-enter the user's password to confirm.
- **Signing Key**:
  Click to load the user certificate from the **Certificate Store**. For details on how to create and import certificates, see *Manage X.509 certificates and keys* on page 221.

You can also specify optional user attributes on the **Attributes** tab, which is explained in the next section.

API Gateway user attributes

You can specify attributes at the user level and at the group level on the **Attributes** tab. Attributes specify user configuration data (for example, attributes used to generate SAML attribute assertions).

The **Attributes** tab enables you to configure user attributes as simple name-value pairs. The following are examples of user attributes:

- `role=admin`
- `email=steve@axway.com`
- `dept=eng`
- `company=axway`

You can add user attributes by clicking the **Add** button. Enter the attribute name, type, and value in the fields provided. The `Encrypted` type refers to a string value that is encrypted using a well-known encryption algorithm or cipher.

API Gateway user groups

API Gateway user groups are containers that encapsulate one or more users. You can specify attributes at the group level, which are inherited by all group members. If a user is a member of more than one group, that user inherits attributes from all groups (the superset of attributes across

the groups of which the user is a member).

To view all existing groups, select the **Environment Configuration > Users and Groups > Groups** node in the tree. The user groups are listed in the table on the main panel. You can find a specific group by entering a search string the **Filter** field.

Add API Gateway user groups

You can create user groups on the **Groups** page. Click the **Add** button on the right to view the **Add Group** dialog.

To specify the new group details, complete the following fields on the **General** tab:

- **Group Name**:
  Enter a name for the new group.
- **Members**:
  Click the **Add** button to display the **Add Group Member** dialog, and select the members to add to the group.

You can also specify optional attributes at the group level on the **Attributes** tab. For more details, see *API Gateway user attributes* on page 234.

Update API Gateway users or groups

To edit details for a specific user or group, select it in the list, and click the **Edit** button on the right. Enter the updated details in the **Edit User** or **Edit Group** dialog.

To delete a specific user or group, select it in the list, and click the **Remove** button on the right. Alternatively, to delete all users or Groups, click the **Remove All** button. You are prompted to confirm all deletions.

Configure system alerts

This topic describes how to configure API Gateway system alerts. System alerts and events are usually sent when a filter fails, but you can also use the alerts for notification purposes. API Gateway can send system alerts to several alert destinations, including a Windows Event Log, UNIX/Linux syslog, SNMP Network Management System, Check Point Firewall-1, email recipient, Amazon Simple Notification Service (SNS), or Twitter.

The main steps when configuring API Gateway to send system alerts are:

1. *Configure an alert destination* on page 236
2. *Configure an alert policy* on page 241

Configure an alert destination

The first step in configuring API Gateway to send alerts is to configure an *alert destination*. This section describes the destinations to which the API Gateway can send alerts to. To configure the alert destinations, go to **Environment Configuration > Libraries > Alerts** node in the Policy Studio tree.

Syslog (local or remote)

Linux operating systems often provide a general purpose logging utility called `syslog`. Both local and remote processes send logging messages to a centralized system logging daemon (`syslog`) that in turn writes the messages to the appropriate log files.

You can configure the level of detail at which `syslog` logs information. This way administrators can centrally manage how log files are handled, instead of separately configuring logging for each process.

Each type of process logs to a different syslog *facility*. There are facilities for the kernel, user processes, authorization processes, daemons, and a number of placeholders used by site-specific processes. API Gateway can log to several facilities such as `auth`, `daemon`, `ftp`, `local0-7`, and `syslog` itself.

Remote syslog

You can configure a remote `syslog` alert destination by specifying the details of the machine on which the `syslog` daemon is running. When an alert event is triggered, API Gateway connects to this daemon and logs to the specified facility.

1. In the **Alerts** node, click **Add > Syslog Remote**.
2. Enter a name for your alert destination.
3. Enter the host name or IP address of the machine where the `syslog` daemon is running.
4. Select the facility that API Gateway sends alerts to, and click **OK**.

Local syslog (UNIX only)

To configure a local `syslog` alert destination, perform the following steps:

1. In the **Alerts** node, click **Add > Syslog Local (UNIX only)**.
2. Enter a name for your alert destination.
3. Select the facility that API Gateway sends alerts to, and click **OK**.

Windows Event Log

You configure the alert messages to be written to a local or remote Windows Event Log.

1. In the **Alerts** node, click **Add > Windows Event Log**.
2. Enter a name for your alert destination.
3. In **UNC Server name**, enter the Universal Naming Code (UNC) of the host machine where the event log is located. For example, to send alerts to the event log running on a machine called `\\NT_SERVER`, enter `\\NT_SERVER`.
4. Click **OK**.

Check Point FireWall-1 (OPSEC)

API Gateway complies with Open Platform for Security (OPSEC). OPSEC compliance is awarded by Check Point Software Technologies to products that have been successfully integrated with at least one of their products, in this case Check Point FireWall-1,

FireWall-1 is the industry leading firewall that provides network security based on a security policy created by an administrator. Although OPSEC is not an open standard, the platform is recognized worldwide as the standard for interoperability of network security, and the alliance contains over 300 different companies. OPSEC integration is achieved through a number of published APIs that enable third-party vendors to interoperate with Check Point products.

You can specify where FireWall-1 is installed, the port it is listening on, and how to authenticate to the firewall. When an alert event is triggered, API Gateway connects to the specified firewall and prevents further requests for the particular client that triggered the alert.

1. In the **Alerts** node, click **Add > OPSEC**.
2. Enter a name for your alert destination.
3. If you have an existing OPSEC configuration file, click **Browse** and load the file. If you do not have an exiting file, click **Template** to generate the required settings into the text area.
4. Ensure that the following settings are set correctly for your firewall:
    - **sam_server auth_port**: The port number used to establish Secure Internal Communications (SIC) connections with the firewall
    - **sam_server auth_type**: The authentication method used to connect to the firewall
    - **sam_server ip**: The host name or IP address of the machine that hosts Check Point Firewall
    - **sam_server opsec_entity_sic_name**: The firewall's SIC name

- **opsec_sic_name**: The OPSEC application SIC Name (application's full DName defined by the VPN-1 SmartCenter Server)
- **opsec_sslca_file**: The name of the file containing the OPSEC application's digital certificate

5. For API Gateway to establish the SSL connection to the firewall, the specified `opsec_sslca_file` must be uploaded to API Gateway. To do this, click **Add** and select the correct file.
6. Click **OK**.

For more information on OPSEC settings, see the documentation for your OPSEC application.

SNMP Network Management System

API Gateway can send Simple Network Management Protocol (SNMP) traps to a Network Management System (NMS).

1. In the **Alerts** node, click **Add > SNMP**.
2. Enter a name for your alert destination, and configure the following:
   - **Host**: The host name or IP address of the machine where the NMS is located
   - **Port**: The port where the NMS is listening
   - **Timeout**: The timeout (in seconds) for connections from API Gateway to the NMS
   - **Retries**: The number of retry attempts if a connection failure occurs
   - **SNMP Version**: The version of SNMP used
3. Click **OK**.

Email recipient

You can configure API Gateway to send alert messages as emails.

1. In the **Alerts** node, click **Add > Email**.
2. Enter a name for your alert destination, and the recipients of the alert mail. If you want to include multiple recipients, use semicolon to separate the email addresses.
3. In **Email Sender (From)**, enter the email address where you want the email appears to be *from*.

   Note Some mail servers do not allow relaying mail when the sender in the **From** field is not recognized by the server.

4. In **Email Subject**, enter the subject that the email alerts will use.
5. In the **SMTP Server Settings**, set the following:
   - **Outgoing Mail Server (SMTP)**: The SMTP server that API Gateway uses to relay the alert email
   - **Port**: The SMTP server port (default port is `25`)

- **Connection Security**: The connection security used to send the alert email: `SSL`, `TLS`, or `NONE` (default)

6. If the SMTP server requires authentication, specify the credentials in **Log on Using**.
7. To set API Gateway to log information on any errors encountered when attempting to send email alerts, select the **Email Debugging**. All trace files are written to the `/trace` directory of your API Gateway installation. This setting is diasbled by default.
8. Click **OK**.

Amazon SNS

You can configure API Gateway to send alert messages to the Amazon Simple Notification Service (SNS).

Amazon SNS is a managed push messaging service that you can used to send push notifications to mobile and smart devices connected to the Internet, as well as to other distributed services. For more details on Amazon SNS, go to http://aws.amazon.com/sns/.

1. In the **Alerts** node, click **Add > Amazon SNS**.
2. Enter a name for this alert destination.
3. In **AWS Credential**, select your AWS security credentials (API key and secret) that API Gateway uses when connecting to Amazon SNS.
4. Select the region appropriate for your deployment.
5. In **Client settings**, select the AWS client configuration API Gateway uses when connecting to Amazon SNS. For more details, see *Configure Amazon SQS queue listener* on page 351.
6. In **Topic ARN**, enter the topic Amazon Resource Name (ARN) to send alerts to.

   When you create a topic, Amazon SNS assigns it a unique ARN that includes the service name (for example, SNS), the region, the AWS ID of the user, and the topic name. The ARN is returned as part of the API call to create the topic.

   For example, `arn:aws:sns:us-east-1:1234567890123456:mytopic` is the ARN for a topic named `mytopic` created by a user with the AWS account ID `123456789012` and hosted in the US East region.

   Whenever a publisher or subscriber needs to perform any action on the topic, they reference the unique topic ARN.

7. Enter the subject of the alerts will use, and click **OK**.

Twitter

If you have a Twitter account, you can configure API Gateway to send tweet alerts to Twitter. API Gateway acts as a client application to make API calls and post alerts on behalf of the user.

Twitter uses the OAuth open authentication standard, and requires that API calls are made for both the user and the client application. Twitter API requires the following credentials to determine which application is calling the API and to verify that the Twitter user in question has authorized access to their account using the specified application:

- The consumer key of the client application
- The consumer secret key of the client application
- The access token that allows the client application to post on behalf of the user
- The access token secret to verify the access token

Twitter identifies and authenticates all requests as coming from both the user performing the request and the registered API Gateway application working on the user's behalf.

Register a client application

To use the Twitter API, you must first have a Twitter account you can use. Then, register a client application for API Gateway:

1. Go to http://dev.twitter.com.
2. On the Twitter toolbar, select **Your apps**.
3. Click **Register a new app**.
4. Enter the details for the API Gateway client application. Some details are arbitrary, but you must specify the following:
    - **Application Type**: `Client`
    - **Default Access Type**: `Read & Write`

        Note The application name may already be registered to another user, so you may need to specify a different unique name.
5. Click **Register Application**. Each client application you register is provisioned a consumer key and consumer secret. These are used, in conjunction with the OAuth library, to sign every request you make to the Twitter API. Using this signing process, Twitter trusts that the traffic identifying itself as you is indeed you.
6. Select your registered application, and select **My Access Token** to view the access token and an access token secret. You must store these safely.

Configure a Twitter alert destination

To configure a Twitter alert destination, perform the following steps:

1. In the **Alerts** node, click **Add > Twitter**.
2. Specify the credentials for the Twitter user that API Gateway sends an alert to:
    - **Consumer Key**: The consumer key of the registered client application
    - **Consumer Secret**: The consumer secret of the registered client application
    - **Access Token**: The access token that represents you

- **Access Token Secret**: The access token secret that represents you

Configure an alert policy

To send alert notifications to alert destinations, configure an alert policy. For example, you can configure an Alert filter to send the notifications, or use a Scripting Language filter for more fine-grain notifications. For more details, see *API Gateway Policy Developer Filter Reference*.

Policy execution scheduling

Overview

You can configure a policy execution scheduler at the level of the API Gateway instance. This enables you to schedule the execution of any policy on a specified date and time in a recurring manner. The API Gateway provides a pre-configured library of schedules to select from when creating a policy execution scheduler. You can also add your own schedules to the globally available library in the Policy Studio.

You can use policy execution scheduling in any policy (for example, to perform a message health check). This feature is also useful when polling a service to enforce a Service Level Agreement (for example, to ensure the response time is less than 1000 ms, and if not, to send an alert).

Cron expressions

In the Policy Studio, policy execution schedules are based on cron expressions. A cron expression is a string that specifies a time schedule for triggering an event (for example, executing a policy). It consists of six required fields and one optional field, each separated by a space, which together specify when to trigger the event. For example, the following expression specifies to run at 10:15 am every Monday, Tuesday, Wednesday, Thursday, and Friday in 2011:

```
0 15 10 ? * MON-FRI 2011
```

Syntax

The following table shows the syntax used for each field:

Field	Values	Special characters
Seconds	0-59	, - * /

Field	Values	Special characters
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day of Month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day of Week	1-7 or SUN-SAT	, - * ? / L #
Year (optional)	empty or 1970-2199	, - * /

Special characters

The special characters are explained as follows:

Special character	Description
,	Separates values in a list (for example, MON,WED,SAT means Mondays, Wednesdays, and Saturdays only).
-	Specifies a range of values (for example, 2011-2015 means every year between 2011 and 2015 inclusive).
*	Specifies all values of the field (for example, every minute).
?	Specifies no value in the Day of Month and Day of Week fields. This enables you to specify a value in one field, but not in the other.
/	Specifies time increments (for example, in the Minutes field, 0/15 means minutes 0, 15, 30, and 45, while 5/15 means minutes 5, 20, 35, and 50). Specifying * before the / is the same as specifying 0 as the start value. The / character enables you to turn on every nth value in the set of values for the specified field. For example, 7/6 in the month field only turns on month 7, and does not mean every 6th month.
L	Specifies the last value in the Day of Month and Day of Week fields. In the Day of Month field, this means the last day of the month (for example, January 31, or February 28 in non-leap years). In the Day of Week field, when used alone, this means 7 or SAT. When used after another value, it means the last XXX day of the month (for example, 5L means the last Thursday of the month). When using the L character, do not specify lists or ranges because this can give confusing results.

Special character	Description
`W`	Specifies the weekday (Monday-Friday) nearest the given day. For example, `15W` means the nearest weekday to the 15th of the month. If the 15th is a Saturday, the trigger fires on Friday 14th. If the 15th is a Sunday, it fires on Monday 16th. If the 15th is a Tuesday, it fires on Tuesday 15th. However, if you specify `1W`, and the 1st is a Saturday, the trigger fires on Monday 3rd to avoid crossing the month boundary. You can only specify the `W` character for a single day, and not a range or list of days.
`#`	Specifies the nth `XXX` weekday of the month in the Day of Week field. For example, a value of `FRI#2` means the second Friday of the month. However, if you specify `#5`, and there are not 5 of the specified Day of the Week in the month, no policy is run that month. When the `#` character is specified, there can only be one expression in the Day of Week field (for example, `2#1,6#4` is not valid because there are two expressions).

Examples

The following are some of the cron expressions provided in the **Schedule Library** in the Policy Studio:

Cron expression	Description
`0 15 10 ? * *`	Run at 10:15am every day.
`0 15 10 ? * 6L 2011-2015`	Run at 10:15am on every last Friday of every month during the years 2011, 2012, 2013, 2014, and 2015.
`0 15 10 ? * 6#3`	Run at 10:15am on the third Friday of every month.
`0 0 10 1,15 * ?`	Run at 10am on the 1st and 15th days of the month.
`0 10,44 14 ? 3 WED`	Run at 2:10pm and at 2:44pm every Wednesday in the month of March.
`0,30 * * ? * SAT,SUN`	Run every 30 seconds but only on Weekends (Saturday and Sunday).

Cron expression	Description
`0 0/5 14,18 * * ?`	Run every 5 minutes starting at 2pm and ending at 2:55pm, and every 5 minutes starting at 6pm and ending at 6:55pm, every day.
`0 0-5 14 * * ?`	Run every minute starting at 2pm and ending at 2:05pm, every day.

Note Note the following:

- Support for specifying both a Day of Week and a Day of Month value is not complete. You must use the `?` or `*` character in one of these fields.
- Overflowing ranges with a larger number on the left than the right are supported (for example, `21-2` for 9pm until 2am, or `OCT-MAR`). However, overuse may cause problems with daylight savings (for example, `0 0 14-6 ? * FRI-MON`).

Add a schedule

To add a schedule to the globally available library in the Policy Studio, perform the following steps:

1. Select the **Environment Configuration** > **Libraries** > **Schedules** node in the tree.
2. Click the **Add** button at the bottom of the **Schedules** window.
3. In the **Schedules** dialog, enter a **Name** (for example, `Run every 30 seconds`).
4. Enter a **Cron expression** (for example, `0/30 * * * * ?`). Alternatively, click the browse button to select an expression in **Cron dialog**. For more details, see *Configure cron expressions* on page 434.
5. Click **OK**.

You can also edit or delete a selected schedule using the appropriate button.

Add a policy execution scheduler

To add a policy execution scheduler in the Policy Studio, perform the following steps:

1. Select the **Environment Configuration** > **Listeners** node on the left.
2. Right-click the instance node (for example, **API Gateway**), and select **Add policy execution scheduler**.
3. Click the button next to the **Schedule** field, select a cron expression in the dialog, and click **OK**.
4. Click the button next to the **Policy** field, select a policy in the dialog, and click **OK**. You can search for a specific policy by entering its name in the text box, and the policy tree is filtered

automatically.

5. Click **OK**.

Global caches

Overview

In cases where a back-end service is serving the same request (and generating the same response) over and over again, it makes sense to use a caching mechanism. When a cache is employed, a unique identifier for the request is cached together with the corresponding response for this request. If an identical request is received, the response can be retrieved from the cache instead of forcing the service to reprocess the identical request and generate the same response. The use of caching in this way helps divert unnecessary traffic from the service and makes it more responsive to new requests.

For example, assume you have deployed a service that returns a list of cities in the USA from an external database, which is then used by a variety of web-based applications. Because the names and quantity of cities in the USA are relatively constant, if the service handles hundreds or thousands of requests every day, this is waste of processing time and effort, especially considering that the database that contains the relatively fixed list of city names is hosted on a separate machine to the service.

If you assume that the list of cities in the database does not change very often, it makes sense to use the API Gateway to cache the response from the service that contains the list of cities. Then when a request for this service is identified by the API Gateway, the cached response can be returned to the client. This approach results in the following performance improvements:

- The API Gateway does not have to route the message on to the service, therefore saving the processing effort required, and perhaps more importantly, saving the time it takes for the round trip.
- The service does not have to waste processing power on generating the same list over and over again, therefore making it more responsive to requests for other services.
- Assuming a naive implementation of database retrieval and caching, the service does not have to query the database (over the network) and collate the results over and over again for every request.

The caching mechanism used in the API Gateway offers full control over the size of the cache, the lifetime of objects in the cache, whether objects are cached to disk, and even whether caches can be replicated across multiple API Gateway instances. This topic describes how to configure both local and distributed caches in the API Gateway, and shows examples of how to configure a policy to cache responses.

Local caches

Local caches are used where a single API Gateway instance has been deployed. In such cases, you do not need to replicate caches across multiple running instances of the API Gateway.

Add a local cache

In the Policy Studio tree, you can add a local cache by selecting the **Environment Configuration** > **Libraries** > **Caches** node, and clicking the **Add** button at the bottom right of the window. Select **Add Local Cache** from the menu. You can configure the following fields on the **Configure Local Cache** dialog:

Cache Name:
Enter a name for the cache.

Maximum Elements in Memory:
Enter the maximum number of objects that can be in memory at any one time.

Maximum Elements on Disk:
Sets the maximum number of objects that can be stored in the disk store at any one time. A value of zero indicates an unlimited number of objects.

Eternal:
If this option is selected, objects stored in the caches never expire and timeouts have no effect.

Overflow to Disk:
Select this option if you want the cache to overflow to disk when the number of objects in memory has reached the amount set in the **Maximum Elements in Memory** field above.

Time to Idle:
Determines the maximum amount of time (in seconds) between accesses that an object can remain idle before it expires. A value of zero indicates that objects can idle for infinity, which is the default value. If the **Eternal** field is selected, this setting is ignored.

Time to Live:
Sets the maximum time between when an object is created and when it expires. The default value is zero, which means that the object can live for infinity. If the **Eternal** field is selected, this setting is ignored.

Persist to Disk:
If selected, the disk store is persisted between JVM restarts. This option is disabled by default.

Disk Expiry Interval:
Configures the number of seconds between runs of the disk expiry thread. The default is 120 seconds.

Disk Spool Buffer Size:
Indicates the size of memory (in MBs) to allocate the disk store for a spool buffer. Writes are made to this memory and then asynchronously written to disk. The default size is 30 MB. If you get `OutOfMemory` exceptions, you can consider lowering this value. However, if you notice poor performance, you should increase the value.

Eviction Policy:
Select the eviction policy that the cache uses to evict objects from the cache. The default policy is **Least Recently Used**. However, you can also use **First in First Out** and **Less Frequently Used**.

Distributed caches

If you have deployed several API Gateways throughout your network, you might need to employ a distributed cache. In this scenario, each API Gateway has its own local copy of the cache but registers a cache event listener that replicates messages to the other caches so that put, remove, expiry, and delete events on a single cache are duplicated across all other caches.

Add a distributed cache

You can add a distributed cache by selecting the **Environment Configuration** > **Libraries** > **Caches** tree node, and clicking the **Add** button at the bottom right of the window. Select **Add Distributed Cache** from the menu, and configure the following fields on the **Configure Distributed Cache** dialog:

Note Many of the settings for the distributed cache are identical to those for the local cache. For details on how to configure these fields, see *Add a local cache* on page 246. The following information refers to fields that are not displayed on both dialogs.

Event Listener Class Name:
Enter the name of the listener factory class that enables this cache to register listeners for cache events, such as put, remove, delete, and expire.

Properties Separator:
Specify the character to use to separate the list of properties.

Properties:
Specify the properties to pass to the `RMICacheReplicatorFactory`. The following properties are available:

- `replicatePuts=true | false`
  Determines whether new elements placed in a cache are replicated to other caches. Default is `true`.
- `replicateUpdates=true | false`
  Determines whether new elements that override (update) existing elements with the same key in a cache are replicated. Default is `true`.
- `replicateRemovals=true`
  Determines whether element removals are replicated. Default is `true`.
- `replicateAsynchronously=true | false`
  Determines whether replications are asynchronous (true) or synchronous (false). Default is `false`.

- `replicateUpdatesViaCopy=true | false`
  Determines whether new elements are copied to other caches (true)or a remove message is sent (false). Default is `true`.
- `asynchronousReplicationIntervalMillis=[number of ms]`
  The asynchronous replicator runs at a set interval of milliseconds. The default is 1000 and the minimum is 10. This property is only applicable if `replicateAsynchronously=true`.

Cache Bootstrap Class Name:
Specifies a `BootstrapCacheLoader` factory that the cache can call on initialization to prepopulate itself. The `RMIBootstrapCacheLoader` bootstraps caches in clusters where `RMICacheReplicator` s are used.

Properties Separator:
The character entered here is used to separate the list of properties listed in the field below.

Properties:
The properties listed here are used to initialize the `RMIBootstrapCacheLoaderFactory`. The following properties are recognized:

- `bootstrapAsynchronously=true | false`
  Determines whether the bootstrap happens in the background after the cache has started (true), or if bootstrapping must complete before the cache is made available (false). Default is `true`.
- `maximumChunkSizeBytes=[integer]`
  Caches can potentially grow larger than the memory limits on the JVM. This property enables the bootstrapper to fetch elements in chunks. The default chunk size is 5000000 (5 MB).

Global distributed cache settings

In a distributed cache, there is no master cache controlling all caches in the group. Instead, each cache is a peer in the group and needs to know where all the other peers in the group are located. *Peer Discovery* and *Peer Listeners* are two essential parts of any distributed cache system.

Edit global distributed cache settings for peer discovery

You can configure the global distributed cache settings by selecting the **Server Settings** node in the Policy Studio tree, and clicking **General** > **Cache**. You can configure the following fields:

Peer Provider Class:
By default, the built-in peer discovery class factory is used:

```
net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory
```

Properties Separator:
Specify the token used as the separator for the list of properties in the next field.

Properties:
The properties listed specify whether the peer discovery mechanism is automatic or manual. If the automatic mechanism is used, each peer uses TCP multicast to establish and maintain a multicast group. This is the default option because it requires minimal configuration and peers can be automatically added and removed from the group. Each peer pings the group every second. If a peer has not pinged any of the other peers after 5 seconds, it is dropped from the group, while a new peer is admitted to the group if it starts pinging the other peers.

To use automatic peer discovery, ensure that the `peerDiscovery` setting is set to `automatic`. You can specify the multicast address and port using the `multicastGroupAddress` and `multicastGroupPort` settings. You can specify the time to live for multicast datagrams using the `timeToLive` setting.

Alternatively, you can configure a manual peer discovery mechanism, whereby each peer definitively lists the peers it wants to communicate with. This should only be used in networks where there are problems propagating multicast datagrams. To use a manual peer discovery mechanism, ensure the `peerDiscovery` setting is set to `manual`. The list of RMI URLs of the other peers in the group must also be specified, for example:

```
rmiUrls=//server2:40001/sampleCache1|//server2:40001/sampleCache2
```

Peer Listener Class:
The peer listener class specified is responsible for listening for messages from peers in the group.

Properties Separator:
Specify the token used to separate the list of properties.

Properties:
The properties entered configure the way the listener behaves. Valid properties are as follows:

- **hostname (optional)**
  Host name of the machine on which the listener is listening.

  Note By default, this is set to `localhost`, which maps to the local loopback address of `127.0.0.1`, which is not addressable from another machine on the network. If you intend this cache to be used over the network, you should change this address to the IP address of the network interface on which the listener is listening.

- **port (mandatory)**
  Specify the port on which the listener is listening, which by default is 40001.
- **socketTimeoutMillis (optional)**
  Enter the number of seconds that client sockets wait when sending messages to this listener until they give up. The default is 2000 ms.

Notify replicators of removal of items during refresh:
A server refresh automatically purges all items from the cache (for example, when configuration updates are deployed to the API Gateway). If this check box is selected, the contents of each peer in the group are also purged. This avoids a situation where a single peer is refreshed (and has its contents purged), but the other peers in the group are not purged. If this option is not selected, the

refreshed peer attempts to bootstrap itself to the other peers in the group, resulting in the cache items becoming replicated in the refreshed cache. This effectively negates the effect of the server refresh and may result in inconsistent behavior.

Distributed caching example

This example describes how to configure Ehcache for three API Gateways:

- You have three API Gateway servers and two distributed caches.
- Each API Gateway server shares its cache with the other two servers.

Step 1 - Change Ehcache settings in Policy Studio

In Policy Studio, select the **Server Settings** node in the Policy Studio tree, and click **General** >**Cache**.

Edit the peer provider properties as follows:

```
peerDiscovery=manual,timeToLive=1,rmiUrls=${env.cache.rmiURL}
```

Edit the peer listener properties as follows:

```
hostName=${env.cache.IP},port=40001,socketTimeoutMillis=120000
```

Step 2 - Configure envSettings.props for each API Gateway server

Add the following settings to the `envSettings.props` file for each API Gateway server.

envSettings.props for API Gateway 1

```
env.cache.IP=10.0.0.1

env.cache.rmiURL=//10.0.0.2:40001/Cache1|//10.0.0.3:40001/Cache1|//10.0.0.2:40001/Ca
che2|//10.0.0.3:40001/Cache2
```

envSettings.props for API Gateway 2

```
env.cache.IP=10.0.0.2

env.cache.rmiURL=//10.0.0.1:40001/Cache1|//10.0.0.3:40001/Cache1|//10.0.0.1:40001/Ca
che2|//10.0.0.3:40001/Cache2
```

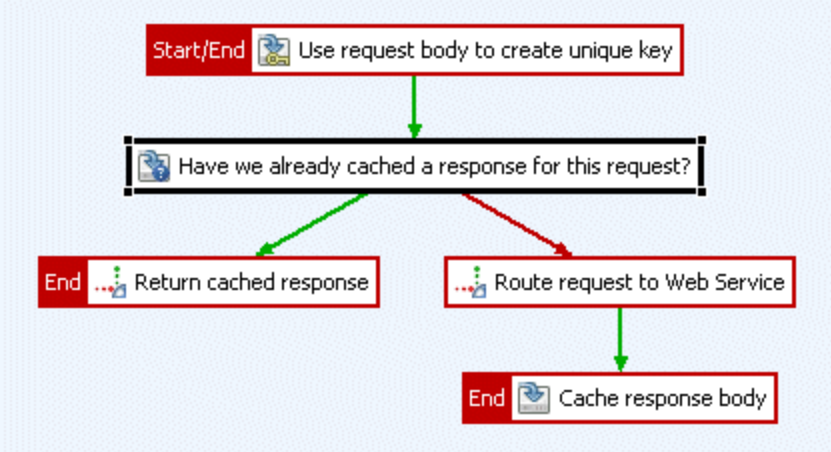
envSettings.props for API Gateway 3

```
env.cache.IP=10.0.0.3

env.cache.rmiURL=//10.0.0.2:40001/Cache1|//10.0.0.1:40001/Cache1|//10.0.0.2:40001/Ca
che2|//10.0.0.1:40001/Cache2
```

Example of caching response messages

This simple example shows how to construct a policy that caches responses from the service. It uses the request body to identify identical successive requests. If the API Gateway receives two successive requests with an identical message body, it returns the corresponding response from the cache instead of routing the request to the service. The following diagram illustrates the complete policy:



The logic of the policy is summarized as follows:

1. The purpose of the first filter is to configure what part of the request you want to use to identify unique requests. This example uses the request body as the unique key, which is then used to look up the appropriate response message from the cache.
2. The second filter looks up the request body in the response cache to see if it contains the request body. If it does, the response message that corresponds to this request is returned to the client.
3. If it does not, the request is routed to the service, which processes it (by connecting to a database over the network and running a SQL statement) and returns a response to the API Gateway.
4. The API Gateway then returns the response to the client and caches it in the response cache.
5. When the next identical request is received by the API Gateway, the corresponding response is located in the responses cache and returned immediately to the client.

You must configure the following caching filters to achieve this policy. For convenience, the routing filters are not included in this example because the configuration options depend on your target service.

Create Key

This filter is used to decide what part of the request is used for a request to be considered unique. Different parts of the request can be identified internally using message attributes (for example, `content.body` contains the request body). The following fields must be configured for this filter:

- **Name**: Use request body to create unique key
- **Attribute Name**: `content.body`
- **Output attribute name**: `message.key`

Is Cached?

This filter looks up the cache to see if a response has been stored for the current request. It looks up the cache using the `message.key` attribute by default. The `message.key` attribute contains a hash of the request message, and can be used as the key for objects in the cache. If the key is found in the cache, the value of the key (cached response for this request) is written to the `content.body` attribute, which can be returned to the client using the **Reflect** filter. You must configure the following fields:

- **Name**: Is a response for this request already cached?
- **Cache containing key**: Response Cache (assuming you have created a cache of this name)
- **Attribute Containing Key**: `message.key`
- **Overwrite attribute name if found**: `content.body`

Reflect

If the **Is Cached?** filter passes, it retrieves the response from the cache and stores it in the `content.body` message attribute. The **Reflect** filter is used to return the cached response to the client.

Routing

If a response for this request could not be located in the cache, the API Gateway routes the request to the service, and waits for a response. For more details on how to route messages, see *Get started with routing configuration* on page 35.

Cache Attribute

When the response has been received from the service, it should be cached for future use. The **Cache Attribute** filter is used to configure the key used to look up the cache and which aspect of the response message is stored as the key value in the cache.

Note This example specifies the value of the `content.body` attribute to cache. Because this filter is configured *after* the routing filters, this attribute contains the response message. The value entered in the **Attribute Key** field should match that entered in the **Attribute containing key** field in the **Is Cached?** filter.

You must configure the following fields:

- **Name**: Cache response body
- **Cache to use**: Response Cache
- **Attribute key**: `message.key`
- **Attribute name to store**: `content.body`

Further information

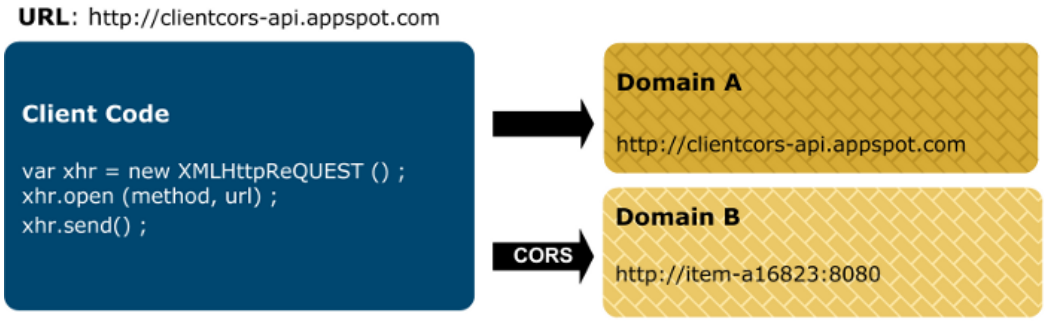
For more information on these filters, see the *API Gateway Policy Developer Filter Reference*.

Cross-Origin Resource Sharing

Overview

Cross-Origin Resource Sharing (CORS) enables client-side code running in a browser in a particular domain to access resources hosted in another domain in a secure manner. Cross-origin requests are typically not permitted by browsers, and CORS provides a framework in which cross-domain requests are treated as same-domain requests.

For example, using CORS, JavaScript embedded in a web page can make an HTTP `XMLHttpRequest` to a different domain. This is used to send an HTTP or HTTPS request to a web server, and to load the server response data back into the script. The following diagram shows an example CORS architecture:



This example is described as follows:

1. A user browses to the following URL:

   ```
   http://client.cors-api.appspot.com
   ```

2. The client page displayed on the left contains JavaScript that attempts to access resources in Domain A (`http://client.cors-api.appspot.com`), and in Domain B (`http://item-a16823:8080`).
3. Attempts by the browser to access resources in Domain A are granted because Domain A is the same domain in which the JavaScript code is running.
4. When the browser attempts to access resources in Domain B, it must use the CORS protocol because Domain B is in a different domain than that in which the JavaScript code is running. In this way, CORS enables client JavaScript code running in the browser in Domain A to access resources in Domain B.

The CORS standard provides CORS HTTP headers that enable servers to serve resources to permitted origin domains. Browsers support these CORS HTTP headers and enforce their restrictions. Browsers can also send *preflight* CORS requests to retrieve supported methods from the server using an HTTP `OPTIONS` method. Then on approval from the server, the browser client can send the request with the appropriate HTTP request method.

CORS request headers

The CORS HTTP request headers are as follows:

- `Origin`
- `Access-Control-Request-Method` (preflight only)
- `Access-Control-Request-Headers` (preflight only)

CORS response headers

The CORS HTTP response headers are as follows:

- `Access-Control-Allow-Origin`
- `Access-Control-Allow-Credentials`
- `Access-Control-Expose-Headers`
- `Access-Control-Max-Age` (preflight only)
- `Access-Control-Allow-Methods` (preflight only)
- `Access-Control-Allow-Headers` (preflight only)

Add a CORS profile

To enable CORS support in API Gateway, you must first add a CORS profile in Policy Studio:

1. In the Policy Studio tree, select **Environment Configuration** > **Libraries** > **CORS Profiles**.
2. Right-click and select **Add a CORS Profile**, and configure the settings on the following tabs.

General

Configure the following general settings:

- **Name**:
  Unique name of the CORS profile. Defaults to `Cross Origin Resource Sharing`.
- **Enable CORS**:
  Specifies whether CORS processing is enabled for the profile. Enabled by default.

Origins

The **Origins** table enables you to configure the list of origins that are allowed to access resources configured with this CORS profile, or exposed by a specific HTTP service. Click **Add** at the bottom right to add an origin.

You can specify origins using the following values:

- `*` (permits all values supplied in the CORS `Origin` header)
- Domain (for example, `http://client.corsapi.appspot.com`)
- Wildcarded value (for example, `http://*.appspot.com` or `http://client.corsapi.appspot.com:*`)

Allowed Headers

The **Allowed Headers** table enables you to configure the list of HTTP headers that are permitted when requesting a resource exposed by this CORS profile or HTTP service. The list of headers is defined by the value of the `Access-Control-Request-Headers` CORS header. Click the

Add button at the bottom right of the window to add an allowed header to the table.

Note The list of allowed headers is checked only during a CORS preflight `OPTIONS` request, which can be sent before access to the resource is granted.

Exposed Headers

The **Exposed Headers** table enables you configure the list of CORS HTTP response headers that are exposed to the client. Click the **Add** button at the bottom right of the window to add an exposed header to the table.

You can specify the list of CORS headers that are exposed to the client, in response to a resource exposed by this profile or HTTP service. You do not need to include the following simple HTTP response headers:

- `Cache-Control`
- `Content-Type`
- `Content-Language`
- `Expires`
- `Last-Modified`
- `Pragma`

Credentials Support

The **Support credentials** setting specifies whether resources using this CORS profile or HTTP service support user credentials. When this option is selected, the `Access-Control-Allow-Credentials` CORS header is sent in the response, with a value of `true`. This setting is not selected by default.

Preflight Results Cache

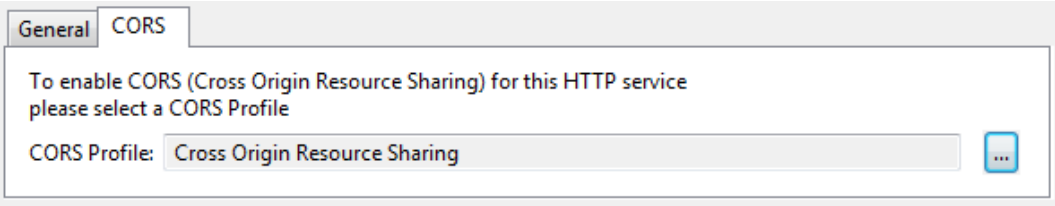
The **Max. age (seconds)** setting specifies how long the results of a CORS preflight `OPTIONS` request can be stored in the client preflight result cache. When this setting is configured, the `Access-Control-Max-Age` CORS header is sent in the response.

Configure CORS for HTTP services

You can also configure a CORS profile at the HTTP service level. This means that the settings configured for the profile are also applied to any child resolvers of this HTTP service. For example, this may be useful when using a third-party load balancer, and you need to configure a CORS profile for the default **API Portal** HTTP service, and specify the load balancer address as an origin.

To configure CORS at the HTTP service level, perform the following steps:

1. In the Policy Studio tree, select an HTTP service (for example, **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services**).
2. Right-click, and select **Edit** to display the **HTTP Services** dialog.
3. Select the **CORS** tab, and click the browse button to select a preconfigured CORS profile. By default, no profile is selected, which means that CORS is disabled.
4. In the **Select CORS Profile** dialog, if no profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its settings. For details on CORS settings, see *Add a CORS profile* on page 255.



For more details on HTTP services, see *Configure HTTP services* on page 275.

Configure CORS for relative paths

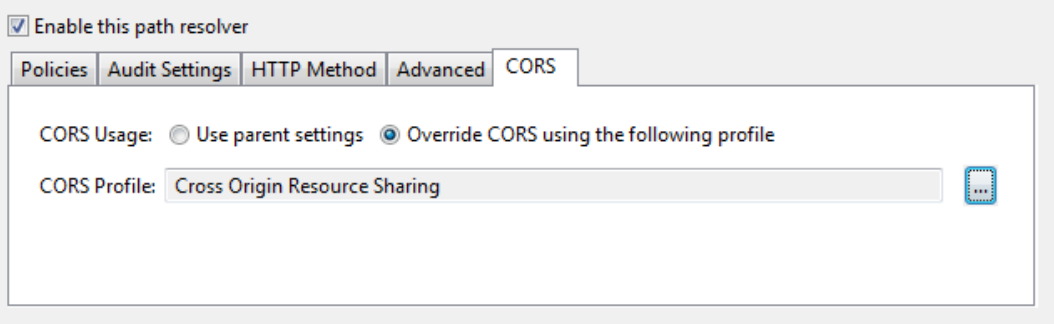
By default, the CORS profile set at the parent HTTP service level is used for all child resolvers of the HTTP service. However, you can override this at the relative path level as follows:

1. In the Policy Studio tree, select a list of relative paths (for example, **Environment Configuration** > **Listeners** > **API Gateway**> **Default Services** > **Paths**).
2. In the **Resolvers** window on the right, right-click a resolver, and select **Edit** to display the dialog.
3. Select the **CORS** tab, and in the **CORS Usage** field, select **Override CORS using the following profile**. By default, no CORS profile is selected, and the parent settings are used.

   Note Relative paths can act as HTTP services, and can accommodate child resolvers. This means that when a relative path has children, and has a CORS profile configured, by default, the children use the parent profile (unless a child overrides it).

4. In the **CORS Profile** field, click the browse button to select a preconfigured CORS profile.
5. If no CORS profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its

settings. For details on CORS settings, see *Add a CORS profile* on page 255.



Note Similarly, you can also override CORS for the following relative path resolvers:

- Static Content Providers
- Static File Providers
- Servlet Application

For more details on relative paths and resolvers, see *Configure relative paths* on page 293.

Key Property Store

A *Key Property Store* (KPS) is a table of data referenced by policies running on an API Gateway. Data in a KPS table is assumed to be read frequently and seldom written, and can be changed without incurring an API Gateway service outage. KPS tables are shared across an API Gateway group. Data can be stored in one of the following locations:

- External Apache Cassandra database—data can be distributed across multiple nodes to provide high availability (default).
- Relational database accessible to all API Gateways in the group.
- JSON files on the local file system (deprecated in this release).

A KPS is typically used to store property values used in policies on an API Gateway. KPS data is injected into policies using selectors created in Policy Studio. Selectors are evaluated and expanded dynamically at runtime. For example, a KPS table contains authorization tokens for different users. A policy looks up the token for the current user, and inserts it into an HTTP request.

Caution Do not edit the default KPS tables in Policy Studio unless under strict supervision from Axway Support. This includes the **API Server**, **OAuth**, or **API Portal** KPS tables available under **Environment Configuration > Key Property Stores**.

For more details on Key Property Stores, see the *API Gateway Key Property Store User Guide*.

KPS tables and collections

KPS tables are organized into *collections*. The tables in a collection typically have a relationship to each other. For example, the OAuth collection contains a set of tables that store all OAuth-related data. Every KPS table is assigned an *alias* so that it can be easily referred to in a policy or a REST

request.

Column	Type	Description
email	String	User email address. This is the primary key used to identify a row in the table.
password	String	User password. This confidential data is encrypted.
firstName	String	User first name.
lastName	String	User surname.
age	Integer	User age.

Enter data in a KPS table

You can enter data in a KPS table using the API Gateway Manager web console for viewing and modifying data. This is available in API Gateway Manager under **Settings > Key Property Stores**. The `kpsadmin` command-line tool also supports data entry in addition to other administrative functions. KPS data can also be read and written by remote programmatic clients using the KPS REST interface.

New values for encrypted fields are always transmitted to the server in the clear. For security, always use HTTPS when accessing KPS over its REST API (this is the default).

The following example shows some simple table data that has been entered in API Gateway Manager, and which follows the example structure in *KPS tables and collections* on page 258:

email	password	firstName	lastName	age
jdoe@acme.com	*****	John	Doe	21
jbloggs@acme.com	*****	Joe	Bloggs	42
jdupont@acme.com	*****	Jean	Dupont	33

In this example, the `email` column is the primary key. You can use this to look up and uniquely identify a row using a selector expression. For example, the following selector expression evaluates to `John`:

```
${kps.customers["jdoe@acme.com"].firstName}
```

The following selector expression evaluates to `42`:

```
${kps.customers["jbloggs@acme.com"].age}
```

For more details on selectors, see *Select configuration values at runtime* on page 421.

KPS data sources

A KPS provides a consistent interface to data that can be stored in different data sources. API Gateway supports the following KPS data sources:

- **External Apache Cassandra database** (default): Used across an API Gateway group to provide high availability in a production environment.
- **Relational Database**: Enables you to use your existing database (for example, Oracle, Microsoft SQL Server, IBM DB2, or MySQL). The following approaches to data storage are supported:
    - *Shared storage*—data for multiple KPS tables is stored in a single dedicated database table. This is the recommended approach.
    - *Per-table storage*—each KPS table is backed by a single database table.
- **JSON files on the local file system**: Suited to single API Gateway deployments. In a multi-API Gateway scenario, file replication or a shared disk is required to ensure all API Gateways use the same data.

Note If a file-based KPS table is shared across API Gateways, the API Gateways must be restarted after data has changed. File-based KPS is deprecated in this release.

Add a KPS collection

A KPS collection is a group of KPS tables. To add a KPS collection, perform the following steps: The newly created KPS collection is displayed on the window on the right.

1. In the Policy Studio tree, right-click **Environment Configuration > Key Property Stores**, and select **Add KPS Collection**.
2. Complete the following fields in the **Add KPS Collection** dialog:
    - **Name**: Enter the KPS name (for example, `CustomerCollection`).
    - **Description**: Enter a text description of your KPS collection.
    - **Alias prefix**: Leave this field blank.
    - **Default data source**: Select one of the following from the list:
        - **File**
        - **Cassandra**
        - **SQL Database**

      Defaults to **Cassandra**.

Edit a KPS collection

To edit a KPS collection, perform the following steps:

1. In the main Policy Studio tree, select the KPS collection to edit under **Environment Configuration > Key Property Stores**.
2. Click the **Properties** tab.
3. You can edit the **Name**, **Description**, or **Alias prefix** for the KPS collection as required.
4. To change the **Default data source**, click the browse button to display a tree of data sources, and select a new default data source from the tree.
5. You can also specify a cache for storage and retrieval of selector results. This improves selector read performance for storage backends such as databases. In the **Cache** field, click the browse button to display a tree of caches, and select a cache. Only local caches are supported.
6. You can add, edit, or delete KPS collection data sources on the **Data Sources** tab. For more information, see the following sections.
7. Click **Save** at the top right to save your changes.

Add a Cassandra data store

To add an Cassandra database data store to a selected KPS collection, perform the following steps:

1. On the **Data Sources** tab, select **Add > Add Cassandra** at the bottom right.
2. Complete the following fields in the **Add Cassandra Data Source** dialog:
   - **Name**: Enter the KPS name (for example, `Customer Cassandra Data Source`).
   - **Description**: Enter a text description of your SQL database data source.
   - **Read Consistency Level**: Select the consistency level for Cassandra read operations from the list. Defaults to `ONE`.
   - **Write Consistency Level**: Select the consistency level for Cassandra write operations from the list. Defaults to `ONE`.

     For more details, see the following:
     - http://docs.datastax.com/en/archived/cassandra/2.2/cassandra/dml/dmlConfigConsistency.html
3. Click **OK**.

For more details on installing and configuring Apache Cassandra, see the *API Gateway Installation Guide*.

Add a database data store

To add an SQL database data store to a selected KPS collection, perform the following steps:

1. On the **Data Sources** tab, select **Add > Add Database** at the bottom right.
2. Complete the following fields in the **Add File Data Source** dialog:
   - **Name**: Enter the KPS name (for example, `Customer DB Data Source`).
   - **Description**: Enter a text description of your SQL database data source.
   - **Database Connection**: Click the button on the right, select a database connection in the dialog (for example, `Default Database Connection`), and click **OK**.

     You can add more database connections to the list by right-clicking **Environment Configuration > External Connections > Database Connections** in the Policy Studio tree, and selecting **Add a Database Connection**.
3. Click **OK**.

Add a file data store

To add a file-based data store to a selected KPS collection, perform the following steps:

1. On the **Data Sources** tab, select **Add > Add File** at the bottom right.
2. Complete the following fields in the **Add File Data Source** dialog:
   - **Name**: Enter the KPS name (for example, `Customer File Data Source`).
   - **Description**: Enter a text description of your file data source.
   - **Directory Path**: Enter the full directory path (for example, `c:\kpsdata`). Each table in the collection has its own JSON file in this directory.
3. Click **OK**.

Add a KPS table

To add a KPS table to a KPS collection, perform the following steps:

1. In the main Policy Studio tree, right-click a KPS collection (for example **CustomerCollection**), and select **Add Table**.
2. Complete the following fields in the **Add KPS Table** dialog:
   - **Name**: Enter the KPS name (for example, `Customers`).
   - **Description**: Enter a text description of your KPS.
   - **Aliases**: Click **Add**, and enter an alias used to identify your KPS (for example, `customers`), and click **OK**. Every KPS must have at least one alias.
3. Click **OK**.

Define the KPS table structure

To define the KPS table structure, perform the following steps:

1. In the main Policy Studio tree, select a KPS table (for example **Customers**), and click the **Structure** tab.
2. Click **Add**, and complete the following fields in the **Add Property** dialog:
   - **Name**: Enter the name of the table column (for example, `email`).
   - **Type**: Select the data type from the list (for example, `java.lang.String`).
   - **Key**: For `java.util.Map` types, select the key type from the list (for example, `java.lang.Integer`).
   - **Value**: For `java.util.Map` and `java.util.List` types, select the value type from the list (for example, `java.lang.Boolean`).
3. Click **OK**. The newly created KPS table column is added to the **Structure** tab.
4. Select **Primary Key** to make a field the primary key for the table.
5. Select **Autogenerated** to auto-generate a field in the KPS data source.
6. Select **Encrypted** to encrypt a field in the KPS data source.
7. Select **Indexed** to index a field in the KPS data source.
8. Repeat the preceding steps to add more table columns.
9. At the bottom, enter names of one or more properties used to look up this table from a selector (for example, `firstName,surname`). If none are specified, selectors can access the table using the primary key.
10. Click **Save** at the top right to save your changes.

API Gateway instances 8

This section describes how to configure API Gateway instances.

Configure API Gateway instances

Overview

This topic shows how to configure a running instance of the API Gateway. You can configure the options described in the following sections on the API Gateway instance in the Policy Studio tree (for example, under **Environment Configuration** > **Listeners** > **API Gateway**).

Add remote hosts

Remote host settings configure the way in which the API Gateway routes to another host machine. For example, if a destination server may not fully support HTTP 1.1, you can configure **Remote Host** settings for the server to optimize the way in which the API Gateway sends messages to it. Similarly, if the server requires an exceptionally long timeout, you can configure this in the **Remote Host** settings.

For more details, see *Configure remote host settings* on page 268.

Add HTTP services

You can add a container for HTTP-related services, including HTTP and HTTPS Interfaces, Directory Scanners, Static Content Providers, Servlet Applications, and Packet Sniffers.

HTTP services act as a container for all HTTP-related interfaces to the API Gateway's core messaging pipeline. You can configure HTTP and HTTPS interfaces to accept plain HTTP and SSL messages respectively. A relative path interface is available to map requests received on a particular URI or path to a specific policy. The Static Content Provider interface can retrieve static files from a specified directory, while the Servlet Application enables you to deploy servlets under the service.

Finally, the Packet Sniffer interface can read packets directly of the network interface, assemble them into HTTP messages, and dispatch them to a particular policy. *Configure HTTP services* on page 275 explains how to configure the available HTTP interfaces.

Add SMTP services

Simple Mail Transfer Protocol (SMTP) support enables the API Gateway to receive email and to act as a mail relay. The API Gateway can accept email messages using the SMTP protocol, and forward them to a mail server. You can also configure optional policies for specific SMTP commands (for example, `HELO/EHLO` and `AUTH`).

Configure SMTP services on page 318 explains how to configure SMTP services, interfaces, and handler policies.

Add file transfer services

You can configure the API Gateway to listen for remote clients that connect to it as a file server. This enables the API Gateway to apply configured policies on transferred files (for example, for schema validation, threat detection or prevention, routing, and so on). The API Gateway supports File Transfer Protocol (FTP), FTP over SSL (FTPS), and Secure Shell FTP (SFTP).

Configure a file transfer service on page 334 explains how to configure the API Gateway as a file transfer service.

Add policy execution scheduling

Policy execution scheduling enables you to schedule the execution of any policy on a specified date and time in a recurring manner. The API Gateway provides a preconfigured library of schedules to select from. You can also add your own schedules to the library.

Policy execution scheduling on page 241 explains how to add a policy execution schedule, and how to add schedules.

Configure JMS messaging system

You can configure the API Gateway to read JMS messages from a JMS queue or topic, run them through a policy, and then route onwards to a web service or JMS queue or topic.

The API Gateway can consume a JMS queue or topic as a means of passing XML messages to its core message processing pipeline. When the message has entered the pipeline, it can be validated against all authentication, authorization, and content-based message filters. Having passed all configured message filters, it can be routed to a destination web service over HTTP, or it can be dropped back on to a JMS queue or topic using the **Messaging System** connection filter.

For more details, see *Configure messaging services* on page 151.

Add Amazon SQS queue listener

The **Amazon SQS Queue Listener** enables you to poll an Amazon SQS queue for messages at a specified rate. When messages are retrieved from the queue, they can be passed to a specified policy for processing.

For more details, see *Configure Amazon SQS queue listener* on page 351.

Add FTP poller

The **FTP Poller** enables you to query and retrieve files by polling a remote file server. When files are retrieved, they can be passed into the API Gateway core message pipeline for processing. For example, this is useful in cases where an external application drops files on to a remote file server, which can then be validated, modified, or routed on over HTTP or JMS by the API Gateway.

For more details, see *Configure an FTP poller* on page 342.

Add directory scanner

The **Directory Scanner** reads XML files from a specified directory and dispatches them to a selected policy. This enables you to search a local directory for XML files, which can then be fed into a security policy for validation. Typically, XML files are transferred by FTP or saved to the file system

by another application. The API Gateway can then pick these files up, run the full array of authentication, authorization, and content-based filters on the messages, and then route them over HTTP or JMS to a back-end system.

For more details, see *Configure a directory scanner* on page 345.

Add POP client

The **POP Client** enables you to poll a POP mail server to read email messages from it, and pass them into a policy for processing.

For more details, see *Configure a POP client* on page 348.

Configure TIBCO

You can configure a TIBCO Rendezvous listener for real-time messaging.

For more details, see *TIBCO Rendezvous listener* on page 350.

API Gateway settings

You can configure per-instance global configuration settings by clicking the **Environment Configuration** > **Server Settings** node in the Policy Studio tree. For example, these include settings for timeouts, caches, logging, monitoring, security, and so on.

For more details on configuring API Gateway instance settings, see the *API Gateway Administrator Guide*.

Cryptographic acceleration

The API Gateway can leverage the OpenSSL Engine API to offload complex cryptographic operations (for example, RSA and DSA) to a hardware-based cryptographic accelerator, and to act as an extra layer of security when storing private keys on a Hardware Security Module (HSM).

The API Gateway uses OpenSSL to perform cryptographic operations, such as encryption and decryption, signature generation and validation. OpenSSL exposes an *Engine API*, which enables you to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL. OpenSSL can, when configured appropriately, call the engine's implementation of these operations instead of its own.

For more information on configuring API Gateway to use an OpenSSL engine, see *Cryptographic acceleration* on page 355.

Configure remote host settings

You can use the **Remote Host Settings** to configure the way in which API Gateway connects to a specific external server or routing destination. For example, typical use cases for configuring remote hosts with API Gateway are as follows:

- Allowing API Gateway to send HTTP 1.1 requests to a destination server when that server supports HTTP 1.1.
- Resolving inconsistencies in the way the destination server supports HTTP.
- Mapping a host name to a specific IP address or addresses (for example, if a DNS server is unreliable or unavailable).
- Setting the timeout, session cache size, input/output buffer size, and other connection-specific settings for a destination server (for example, if the destination server is particularly slow, you can set a longer timeout).
- Stop accepting inbound connections on the HTTP interface when API Gateway loses connectivity to the remote host.
- Set timeouts for incoming connections, for example, to configure long polling. For more information, see *Configure an incoming remote host* on page 273.

You can add remote hosts *per-instance* to the API Gateway instance in the Policy Studio tree. For example, select **Environment Configuration** > **Listeners** > **API Gateway** >, right-click this instance, and select **Add Remote Host**. The tabs in the **Remote Host Settings** configuration window are described in the following sections.

General settings

You can configure the following settings on the **General** tab:

Host alias:
The human readable alias name for the remote host (for example, `StockQuote Host`). This setting is required.

Host name:
The host name or IP address of the remote host to connect to (for example `stockquote.com`). If the host name entered in a **Static Router** filter matches this host name, the connection-specific settings configured on the **Remote Host** dialog are used when connecting to this host. This also includes any IP addresses listed on the **Addresses and Load Balancing** tab, which override the default network DNS server mappings, if configured. This setting is required.

Port:
The TCP port on the remote host to connect to. Defaults to `80`.

Maximum connections:
The maximum number of connections to open to a remote host. If the maximum number of connections has already been established, the API Gateway instance waits for a connection to drop or become idle before making another request. The default value is `-1`, which allows unlimited connections. In the absence of a remote host, a global default value of `128` applies.

Allow HTTP 1.1:
The API Gateway uses HTTP 1.0 by default to send requests to a remote host. This prevents any anomalies if the destination server does not fully support HTTP 1.1. If the API Gateway is routing on to a remote host that fully supports HTTP 1.1, you can use this setting to enable API Gateway to use HTTP 1.1.

Include Content Length in Request:
When this option is selected, the API Gateway includes the `Content-Length` HTTP header in all requests to this remote host. This setting only applies to outgoing remote host connections.

Include Content Length in Response:
When this option is selected, if the API Gateway sends a response to this remote host that contains a `Content-Length` HTTP header, it returns this length to the client. This setting only applies to incoming remote host connections.

Send Server Name Indication TLS extension to server:
Adds a field to outbound TLS/SSL calls that shows the name that the client used to connect. This can be useful if the server handles several different domains, and needs to present different certificates depending on the name the client used to connect. For example, this is required by some cloud-based services such as Amazon CloudFront. This setting is not selected by default.

Note To send the SNI extension, you must ensure that the **Verify server's certificate matches requested hostname** setting is also selected. In addition, the **Port** setting must be the port that you are connecting to the server with (for example, `443` is the default port for SSL).

Verify server's certificate matches requested hostname:
Ensures that the certificate presented by the server matches the name of the remote host being connected to. This prevents host spoofing and man-in-the-middle attacks. This setting is selected by default.

Address and load balancing settings

You can configure the following settings on the **Addresses and Load Balancing** tab:

Addresses to use instead of DNS lookup:
You can add a list of IP addresses that the API Gateway uses instead of attempting a DNS lookup on the host name provided. This is useful in cases where a DNS server is not available or is unreliable. By default, connection attempts are made to the listed IP addresses on a round-robin basis.

For example, if a **Static Router** filter is configured to route to `www.webservice.com`, it first checks if any remote hosts have been configured with a **Host Name** entry matching `www.webservice.com`. If it finds a **Remote Host** with matching **Host Name** , it resolves the host name to the IP addresses listed here. In addition, it uses all the connection-specific settings configured on the **Remote Host** dialog when routing messages to these IP addresses. If it cannot find a matching host, the **Static Router** filter uses whatever DNS server has been configured for the network on which the API Gateway is running.

To add a list of IP addresses for a remote host, perform the following steps:

1. In the **Addresses to use instead of DNS lookup** box, select a priority group (for example, **Highest Priority**).
2. Click **Add**.
3. Enter an IP address or server name in the **Configure IP Address** dialog.
4. Click **OK**.
5. Repeat these steps to add more IP addresses as appropriate.

Load balancing:
The **Load Balancing Algorithm** drop-down box enables you to specify whether load balancing is performed on a simple round-robin basis or weighted by response time. **Simple Round Robin** is the default algorithm. Connection attempts are made to the listed IP addresses on a round-robin basis in each priority group. The **Weighted by response time** algorithm compares the request/reply response times for the server address in each priority group. This is the simplest way of estimating the relative load of the address.

This algorithm works as follows:

1. The address with the least response time is selected to send the next message to.
2. If the address fails to send the message, it ignores that address for a period of time and selects another address in the same way.
3. If all addresses in a given group fail to accept a connection, addresses in the next group in ascending order of priority are used in the same way.
4. Only when all addresses in all priorities have failed to accept connections is delivery of the message abandoned, and an error raised.

The response times used by this algorithm decline over time. You can specify the rate of exponential decline by specifying a **Period to wait before response time is halved**. The default is 10,000 ms (10 sec). This enables addresses that were heavily loaded for a period of time to eventually resume accepting messages after the load subsides.

For example, server A takes 100 ms to reply, and the other servers in the same priority group reply in 25 ms. A **Period to wait before response time is halved** of 10,000 ms (10 sec) means that after 20 seconds server A is retried along with the other servers. In this case, the response time has been halved twice (100 ms / 2 / 2 = 25 ms).

Advanced settings

The options available on the **Advanced** tab are used when creating sockets for connecting to the remote host. Default values are provided for all fields, which should only be modified under advice from the Axway Support.

You can configure the following configuration options on the **Advanced** tab:

Connection Timeout:
If a connection to this remote host is not established within the time set in this field, the connection times out and the connection fails. Defaults to 30000 milliseconds (30 seconds).

Active Timeout:
When API Gateway receives a large HTTP request, it reads the request off the network when it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout**, API Gateway closes the connection. This prevents a remote host from closing the connection while sending data. Defaults to 30000 milliseconds (30 seconds).

For example, the remote host's network connection is pulled out of the machine while sending data to API Gateway. When API Gateway has read all the available data off the network, it waits the **Active Timeout** period before closing the connection.

The **Active Timeout** value is also used as a wait time when the maximum number of connections for a remote host is reached. For example, when a remote host reaches the **Maximum connections** value, API Gateway waits the active timeout period before giving up on trying to make a new connection.

Transaction Timeout (ms):
A configurable transaction timeout that detects slow HTTP attacks (slow header write, slow body write, slow read) and rejects any transaction that keeps the worker threads occupied for an excessive amount of time. The default value is 240000 milliseconds.

Max Received Bytes:
The maximum number of bytes received in a transaction. This is a configurable maximum length for the received data on transactions that API Gateway can handle. This setting limits the entire amount of data received over the link, regardless of whether it consists of body, headers, or request line. The default value is 10 MB (10485760 bytes).

Max Sent Bytes:
The maximum number of bytes sent in a transaction. This is a configurable maximum length for the transmitted data on transactions that API Gateway can handle. This setting limits the entire amount of data sent over the link, regardless of whether it consists of body, headers, or request line. The default value is 10 MB (10485760 bytes).

Idle Timeout:
The API Gateway supports HTTP 1.1 persistent connections. The **Idle Timeout** is the time that API Gateway waits after sending a message over a persistent connection to the remote host before it closes the connection. Defaults to 15000 milliseconds (15 seconds).

Typically, the remote host tells the API Gateway that it wants to use a persistent connection. The API Gateway acknowledges this, and keeps the connection open for a specified period of time after sending the message to the host. If the connection is not reused by within the **Idle Timeout** period, the API Gateway closes the connection.

Input Buffer Size:
The maximum amount of memory allocated to each request. The default value is 8192 bytes.

Output Buffer Size:
The maximum amount of memory allocated to each response. The default value is 8192 bytes.

Cache addresses for (ms):
The period of time to cache addressing information after it has been received from the naming service (for example, DNS). The default value is 300000 milliseconds.

SSL Session Cache Size:
Specifies the size of the SSL session cache for connections to the remote host. This controls the number of idle SSL sessions that can be kept in memory. Defaults to `32`. If there are more than 32 simultaneous SSL sessions, this does not prevent another SSL connection from being established, but means that no more SSL sessions are cached. A cache size of `0` means no cache, and no outbound SSL connections are cached.

Tip You can use this setting to improve performance because it caches the slowest part of establishing the SSL connection. A new connection does not need to go through full authentication if it finds its target in the cache.

At `DEBUG` level or higher, the API Gateway outputs trace when an entry goes into the cache, for example:

```
DEBUG   09:09:12:953 [0d50] cache SSL session 11AA3894 to support.acme.com:443
```

If the cache is full, the output is as follows:

```
DEBUG   09:09:12:953 [0d50] enough cached SSL sessions 11AA3894 to
support.acme.com:443 already
```

Input Encodings:
Click the browse button to specify the HTTP content encodings that the API Gateway can accept from peers. The available content encodings include `gzip` and `deflate`. By default, the content encodings configured the **Default Settings** are used. You can override this setting at the remote host and HTTP interface levels. For more details, see *Compressed content encoding* on page 430.

Output Encodings:
Click the browse button to specify the HTTP content encodings that the API Gateway can apply to outgoing messages. The available content encodings include `gzip` and `deflate`. By default, the content encodings configured the **Default Settings** are used. You can override this setting at the remote host and HTTP interface levels. For more details, see *Compressed content encoding* on page 430.

Include correlation ID in headers:
Specifies whether to insert the correlation ID in outbound messages. This means that an `X-CorrelationID` header is added to the outbound message. This is a transaction ID that is attached to each message transaction that passes through API Gateway, and which is used for traffic monitoring in the API Gateway Manager web console. You can use the correlation ID to search for messages in the web console, and you can also access its value from a policy using the `id` message attribute. This setting is selected by default.

Configure watchdogs

You can configure an HTTP interface to shut down based on certain *conditions*. One such condition is dependent on the API Gateway being able to contact a particular back-end web service running on a remote host. To do this, you can configure an **HTTP Watchdog** for a remote host to poll the endpoint. If the endpoint cannot be reached, the HTTP interface is shut down.

To configure the API Gateway to shut down an HTTP interface based on the availability of a remote host, perform the following steps:

1. Configure an **HTTP Watchdog** for the remote host.
2. Configure a **Requires Endpoint** condition on the HTTP interface.
3. When configuring this condition, select the remote host configured in step 1 (the host with the associated Watchdog).

Note When **Load Balancing** is configured as **Weighted by response time**, and remote host watchdogs are configured, the watch dog polling also contributes to the load balancing calculations.

For more information on adding a watchdog to a remote host, see *Configure HTTP watchdog* on page 273. For more information on adding conditions to an HTTP interface, see *Configure conditions for HTTP interfaces* on page 288.

Configure an incoming remote host

A remote host is normally used to configure specific connection features for the outward connection, that is, for the connection from API Gateway to the back-end service. However, you can also configure a remote host for an incoming connection, that is, for the connection from the client to API Gateway.

To configure an incoming remote host, configure the following settings on the **General** tab of the remote host settings:

1. Enter `incoming` in the **Port** field.

   For an incoming connection, the port is referring to the remote address of the TCP connection. Incoming connections arrive from effectively arbitrary remote ports, so this acts as a wildcard for all incoming connections.
2. Enter the IP address of the host in the **Host name** field, rather than the DNS name.

   A CIDR style netmask can be specified (for example, `192.168.0.0/24` matches any address in the `192.168.0.x` range). This works on a longest-match basis if more than one network specification matches the client.

Configure HTTP watchdog

Overview

An HTTP Watchdog can be added to a Remote Host configuration in order to periodically poll the Remote Host to check its availability. For example, if the Remote Host becomes unavailable for some

reason, an HTTP interface can be brought down and can stop accepting requests. Once the Remote Host comes back online, the HTTP interface automatically starts up and starts accepting requests again.

To learn more about the reasons for shutting down an HTTP interface if certain *conditions* do not hold, see *Configure conditions for HTTP interfaces* on page 288.

To configure an HTTP Watchdog, right-click a previously configured Remote Host in the Policy Studio tree (for example, under **Environment Configuration** > **Listeners** > **API Gateway**). Then select **Watchdog** > **Add**, and configure the settings in the dialog.

Configuration

Configure the following settings:

Valid HTTP Response Code Ranges:
You can use this section to specify the HTTP response codes that you can regard as proof that the Remote Host is available. For example, if a 200 OK HTTP response is received for the poll request, the Remote Host can be considered available.

To specify a range of HTTP status codes, click the **Add** button and enter the **Start** and **End** of the range of HTTP response codes in the fields provided. An exact response code can be specified by entering the response code in both fields (for example, `200`).

HTTP Request for Polling:
The fields in this section enable you to configure the type and URI of the HTTP request to poll the Remote Host with. The default is the *Options* HTTP command with a URI of `*`, which is typically used to retrieve status information about the HTTP server. To use an alternative HTTP request to poll the Remote Host, select an HTTP request method from the **Method**, and specify the **URI** field.

Remote Host Polling:
The settings in this section determine when and how the HTTP Watchdog polls the Remote Host. The **Poll Frequency** determines how often the Watchdog is to send the polling request to the Remote Host.

By default, the Watchdog uses real HTTP requests to the Remote Host to determine its availability. In other words, if the API Gateway is sending a batch of requests to the Remote Host, it uses the response codes from these requests to decide whether or not the Remote Host is up. Therefore, the Watchdog effectively "polls" the Remote Host by sending real HTTP requests to it.

To configure the Watchdog to send poll requests during periods when it is not sending requests to and receiving responses from the Remote Host, select **Poll if up**. In this case, the Watchdog uses real HTTP requests to poll the Remote Host as long as it sends them, but starts sending test poll requests when it is not sending HTTP requests to the Remote Host to test its availability.

Note When a Remote Host is deemed to be down (an invalid HTTP response code was received) the Watchdog continues to poll it at the configured **Poll Frequency** until it comes back up again (until a valid HTTP response code is received).

Configure HTTP services

This topic describes the function and configuration of the following HTTP services:

API Gateway uses *HTTP services* to handle traffic from various HTTP-based sources. The available HTTP services are as follows:

- **HTTP interfaces**: *HTTP interfaces* define the ports and IP addresses on which API Gateway listens for incoming requests. You can also add *HTTPS interfaces* to specify SSL certificates to authenticate to clients, and certificates considered trusted for establishing SSL connections with clients. See *HTTP and HTTPS interfaces* on page 277
- **Relative path**: You can configure *relative paths* so that when a request is received on a specific path, API Gateway can map it to a specific policy, or chain of policies. For more details, see *Configure relative paths* on page 293.
- **Static content provider**: You can use a *static content provider* to serve static HTTP content from a particular directory. In this case, API Gateway is effectively acting as a web server. For more details, see *Static content providers* on page 300.
- **Servlet applications**: API Gateway can act as a servlet container, which enables you to drop servlets into the HTTP services configuration. This should only be used by developers with very specific requirements and under strict advice from the Axway Support. For more details, see *Servlet applications* on page 302.
- **Packet sniffer**: You can add a *packet sniffer* to intercept network packets from the client, assemble them into complete HTTP messages, and log these messages to an audit trail. Because the packet sniffer operates at the network layer (unlike an HTTP-based traffic monitor at the application layer), the packets are intercepted transparently to the client. This means that the packet sniffer is a *passive service*, which is typically used for management and monitoring instead of general policy enforcement. For more details, see *Packet sniffers* on page 286.

HTTP services groups

An *HTTP services group* is a container around one or more HTTP services. Usually, an HTTP services group is configured for a particular type of HTTP service. For example, you could have an **HTTP Interfaces** group that contains the configured HTTP interfaces, and another **Static Providers** group to manage static content providers. While organizing HTTP services by type eases the task of managing services, API Gateway is flexible enough to enable administrators to organize services into groups according to whatever scheme best suits their requirements.

This section describes a scenario where HTTP services groups can prove useful, and how to use separate services groups to process, for example, SSL traffic on a different channel.

HTTP interfaces and relative paths

An HTTP services group must have at least one HTTP interface together with at least one relative path. The HTTP interface determines which TCP port API Gateway listens on, and the relative path maps a request received on a particular path (request URI) to specific policies. You can add several HTTP interfaces to a group, in which case requests received on any one of the opened ports are processed in the same manner. For example, `http://<HOST>:8080/test` and `http://<HOST>:8081/test` requests can both be processed by the same policy (mapped to the `/test` relative path).

You can also add multiple relative paths to a HTTP services group, where each path is bound to a specific policy or chain of policies. For example, if a request is made to `http://<HOST>:8080/a`, it is processed by Policy A. If a request is made to `http://<HOST>:8080/b`, it is handled by Policy B. Requests made to the other interface are also processed by the same policy, meaning that a request made to `http://<HOST>:8081/b` is also handled by Policy B.

This means that relative paths configured under a HTTP services group are bound to all HTTP interfaces configured for that group. If you have two interfaces listening on ports `8080` and `8081`, API Gateway handles requests to `http://<HOST>:8080/a` and `http://<HOST>:8081/a` identically.

Example configuration

In this example configuration, a SSL validation policy is added to process requests to `http://<HOST>:443/a`, while the existing Schema Validation policy continues to handle requests for `http://<HOST>:8080/a`. To distinguish between receiving requests on the two different ports, a new `SSL HTTP Services Group` is added alongside the existing `HTTP Services Group`. The new group opens a single HTTPS interface that listens on the SSL port `443`, and is configured with a relative path of `/a` to handle requests on this path:

Service Group	HTTP Port	Relative Path	Policy
`HTTP Services Group`	`8080`	`/a`	Schema Validation Policy
`SSL HTTP Services Group`	`443`	`/a`	SSL Validation Policy

Using HTTP services groups in this way, you can configure API Gateway to dispatch requests received on the same path (for example, `/a`) to different policies depending on the port where the request was accepted.

Default HTTP services groups

By default, API Gateway ships with preconfigured HTTP services groups (for example, **Default Services** and **Sample Services**). These groups contain some generic default policies you can use with an out-of-the-box installation of API Gateway.

In addition to the preconfigured services groups, you can add new HTTP services groups to dispatch requests to different policies based on the port on which the requests are received.

For details on the default service group used by the Admin Node Manager and API Gateway Analytics, see *Management services* on page 283.

Add an HTTP services group

1. In Policy Studio, click **Environment Configuration > Listeners**.
2. Right-click **API Gateway**, and select **Add HTTP Services**.
3. Enter a name for the group.
4. By default, Cross Origin Resource Sharing (CORS) is disabled because no profile is selected. To enable CORS for the HTTP services group, on the **CORS** tab, select a preconfigured CORS profile. If no profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. To edit an existing profile, right-click the profile and select **Edit**. For details on CORS settings, see *Add a CORS profile* on page 255.

   For more details on CORS, see *Cross-Origin Resource Sharing* on page 253.

When you have created an HTTP services group, you can configure it with the HTTP services described in this topic.

HTTP and HTTPS interfaces

An HTTP interface defines the address and port that API Gateway listens on. There are two types of interface: HTTP and HTTPS. The HTTP interface handles standard, non-authenticated HTTP requests, while the HTTPS interface can accept mutually authenticated SSL connections.

Before you can configure an interface, you must first configure a HTTP services group. See *HTTP services groups* on page 275.

Add HTTP or HTTPS interface

To create an HTTP interface, in Policy Studio tree, click **Environment Configuration > Listeners > API Gateway**, select the HTTP services group (for example, **Default Services**), right-click the **Ports** node, and select **Add HTTP**or **Add HTTPS**.

Configure Network settings

The following fields on the **Network** tab are common to both HTTP and HTTPS interfaces and must be configured:

- **Port**: The port number that API Gateway listens on for incoming HTTP requests.
- **Address**: The IP address or host of the network interface on which instance listens.

For example, you can configure the instance to listen on port `80` on the external IP address of a machine, while having a web server running on the same port but on the internal IP address of the same machine. By entering `*`, the instance listens on all interfaces available on the machine hosting API Gateway.

- **Protocol**: Select the Internet Protocol version (IPv) that this interface uses. You can select `IPv4`, `IPv6`, or both of these protocol versions. The default is `IPv4`.
- **Trace level**: The level of trace output. The possible values in order of least verbose to most verbose output are:
  - `FATAL`
  - `ERROR`
  - `INFO`
  - `DEBUG`
  - `DATA`

  The default trace level is read from the system settings.
- **Enable interface**: This setting is enabled by default. If you want to disable the HTTP interface, deselect this setting.

Configure Traffic Monitor settings

The fields on the **Traffic Monitor** tab are common to both HTTP and HTTPS interfaces. To override the system-level settings at HTTP or HTTPS interface level, select **Override settings for this port**, and configure the relevant options. For more details, see the *API Gateway Administrator Guide*.

Configure Advanced settings

The following fields on the **Advanced** tab are common to both HTTP and HTTPS interfaces and must be configured:

- **Backlog**: When API Gateway is busy handling concurrent requests, the operating system can accept additional incoming connections. In such cases, a backlog of connections can build up while the operating system waits for the instance to finish handling current requests. The specified value is the maximum number of connections API Gateway instance allows the operating system to accept and queue up until the instance is ready to read them. The larger the backlog, the larger the memory usage. The smaller the backlog, the greater the potential for dropped connections.
- **Idle Timeout**: API Gateway supports the use of HTTP 1.1 persistent connections. Typically, a client informs API Gateway that it wants to use a persistent connection. API Gateway acknowledges this and keeps the connection open for a certain time after sending the response to the client. If the client does not reuse the connection by sending up another request within the timeout period, API Gateway closes the connection. The **Idle Timeout** value is the time (in milliseconds) that API Gateway waits after sending a response over a persistent connection before closing the connection. The default value is 60000 milliseconds (60 seconds).

- **Active Timeout**: When API Gateway receives a large HTTP request, it reads the request off the network as it becomes available. If the time between reading successive blocks of data exceeds the **Active Timeout** value, API Gateway closes the connection. For example, if the client loses network connection while sending the data, instead of being tied to the transaction for a long time, API Gateway first reads all the available data off the network and then waits the **Active Timeout** period before closing the connection. The default value is 60000 milliseconds (60 seconds).
- **Maximum Memory per Request**: The maximum amount of memory in bytes that API Gateway allocates to each request. For more details, see the *API Gateway Administrator Guide*.
- **Input Encodings**: The HTTP content encodings that API Gateway can accept from peers. By default, the content encodings configured in the **Default Settings** are used. You can override this setting at the HTTP interface level and in the **Remote Host Settings**. For more details, see *Compressed content encoding* on page 430.
- **Output Encodings**: The HTTP content encodings that API Gateway can apply to outgoing messages. By default, the content encodings configured in the **Default Settings** are used. You can override this setting at the HTTP interface level and in the **Remote Host Settings**. For more details, see *Compressed content encoding* on page 430.
- **Transparent Proxy - allow bind to foreign address**: This enables you to use API Gateway as a *transparent proxy* on Linux systems with the `TPROXY` kernel option set. When selected, the value in the **Address** field can specify any IP address, and API Gateway handles incoming traffic for the configured address/port combinations. For more details and an example, see *Configure a transparent proxy* on page 290.
- **Include correlation ID in headers**: This specifies whether to insert the correlation ID (for example, `Id-54bbc74f515d52d71a4c0000`) in outbound messages. For the HTTP transport, this means that an `X-CorrelationID` header is added to the outbound message. This is a transaction ID that is tagged to each message transaction that passes through API Gateway. You can use the correlation ID to search for messages when monitoring traffic in the API Gateway Manager web console. You can also access the its value using the `id` message attribute in an API Gateway policy. This setting is selected by default.
- **Threat Protection Settings**: This specifies the **Threat Protection Profile** that is used to protect this interface with Apache ModSecurity threat protection rules. ModSecurity is a toolkit for real-time HTTP traffic monitoring, logging, and access control, which helps to mitigate application-level threats to APIs. The ModSecurity engine is embedded in API Gateway to provide API firewalling. If no threat protection profiles have been configured, right-click the **Threat Protection Profiles** node in the dialog, and select **Add a Threat Protection Profile**. For more details, see "Manage API firewalling" in the *API Gateway Administrator Guide*.

Configure HTTPS-specific settings

The following settings apply only to HTTPS interfaces and are not visible when creating a HTTP interface.

Network settings

In addition to the fields configured for an HTTP interface on the **Network** tab, you must configure the following setting:

- **X.509 Certificate**: Click this button to select the certificate that API Gateway uses to authenticate itself to clients during the SSL handshake. The list of certificates currently stored in the API Gateway certificate store is displayed. Select a single certificate from this list. For more details, see *Manage X.509 certificates and keys* on page 221.

Mutual Authentication settings

- **Client Certificates**: Define how clients can authenticate to API Gateway on the **Mutual Authentication** tab. Choose from the following:
  - **Ignore Client Certificates**: API Gateway ignores client certificates if they are presented during the SSL handshake.
  - **Accept Client Certificates**: API Gateway accepts client certificates when presented, but clients that do not present certificates are not rejected.
  - **Require Client Certificates**: API Gateway only accepts connections from clients that present a certificate during the SSL handshake.
- **Maximum depth of client certificate chain**: Specify how many CA certificates in a chain of one or more are trusted when validating the client certificate. By default, only one issuing CA certificate is used, and this certificate must be checked in the list of trusted root certificates. If more than one certificate is used, only the top-level CA must be considered trusted, while the intermediate CA certificates are not.

  Client certificates are typically issued by a Certificate Authority (CA). In most cases, the CA includes a copy of its certificate in the client certificate so that consumers of the certificate can decide whether or not to trust the client based on the issuer of the certificate.
  A *chain* of CAs can also issue the client certificate. For example, a top-level organization-wide CA (for example, Company CA) may have issued department-wide CAs (for example, Sales CA, QA CA, and so on), and each department CA is then responsible for issuing all department members with a client certificate. In such cases, the client certificate may contain a chain of one or more CA certificates.
- **Root Certificate Authorities trusted for mutual authentication**: Select the root CA certificates that API Gateway considers trusted when authenticating clients during the SSL handshake. Only certificates signed by the CAs selected here are accepted.

Configure Advanced SSL settings

You can configure the followingon the **Advanced (SSL)** tab:

- **Check that the SSL certificate's Subject CN resolves to network address**: When this setting is selected, API Gateway attempts to resolve the SSL certificate's `Subject` Common Name (CN) to the network address configuring the SSL interface. If API Gateway cannot resolve

the `Subject` CN to the network address, it logs a warning in the error traces. This setting is selected by default. To disable checking the certificate's `Subject` CN, deselect this setting.

- **SSL Server Name Identifier (SNI)**: Specify the host names that clients request in the **SSL Server Name Identifier (SNI)** table. SNI is an optional TLS feature where the client indicates to the server the host name used to resolve the server address. This enables a server to present different certificates for clients to ensure the correct site is contacted.

  For example, the server IP address is `192.168.0.1`. Clients consult the DNS to resolve a host name to an address and contact the server IP address using TCP/IP. If both `www.acme.com` and `www.anvils.com` resolve to `192.168.0.1`, without SNI, the server does not know which host name the client uses to resolve the address, because it is not party to the client DNS name resolution. The server may certify itself as either service, but when the connection is established, it does not know which host name the client connects to.
  With SNI, the client provides the name of the host (for example, `www.anvils.com`) in the initial SSL exchange, before the server presents its certificate in its distinguished name (for example, `CN=www.anvils.com`). This way, the server can certify itself correctly as providing a service for the client's requested host name.
  To specify an SNI, click **Add**, specify the server host name in **Client requests server name**, click **Server assumes identity** to import a Certificate Authority certificate into the Certificate Store, and click **OK**.

- **Ciphers**: Specify the ciphers that the server supports in the **Ciphers** field. The server selects the highest-strength cipher that is also supported by the client from this list as part of the SSL handshake. The default cipher string of `FIPS:!SSLv3:!aNULL` performs the following:

  - Enables FIPS-compatible cipher suites only
  - Explicitly blocks cipher suites that require SSLv3 or lower
  - Forces the use of TLSv1.2 only
  - Forbids unauthenticated cipher suites

    For more information on the syntax of this setting, see the OpenSSL documentation.

- **SSL session cache size**: Specify the number of idle SSL sessions that can be kept in memory. The default is `32`. If there are more simultaneous SSL sessions, new SSL connections can still be established, but no more SSL sessions are cached. If you set cache size to `0`, there is no cache. No outbound SSL connections are cached.

  At `DEBUG` level or higher, API Gateway logs a trace when an entry goes into the cache, for example:

  ```
  DEBUG   09:09:12:953 [0d50] cache SSL session 11AA3894 to
  support.acme.com:443
  ```

  API Gateway also logs a trace when the cache is full, for example:

  ```
  DEBUG   09:09:12:953 [0d50] enough cached SSL sessions 11AA3894 to
  support.acme.com:443 already
  ```

Tip You can use this setting to improve performance because API Gateway caches the slowest part of establishing the SSL connection. A new connection does not need to go through full authentication if it finds its target in the cache.

- **Ephemeral DH key parameters**: Specify the parameters used to generate Diffie Hellman (DH) keys. The DH key agreement algorithm is used to negotiate a shared secret between two SSL peers. This enables two parties without prior knowledge of each other to jointly establish a shared secret key over an insecure communication channel.

  When DH key parameters are not specified, the SSL client uses the public RSA key in the server's certificate to encrypt data sent to the SSL server and establish a shared secret with the server. However, if the RSA key is ever discovered, any previously recorded encrypted conversations can be decrypted. DH key agreement offers Perfect Forward Secrecy (PFS) because there is no such key to be compromised.
  There are two options when setting DH key parameters:

  - Enter a number (for example, `512`), and the server automatically generates DH parameters with a prime number of the correct size.
  - Paste the Base64 encoding of an existing serialized DH parameters file. You can use standard DH parameters based on known good prime numbers. OpenSSL ships with the `dh512.pem` and `dh1024.pem` files. For example, you can set the DH parameters to the following Base64-encoded string in `pdh512.pem`:

    ```
    -----BEGIN DH PARAMETERS-----

    MEYCQQD1Kv884bEpQBgRjXyEpwpy1obEAxnIByl6ypUM2Zafq9AKUJsCRtMIPWakXU
    GfnHy9iUsiGSa6q6Jew1X
    pKgVfAgEC
    -----END DH PARAMETERS-----
    ```

  The DH parameters setting is required if the server is using a DSA-keyed certificate, but also has an effect when using RSA-based certificates. DH (or similar) key agreement is required for DSA-based certificates because DSA keys cannot be trivially used to encrypt data like RSA keys can.

  Note The EDH key is always used once only to guarantee forward secrecy. This ensures that if the key is compromised, previous keys is not compromised.

- **SSL Protocol Options**: You can configure the following SSL protocol options:

Option	Description
Do not use the SSL v2 protocol	SSL v2 is not used for incoming connections to avoid any weaknesses in this protocol. This is selected by default.
Do not use the SSL v3 protocol	SSL v3 is not used for incoming connections to avoid any weaknesses in this protocol. This is selected by default.

Option	Description
Do not use the TLS v1 protocol	TLS v1.0 is not used for incoming connections to avoid any weaknesses in this protocol. This is selected by default.
Do not use the TLS v1.1 protocol	TLS v1.1 is not used for incoming connections to avoid any weaknesses in this protocol. This is selected by default.
Do not use the TLS v1.2 protocol	TLS v1.2 is not use for incoming connections to avoid any weaknesses in this protocol. This is *not* selected by default.
Prefer local cipher preferences over client's proposal	When choosing a cipher during the SSL/TLS handshake, the client's preferences are selected by default from the list of ciphers supported by the client and the server. When this option is selected, the server's preferences are used instead. This option is *not* selected by default. For more details on ciphers, see the OpenSSL documentation.

Configure conditions for an HTTP Interface

You can configure API Gateway to bring down an active HTTP interface if certain *conditions* fail to hold. For example, API Gateway can bring a HTTP interface if a remote host is not available, or if a physical network interface on the machine running API Gateway loses network connection. For more details, see *Configure conditions for HTTP interfaces* on page 288.

Management services

The Management Services group exposes a number of services that the Admin Node Manager and API Gateway Analytics use for remote configuration and monitoring. The Management Services interfaces and policies are displayed in the Policy Studio tree:

- The **Management Services** policy container under **Policies**
- The **Management Services** HTTP interfaces under **Environment Configuration > Listeners > Admin Node Manager**

Note Admin Node Manager may not function correctly if you change the HTTP interfaces, relative path, servlet applications, or static content provider exposed under the Management Services group. Because of this, the Management Services group should only be modified under strict supervision from Axway Support.

Management services group

By default, the Management Services group consists of the following:

- **HTTP Interface**: The default port where Admin Node Manager exposes all its management services so that they can be configured remotely is port `8090`. At startup, Policy Studio can connect to this port to read and write API Gateway configuration data. By default, API Gateway Analytics exposes all its management services on port `8040`. For more details, see *Change the management services port* on page 284.
- **Relative Path**: The relative path `/` is mapped to a default management policy called **Protect Management Interface**, which is available in the **Management Services** policy container. The policy performs HTTP Basic authentication and passes control to a **Call Internal Service** filter, a special filter that passes messages to a servlet application or static content provider based on the path on which the request was received.

Request processing cycle

For example, with the default configuration, a request is received on `http://localhost:8090/api`. The following steps summarize the request processing cycle:

1. The relative path `/` matches all incoming requests, and requests are dispatched to whatever policy the relative path is mapped to, in this case, the **Protect Management Interface** policy.
2. The **Protect Management Interface** policy authenticates the originator of the request using HTTP Basic authentication. Authentication is necessary because all configuration operations are considered privileged operations and only be carried out by those with the required authority. If the originator is successfully authenticated, the policy invokes the **Call Internal Service** filter.
3. The **Call Internal Service** filter attempts to match the relative path that the request was received on against all the servlets and content providers configured in the same services group as this interface, because the message is received on the management interface (port `8090`).
4. The **Call Internal Service** filter finds `/api/`servlet matching the path the request was received on included in the servlets and content providers configured for the Management Services group, and invokes the servlet.

Change the management services port

By default, Admin Node Manager uses port `8090` as the default management services port. To specify a different port, perform the following steps:

1. In the Policy Studio tree, click **Environment Configuration > Listeners > Node Manager > Management Services > Ports**.
2. In the **Interfaces** pane, right-click the **Management HTTPS interface**, and select **Edit**.
3. In **Port**, enter the port you want to use (for example, `8091`), and click **OK**.

4. Deploy the change to API Gateway.
5. Restart Policy Studio. You must always restart Policy Studio when Management Services are updated.
6. Use the updated port number in the URL to reconnect Policy Studio (for example, `https://HOST:8091/api`).

Customize HTTP security headers

You edit the Management Services group the Admin Node Manager uses to customize the HTTP security headers included in the API Gateway response on port `8090`. You can edit the Admin Node Manager configuration using either Policy Studio or Entity Explorer.

Note Management Services apply to the Admin Node Manager and API Gateway Analytics only. Any modifications must be done under strict advice and supervision from the Axway Support.

Edit Admin Node Manager configuration in Policy Studio

To change the Admin Node Manager configuration, create a copy of your existing configuration where to do the edits. When ready, copy the required configuration files back to your actual API Gateway configuration.

1. In Policy Studio, click **File > New Project**.
2. Enter a project **Name**, and click **Next**.
3. Select to start the new project **From existing configuration**, and click **Next**.
4. Click the browse button, select the following directory, and click **Finish**:

```
INSTALL_DIR/apigateway/conf/fed
```

5. In the Policy Studio tree, click **Environment Configuration > Listeners > Node Manager > Management Services** and open **Paths**.
6. In the **Resolvers** pane, right-click the **/** static content node, and click **Edit**.
7. On the **General** tab, click **Add**, enter the HTTP security header name/value pair, and click **OK**. For example:
   - **HTTP Header**: `X-XSS-Protection`
   - **Value**: `1; mode=block`
8. Repeat to add any additional HTTP security header name/value pairs (for example, `Strict-Transport-Security` or `Public-Key-Pins`), then click **OK** to return to Resolvers pane.
9. Under **Paths**, right-click the **Login** static content node, and click **Edit**.
10. On the **General**, click **Add**, and add the HTTP security headers that you added to the **/** static content node.

11. After you are finished editing the configuration, manually copy the `.xml` and `.md5` configuration files from the new project you edited to `INSTALL_DIR/conf/fed/`, and make a backup.
12. Restart the Admin Node Manager.

Edit Admin Node Manager configuration in Entity Explorer

Perform the following steps:

1. Go to the following directory:

```
INSTALL_DIR/apigateway/posix/bin
```

2. Run `esexplorer`.
3. Right-click on the store you want, select **Connect**, and browse to the following file:

```
INSTALL_DIR/apigateway/conf/fed/configs.xml
```

4. Click **System Components > Service > Management Services**, and expand **/,*** static content node.
5. Right-click the **/,*** static content node, select **Add a new Property**, and click the new property (for example, `key-0`).
6. In the `name` row on the right, double-click the **Value** field, and enter the name of the HTTP security header (for example, `X-XSS-Protection`).
7. Right-click under this field, select **Create a value** to add a `value` row, double-click **Value** on the row, and enter the value of the HTTP security header (for example, `1; mode=block`).
8. Repeat to add any additional HTTP security header name/value pairs (for example, `Strict-Transport-Security` or `Public-Key-Pins`).
9. Click **Update** to save the changes.
10. Select **System Components > Service > Management Services**, and expand the **/login/,*** static content node,
11. Right-click **/login/,***, select **Add a new Property**, and click the new property.
12. Add the HTTP security headers that you added under the **/,*** static content node in the same way.
13. Click **Update** to save the changes.
14. Restart the Admin Node Manager.

Packet sniffers

Packet Sniffers are a type of Passive Service. Rather than opening up a TCP port and *actively*

listening for requests, the Packet Sniffer *passively* reads raw data packets off the network interface. The Sniffer assembles these packets into complete messages that can then be passed into an associated policy.

Because the Packet Sniffer operates passively (does not listen on a TCP port) and, therefore, completely transparently to the client, it is most useful for monitoring and managing web services. For example, the Sniffer can be deployed on a machine running a web server acting as a container for web services.

Assuming that the web server is listening on TCP port 80 for traffic, the Packet Sniffer can be configured to read all packets destined for port 80 (or any other port, if necessary). The packets can then be marshalled into complete HTTP/SOAP messages by the Sniffer and passed into a policy that logs the message to a database, for example.

Note On Linux platforms, API Gateway must be started by the root user to gain access to the raw packets.

Configuration

Since Packet Sniffers are mainly for use as passive monitoring agents, they are usually created within their own HTTP Service Group. For example, you can create a new Service Group for this purpose by right-clicking on the API Gateway instance, selecting the **Add HTTP Services** menu option, and entering `Packet Sniffer Group` on the **HTTP Services** dialog.

You can then add a relative path service to this Group by right-clicking the **Packet Sniffer Group**, and selecting the **Add Relative Path** menu option. Enter a path in the field provided, and select the policy that you want to dispatch messages to when the Packet Sniffer detects a request for this path (after it assembles the packets). For example, if the relative path is configured as `/a`, and the Packet Sniffer assembles packets into a request for this path, the request is dispatched to the policy selected in the relative path service.

Finally, you can add the Packet Sniffer by right-clicking the **Packet Sniffer Group** node, selecting **Packet Sniffer**, and then the **Add** menu option. Complete the following fields on the **Packet Sniffer** dialog:

Device to Monitor:
Enter the name or identifier of the network interface that the Packet Sniffer is to monitor. The default entry is `any`.

Note This setting is only valid on Linux. On Linux-based systems, network interfaces are usually identified using names like `eth0`, `eth1`, and so on. On Windows, these names are more complicated (for example, `\Device\NPF_{00B756E0-518A-4144 ... }`.

Filter:
The Packet Sniffer can be configured to only intercept certain types of packets. For example, it can ignore all UDP packets, only intercept packets destined for port 80 on the network interface, ignore packets from a certain IP address, listen for all packets on the network, and so on.

The Packet Sniffer uses the libpcap library filter language to achieve this. This language has a complicated but powerful syntax that allows you to *filter* what packets are intercepted and what packets are ignored. As a general rule, the syntax consists of one or more expressions combined with conjunctions, such as `and`, `or`, and `not`. The following table lists a few examples of common filters and explains what they filter:

Filter expresssion	Description
`port 80`	Capture only traffic for the HTTP Port (`80`).
`host 192.168.0.1`	Capture traffic to and from IP address `192.168.0.1`.
`tcp`	Capture only TCP traffic.
`host 192.168.0.1 and port 80`	Capture traffic to and from port 80 on IP address `192.168.0.1`.
`tcp portrange 8080-8090`	Capture all TCP traffic destined for ports from 8080 through to 8090.
`tcp port 8080 and not src host 192.168.0.1`	Capture all TCP traffic destined for port 8080 but not from IP address `192.168.0.1`.

The default filter of `tcp` captures all TCP packets arriving on the network interface. For more information on how to configure filter expressions like these, refer to the `tcpdump` man page available from http://www.tcpdump.org/tcpdump_man.html.

Promiscuous Mode:
When listening in promiscuous mode, the Packet Sniffer captures all packets on the same Ethernet network, regardless of whether or not the packets are addressed to the network interface that the Sniffer is monitoring.

Configure conditions for HTTP interfaces

Overview

In certain cases, it may be desirable to pull down the HTTP interface that accepts traffic for the API Gateway. For example, if the back-end web service is unavailable or if the physical interface on the machine loses connectivity to the network, it is possible to shut down the HTTP interface so that it stops accepting requests.

A typical scenario where this functionality proves useful is as follows:

- A load balancer sits in front of several running instances of the API Gateway and round-robins requests between them all.
- A client sends SSL requests through the load balancer, which forwards them opaquely to one of the API Gateway instances.
- The API Gateway terminates the SSL connection, processes the message with the configured policy, and forwards the request onto the back-end web service.

In this deployment scenario, the load balancer does not want to keep sending requests to an instance of the API Gateway if it has either lost connectivity to the network or if the back-end web service is unavailable. If either of these *conditions* hold, the load balancer should stop attempting to route requests through this instance of the API Gateway and use the other instances instead.

So then, how can the load balancer determine the availability of the web service and also the connectivity of the machine hosting the API Gateway to the network on which the web service resides? Given that the request from the client to the API Gateway is over SSL, the load balancer has no way of decrypting the encrypted SSL data to determine whether or not a SOAP Fault, for example, has been returned from the API Gateway to indicate a connection failure.

The solution is to configure certain *conditions* for each HTTP interface, which must hold for the HTTP interface to remain available and accept requests. If any of the associated conditions fail, the interface is brought down and does not accept any more requests until the failed condition becomes true and the HTTP interface is restarted.

When the load balancer receives a connection failure from the API Gateway (which it does when the HTTP interface is down) it stops sending requests to this API Gateway and chooses to round-robin requests amongst the other instances instead.

The following conditions can be configured on the HTTP interface:

- Requires Endpoint:
  The HTTP interface remains up only if the remote host is available. The remote host is polled periodically to determine availability so that the HTTP interface can be brought back up automatically when the Remote Host becomes available again.
- Requires Link:
  The HTTP interface remains up only if a named physical interface has connectivity to the network. As soon as a down physical interface regains connectivity, the HTTP interface automatically comes back up again.

You can configure conditions for an HTTP interface using the HTTP interface **Ports** node (for example, `*:8080`) under the API Gateway instance in the Policy Studio tree. For example, this is available under **Environment Configuration** > **Listeners** > **API Gateway**.

Right click the HTTP interface in the right pane, and select **Add Condition** menu option, and then either the **Requires Endpoint** or **Requires Link** option depending on your requirements. The sections below describe how to configure these conditions.

Configure Requires Endpoint condition

A **Requires Endpoint** condition can be configured in cases where you only want to keep the HTTP interface up if the back-end web service (the Remote Host) is available. An HTTP Watchdog can be configured for the Remote Host, which is then responsible for polling the Remote Host periodically to ensure that the web service is available. See *Configure remote host settings* on page 268 and *Configure HTTP watchdog* on page 273 for more information.

Remote Host:
The HTTP interface shuts down if the Remote Host selected here is deemed to be unavailable. The Remote Host can be continuously polled so that the interface can be brought up again when the Remote Host becomes available again.

Configure Requires Link condition

The **Requires Link** condition is used to bring down the HTTP interface if a named physical network interface is no longer connected to the network. For example, if the cable is removed from the Ethernet switch, the dependent HTTP interface is brought down immediately. The HTTP interface only starts listening again when the physical interface is connected to the network again (when the Ethernet cable is plugged back in).

Note The **Requires Link** condition is only available on Linux platforms.

Interface Name:
The HTTP interface is brought down if the physical network interface named here is no longer connected to the network. On UNIX platforms, physical network interfaces are usually named `eth0`, `eth1`, and so on.

Configure a transparent proxy

Overview

On Linux systems with the `TPROXY` kernel option enabled, you can configure the API Gateway as a *transparent proxy*. This enables the API Gateway to present itself as having the server's IP address from the point of view of the client, or having the client's IP address from the point of view of the server. This can be useful for administrative or network infrastructure purposes (for example, to keep using existing client/server IP addresses, and for load-balancing).

You can configure transparent proxy mode both for inbound and outbound API Gateway connections:

- Incoming interfaces can listen on IP addresses that are not assigned to any interface on the local host.
- Outbound calls can present the originating client's IP address to the destination server.

Both of these options act independently of each other.

Configure transparent proxy mode for incoming interfaces

To enable transparent proxy mode on an incoming interface, perform the following steps:

1. In the Policy Studio tree, expand the **Environment Configuration** > **Listeners** > **API Gateway** node.
2. Right-click your service, and select **Add Interface** > **HTTP** or **HTTPS** to display the appropriate dialog (for example, **Configure HTTP Interface**).
3. Select the check box labeled **Transparent Proxy (allow bind to foreign address)**. When selected, the value in the **Address** field can specify any IP address, and incoming traffic for the configured address/port combinations is handled by the API Gateway.

For more details on configuring interfaces, see *Configure HTTP services* on page 275.

Configure transparent proxy mode for outgoing calls

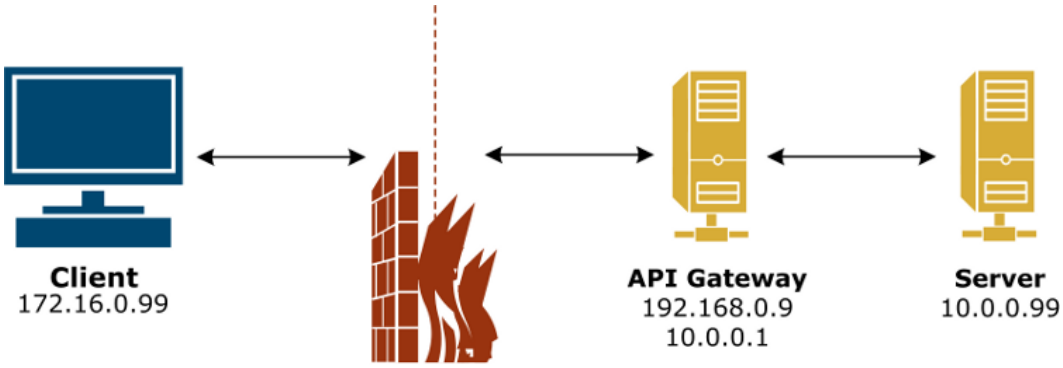
Transparent proxy mode for outgoing calls must be enabled at the level of a connection filter in a policy. To enable transparent proxy mode for outbound calls, perform the following steps:

1. Ensure that your policy contains a connection filter (for example, **Connect to URL** or **Connection**, available from the **Routing** category in the filter palette).
2. In your connection filter, select the **Settings** tab and expand the **Proxy** section.
3. Select the check box labeled **Transparent Proxy (present client's IP address to server)**. When selected, the IP address of the original client connection that caused the policy to be invoked is used as the local address of the TCP connection to the destination server.

For more details on configuring connection filters, see "Connection" in the *API Gateway Policy Developer Filter Reference* and "Connect to URL" in the *API Gateway Policy Developer Filter Reference*.

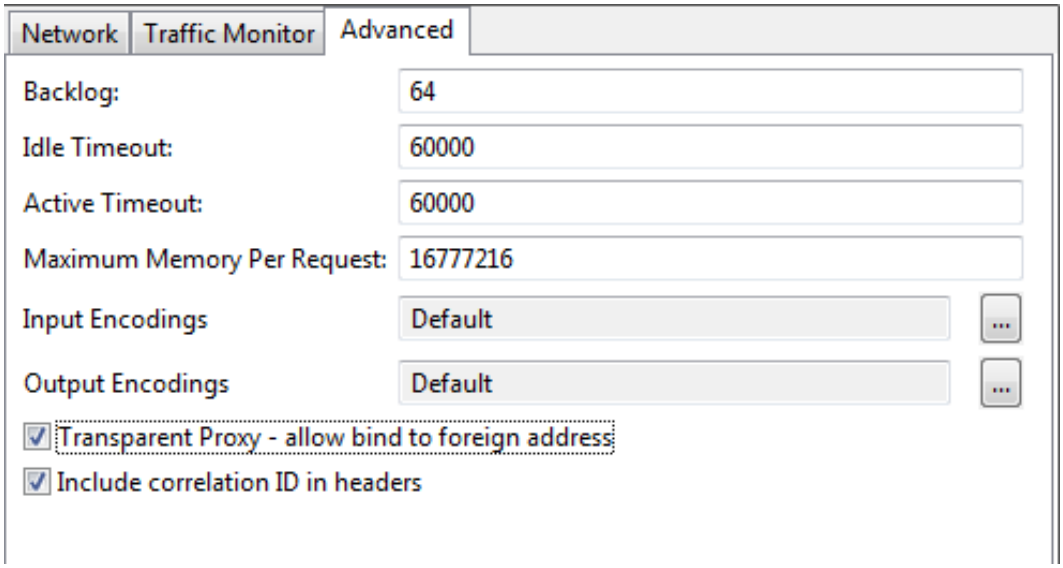
Transparent proxy example

A typical configuration example of transparent proxy mode is shown as follows:



In this example, the remote client's address is `172.16.0.99`, and it is attempting to connect to the server at `10.0.0.99` on port `80`. The front-facing firewall is configured to route traffic for `10.0.0.99` through the API Gateway at address `192.168.0.9`. The server is configured to use the API Gateway at address `10.0.0.1` as its default IP router.

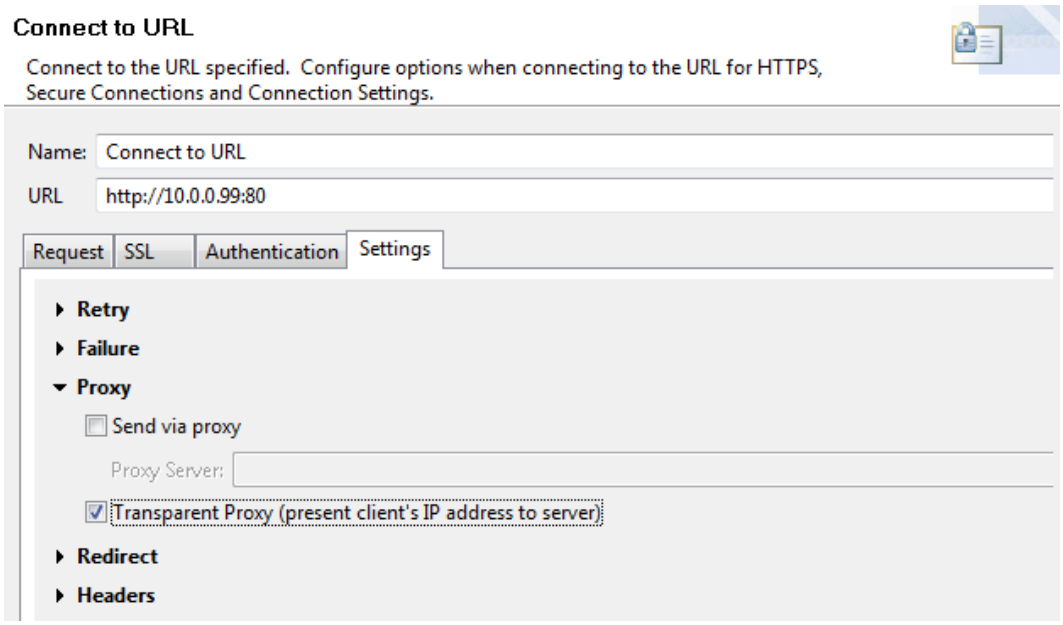
The API Gateway is multihomed, and sits on both the `192.168.0.0/24` and `10.0.0.0/24` networks. In the Configure HTTP Interface dialog, the API Gateway is configured with a listening address of `10.0.0.99` and port of `80` on the **Network** tab, and with transparent proxy mode enabled on the **Advanced** tab. For example:



The API Gateway accepts the incoming call from the client, and processes it locally. However, there is no communication with the server yet. The API Gateway can process the call to completion and respond to the client—it is masquerading as the server.

If the API Gateway invokes a connection filter when processing this call (with transparent proxying enabled), the connection filter consults the originating address of the client, and binds the local address of the new outbound connection to that address before connecting. The server then sees the incoming call on the API Gateway originating from the client (`172.16.0.99`), rather than either of the API Gateway's IP addresses.

The following dialog shows the example configuration for the **Connect to URL** filter:



The result is a transparent proxy, where the client sees itself as connecting directly to the server, and the server sees an incoming call directly from the client. The API Gateway processes two separate TCP connections, one to the client, one to the server, with both masquerading as the other on each connection.

Note Either side of the transparent proxy is optional. By configuring the appropriate settings for the incoming interface or the connection filter, you can masquerade only to the server, or only to the client.

Configure relative paths

A *relative path* binds policies to a specific relative path location (for example `/test/path`). When the API Gateway receives a request on the specified path, it invokes the specified policy or policy chain. This topic explains configuring relative path resolvers. For details on configuring policies, see *Manage policies* on page 65.

You can use a *Static Content Provider* or *Static File Provider* to serve static HTTP content or files from a directory. In this case, the API Gateway instance acts as a web server. The API Gateway instance can also act as a *Servlet Application* container, which enables you to drop servlets into the HTTP Services configuration. This should only be used by developers with specific requirements under strict advice from Axway Support.

Relative paths can have nested child resolvers of the following type:

- Relative path
- Static content provider
- Static file provider
- Servlet application

This topic explains how to use the Policy Studio to configure each of these relative path resolver types. For details on configuring HTTP Services Groups and HTTP Interfaces, see *Configure HTTP services* on page 275.

Configure a relative path

To configure a relative path for a specific HTTP Service Group (for example, **Default Services**), perform the following steps:

1. In the Policy Studio tree, select **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** > **Relative Path**. You can also click **Add** on the right.
3. In the **Configure Relative Path** dialog, specify whether to enable listening on the specified path using the **Enable this path resolver** setting, which is set by default.

Alternatively, when editing a policy, you can click **Add Relative Path** at the bottom of the policy canvas beside the **Context** drop-down list. The next sections explain how to configure the settings on the **Configure Relative Path** dialog.

Policies settings

Use the **Policies** tab to specify the relative path and the policies that are called. The API Gateway invokes the selected policies when it receives a request on the specified path. You can specify a single policy or a chain of policies. Policies are called in the order displayed on this tab. Complete the following fields:

When a request arrives that matches the path:
Enter a relative path (for example, `/test/path`) for the selected HTTP Services Group. Requests received on this relative path are processed by the policies selected on this tab.

Global Request Policy:
If a global request policy is configured, when you select this setting, the global request policy is called first in the policy chain. For more details, see *Configure global policies* on page 67.

Path Specific Policy:
To configure a path-specific policy, select this setting, and browse to select a policy from the dialog. You can search for a specific policy by entering its name in the text box, and the policy tree is filtered automatically. The **Path Specific Policy** field is auto-populated with the currently selected policy when the dialog is launched using the **Add Relative Path** button at the bottom of the policy canvas.

Global Response Policy:
If a global response policy is configured, when you select this setting, the global response policy is called last in the policy chain. For more details, see *Configure global policies* on page 67.

When you select multiple policies to form a policy chain, the behavior is the same as for a policy shortcut chain filter. Policies are only evaluated when selected, and when the policy can be reached. If any reachable policy fails, the chain fails, and no more policies are evaluated.

Logging settings

The **Logging Settings** tab enables you to configure the logging level for all filters executed on the relative path, and to configure when message payloads are logged.

Logging Level

You can configure the following settings on all filters executed on the specified relative path:

Logging Level	Description
Fatal	Logs Fatal log points that occur on all filters executed.
Failure	Logs Failure log points that occur on all filters executed. This is the default logging level.
Success	Logs Success log points that occur on all filters executed.

For details on logging levels, and configuring logging for a filter, see "Set transaction log level and log message" in the *API Gateway Policy Developer Filter Reference*.

Payload Level

You can configure the following settings on the specified relative path:

Payload Logging	Description
On receive request from client	Log the message payload when a request arrives from the client.
On send response to client	Log the message payload before the response is sent back to the client.
On send request to remote server	Log the message payload before the request is sent using any **Connection** or **Connect to URL** filters deployed in policies.
On receive response from remote server	Log the message payload when the response is received using any **Connection** or **Connect to URL** filters deployed in policies.

For details on how to log message payloads at any point in a specific policy, see "Log message payload" in the *API Gateway Policy Developer Filter Reference*.

Access Log

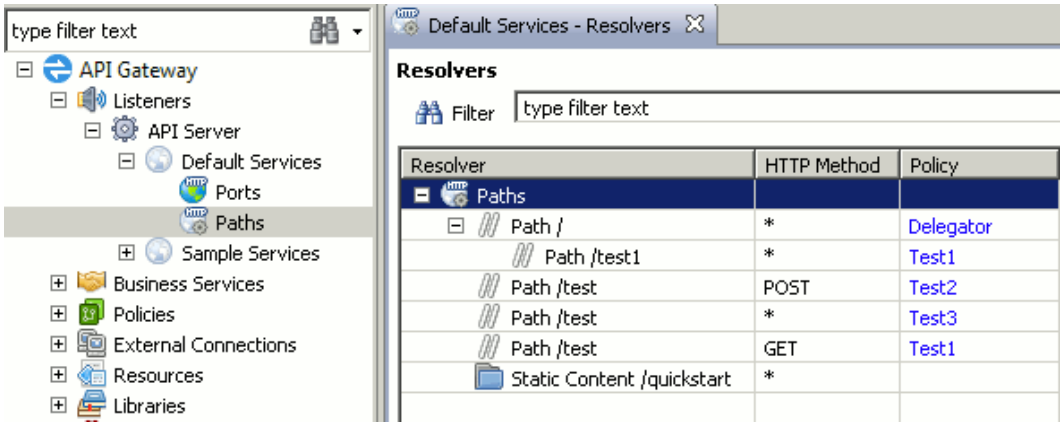
Select the **Include in server access log records** setting to add this relative path to the API Gateway Access Log. This enables the Access Log at the service level. This setting is not selected by default.

Note You must also enable the Transaction Access Log at the API Gateway level. In the Policy Studio tree, select **Environment Configuration** > **Server Settings** > **Logging** > **Access Log**, and ensure that **Writing to Transaction Access Log** is enabled. For more details, see the *API Gateway Administrator Guide*.

HTTP method settings

The **HTTP Method** tab enables you to configure an accepted HTTP Method (for example, `POST`). The default is `*`, which means that all HTTP Methods are accepted. You can override the default behavior, and select an appropriate HTTP method for the relative path from the list.

You can also configure multiple HTTP methods on paths of the same name. This enables you to call different policies for different HTTP methods, as shown in the following example:



In this example, the `/test` path is configured three times, each using a different HTTP Method as follows:

- If a `GET` request is sent to the API Gateway on the `/test` path, the `Test1` policy is executed.
- If a `POST` request is sent, the `Test2` policy is executed.
- If any other type of request is sent (for example, `DELETE`, `PUT`, and so on), the `Test3` policy is executed.

For details on the `Path/Test1` subpath in this example, see *Nested relative paths* on page 297.

Advanced settings

On the **Advanced** tab, select whether to resolve this relative path using an **Exact path match**. The default is to use a longest path match (explained in the next section). This setting enables you to further restrict the match to an exact path match (for example, `test1/helloService`). This setting is not selected by default.

CORS settings

On the **CORS** tab, you can configure settings for Cross Origin Resource Sharing (CORS). For details on CORS, see *Cross-Origin Resource Sharing* on page 253. By default, the CORS profile set at the HTTP service level is used for all child resolvers of the HTTP service. However, you can override this at the relative path level as follows:

1. In the **CORS Usage** field, select **Override CORS using the following profile**. By default, no CORS profile is selected, and the parent settings are used.

   Note Relative paths can act as HTTP Services, and can accommodate child resolvers. This means that when a Relative Path has children, and has a CORS profile configured, by default, the children use the parent profile (unless a child overrides it).

2. In the **CORS Profile** field, click the button on the right to select a preconfigured CORS Profile.
3. If no CORS Profiles have already been configured, right-click **CORS Profiles**, and select **Add a CORS Profile**. You can also right-click an existing profile, and select **Edit** to update its settings. For details on CORS settings, see *Add a CORS profile* on page 255.

Nested relative paths

Using the example shown in *HTTP method settings* on page 296, when you have a path `/` that has a child subpath (in this case `/test1`), the following occurs when a request arrives at the API Gateway:

1. Incoming request with path `/test1` is received
2. Request is resolved against `/` (using the longest path match algorithm)
3. API Gateway checks if there are any children (in this case `/test1`)
4. API Gateway checks if any children can process the request (`/test1` is a match)
5. Path resolution is successful

When the request is being processed, each policy associated with a matching path is executed. In the above example, both `/` and `/test1` make up the match. This means that the policy associated with `/` is executed first, and if that passes, the policy associated with `/test1` is executed. The parent policy uses the **Call Internal Service** filter, which enables child resolvers to be invoked.

Nested paths are generally used as a protection mechanism. For example, a system might be configured with `/` as the only root path, with a number of children. `/` could have a Role-Based Access Control (RBAC) policy associated with it protecting all children. If the RBAC policy succeeds, access is granted, and the child policy is executed. If it fails, access is denied. This mechanism is implemented in the Admin Node Manager. You can view this in Policy Studio by opening the Admin Node Manager configuration.

Note The difference between `/` having a child `/test1`, and just having a root relative path of `/test1` is due to path resolution and policy execution. To protect `/test1` using RBAC, or run a prerequisite policy prior to running the policy associated with `/test1`, you should use subpaths.

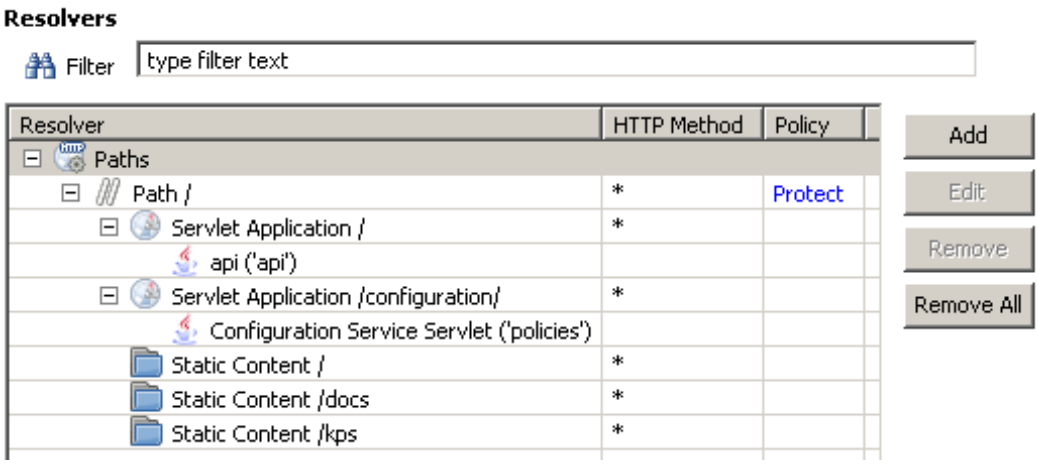
You can also achieve this using a **Policy Shortcut** filter in the policy associated with `/test1` (as a root path). This may be sufficient for a small number of root policies. But when a large number of policies need protection, subpaths are a more elegant solution. Children no longer need a **Policy Shortcut** filter with the policy associated with the parent path as protector.

Add a nested relative path

To add a child node to a relative path, right-click the appropriate relative path in the **Resolvers** window, and select the node type (for example, **Add** > **Relative Path**). You can add child nodes of the following type:

- Relative path
- Static content provider
- Servlet application

In the following example, the main relative path is `/`, which calls the **Protect** policy. Content is served by the underlying Servlet Application and Static Content resolvers only when the **Protect** policy succeeds:



Note The parent policy (in this case, **Protect**) must use the **Call Internal Service** filter. This acts as a loopback and enables child resolvers to be invoked. When this prerequisite is met, you can add nested relative paths as required.

How to access message attributes from parent resolvers

Because invoking the **Call Internal Service** policy results in a new transaction being created (in a loopback connection), a new message whiteboard is generated. This means that you cannot access the message attributes from the parent resolver policy directly. However, in the API Gateway trace, the `dwe.protocol.loopback.message` message attribute is traced in the child policy. For example:

```
dwe.protocol.loopback.message {
  Value:com.vordel.dwe.http.HTTPMessage@df4117
  Type:com.vordel.dwe.http.HTTPMessage
}
```

This is the message object from the parent policy. For example, given a parent resolver policy that generates an `attr1` message attribute, you can use the following selector in the child resolver policy to access `attr1`:

```
${dwe.protocol.loopback.message.attr1}
```

Order of resolution

The order of resolution for nested relative paths is to first resolve at the parent level. If resolution is successful, and there are children present, then attempt to resolve at the child level. There is no precedence between resolver node types (relative path, static content provider, and servlet application).

Path resolution is performed using the longest path match algorithm by default, regardless of whether nested subpaths are used. For details on exact path match, see *Advanced settings* on page 297.

Example nested path resolution

Using the example relative path shown in *Add a nested relative path* on page 298, consider the following inbound requests to the Node Manager:

```
https://testpc:8090/api/deployment/domain/deployments
https://testpc:8090/common/themes/blue/images/server_icon.png
```

The `/api/deployment/domain/deployments` request is resolved as follows:

- Matches the root relative path `/` as a longest path match
- Matches Servlet Application `/` as a longest path match (1 char)
- Matches Servlet `/api` as a longest path match (4 char)
- Matches Static Content `/` as a longest path match (1 char)
- Does not match Servlet Application `/configuration`
- Does not match Static Content `/docs`
- Does not match Static Content `/kps`

In this example, there are two matches, the `api` Servlet under the Servlet Application on `/`, and the Static Content on `/`. Because the API Gateway uses the longest path match, the `api` Servlet wins, and the request is routed to that resolver. There is no precedence between resolvers, all resolvers are queried for a match.

The `/common/themes/blue/images/server_icon.png` request is resolved as follows:

- Matches the root relative path `/` as a longest path match
- Matches servlet application `/` as a longest path match (1 char)
- Does not match servlet `/api`
- Matches static content `/` as a longest path match (1 char)
- Does not match servlet application `/configuration`
- Does not match static content `/docs`
- Does not match static content `/kps`

In this example, there is only one match, the Static Content on `/`. In this case, the Servlet Application on `/` is not considered because none of its children can resolve the request path.

Note If there are two resolver matches, and each matches on the same path length, the last resolver visited during path resolution is used, in the order in which the resolver was read and loaded from the configuration.

Static content providers

A *Static Content Provider* can be used with an HTTP Interface to serve static content from a specified directory. A relative path is associated with each Static Content Provider so that requests received on this path are dispatched directly to the provider and are not mapped to a policy in the usual way. For example, you can configure a Static Content Provider to serve content from the `/opt/mydirectory` folder when it receives requests on the relative path `/mydirectory`.

Add a static content provider

To add a Static Content Provider to an HTTP Services group (for example, **Default Services**):

1. Select **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** > **Static Content Provider**.
3. Complete the following fields on the **General** tab.

Relative Path:
Enter the path that you want to receive requests for static content on.

Content Directory:
Enter or browse to the location of the directory that you want to serve static content from.

Index File:
Enter the name of the file to use as the index file for content retrieved from the directory specified in the field above. This file is retrieved by default if no resource is explicitly specified in the request URI. For example, if the client requests `http://[HOST]:8080/docs` (with only a relative path specified instead of a specific resource), the file specified here is retrieved. This file must exist in the directory specified in the previous field.

Allow Directory Listings:
If this is selected, the Static Content Provider returns full directory listings for requests specifying a relative path only. For example, if selected, and if a request is received for `http://[HOST]:8080/docs/samples`, the list of directories under this directory is returned, assuming that this directory exists on the file system. You can deselect to prevent attacks where a hacker can send up different request URIs in the hope that the server returns some information about the directory structure of the server.

HTTP Method:
The **HTTP Method** tab enables you to configure an accepted HTTP Method (for example, `POST`). The default is `*`, which accepts all HTTP methods. You can override the default behavior, and select an appropriate HTTP method for this resolver from the list. For more details, see *HTTP method settings* on page 296.

Static file providers

A *Static File Provider* can be used with an HTTP Interface to serve a static file from a specified directory. A relative path is associated with each Static File Provider so that requests received on this path are dispatched directly to the file provider and are not mapped to a policy in the usual way. For example, you can configure a Static File Provider to serve `/my_brand/favicon.ico` when it receives requests on the `/favicon.ico` relative path.

Add a static file provider

To add a Static File Provider to an HTTP Services group (for example, **Default Services**):

1. Select **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services** > **Paths**.

2. Right-click **Paths**, and select **Add** > **Static File Provider**.
3. Complete the following fields on the **General** tab.

Relative Path:
Enter the path that you want to receive requests for the static file on (for example, `/favicon.ico`).

File:
Enter or browse to the location of static file you want to serve (for example, `$VDISTDIR/webapps/emc/favicon.ico`, where `$VDISTDIR` specifies the directory where the API Gateway is installed).

HTTP Method:
This tab enables you to configure an accepted HTTP method for the static file. The default is `GET`, which means that only HTTP GET calls are accepted. You can override the default, and select a different HTTP method for this resolver from the list. For more details, see *HTTP method settings* on page 296.

Servlet applications

Developers can write their own Java servlets and deploy them under the API Gateway to serve HTTP traffic. Conversely, they can remove some of the default servlets from the out-of-the-box configuration (for example, to remove the ability to view logging remotely). This pairing down of unwanted functionality may be required to further lock down the machine on which the API Gateway is running.

Note Adding and removing Servlet Applications should be performed only by developers with very specific requirements and under strict guidance from the Axway Support team. These instructions simply outline how to configure the fields on the dialogs used to set up Servlet Applications. For more detailed instructions, contact the Axway Support team.

When editing Admin Node Manager or API Gateway Analyticsconfiguration, there are some default Servlet Applications available under the **Management Services** group.

Caution Deleting any default Servlet Applications may prevent the API Gateway from functioning correctly. You should only delete default Servlet Applications under strict supervision of Axway Support.

Add a servlet application

To add a Servlet Application to an HTTP Services Group (for example, **Default Services**), perform the following steps:

1. Select **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services** > **Paths**.
2. Right-click **Paths**, and select **Add** > **Servlet Application**.
3. Complete the following fields on the **General** tab.

Relative Path:
Enter the servlet *context* in this field. You can add multiple servlets under this context, where each servlet is mapped to a unique URI.

Session Timeout:
Enter the timeout in seconds after which an inactive session is closed. Click **OK**.

HTTP Method:
This tab enables you to configure an accepted HTTP method (for example, `POST`). The default is `*`, which accepts all HTTP methods. You can override the default, and select an appropriate HTTP method for this resolver from the list. For more details, see *HTTP method settings* on page 296.

Add a servlet

The new Servlet Application now appears in the **Resolvers** window. To add a new servlet, right-click the new Servlet Application, and select **Add Servlet**. Configure the following fields on the **Servlet** dialog:

URI:
The path entered maps incoming requests on a particular request URI to the Java servlet class entered in the field below. This path must be unique across all Servlets added under this Servlet Application (servlet context).

Class:
Enter the fully qualified class name of the servlet class. You can add this class to the server runtime by adding the JAR, class file, or package hierarchy to the `[VDISTDIR]/ext/lib` folder. `VDISTDIR` is your API Gateway distribution directory, which is the location where the API Gateway is installed.

Read Timeout:
Specify the time in seconds that the servlet should wait before closing an idle connection.

Servlet Properties:
You can configure properties for each servlet by clicking the **Add** button, and entering a name and value in the fields provided on the **Properties** dialog.

Web service resolvers

A web service resolver is used to identify messages destined for a web service, and to map them to the **Service Handler** (**Web Service Filter**) for that web service. When you import a WSDL file in the **Web Service Repository**, a new web service resolver node is created for each imported web service under the **Paths** for the relevant HTTP services group (for example, Default Services). You can edit the web service resolver settings by right-clicking its tree node, and selecting **Edit**.

The following settings are available in the **Web Service Resolver** dialog:

Enable this Web service resolver:
Specify whether to enable this web service resolver. This is enabled by default.

Name:
You can edit the name of the web service resolver.

Web service:
Click the browse button to select a web service to resolve to. Defaults to the web service imported into the Web Services Repository when this resolver was created.

Policies:
On the **Policies** tab, select the path and the policies to use for the web service. You can specify a single policy or a chain of policies. Policies are called in the order displayed on this tab. The global request policy, the policy automatically generated when the WSDL file is imported, and the global response policy are all selected in a chain by default. Complete the following fields:

- **Matches the paths in the WSDL**:

  Select this option if you want the resolver to use the paths specified in the WSDL file. This is the default.

- **Matches this path**:

  Select this option if you want to specify a different path from the WSDL file, and enter the path.

- **Global Request Policy**:

  If a global request policy is configured, when you select this setting, the global request policy is called first in the policy chain. For more details, see *Configure global policies* on page 67.

- **Path Specific Policy**:

  To configure a path-specific policy, select this setting, and browse to select a policy from the dialog.

- **Global Response Policy**:

  If a global response policy is configured, when you select this setting, the global response policy is called last in the policy chain. For more details, see *Configure global policies* on page 67.

Policies are only evaluated when selected, and when the policy can be reached. If any selected policy fails, the chain fails, and no more policies are evaluated.

Logging Settings:
The **Logging Settings** tab enables you to configure the logging level for all filters executed on the web service, and to configure when message payloads are logged. The default logging level for all filters on the web service is `Failure`. These logging settings are the same as those already described for the relative path. For more details, see *Logging settings* on page 295.

HTTP Method:
The **HTTP Method** tab enables you to configure an accepted HTTP Method (for example, `POST`). The default is `*`, which means that all HTTP Methods are accepted. You can override the default behavior, and select an appropriate HTTP method from the list. The **HTTP Method** settings are the same as those already described for the relative path. For more details, see *HTTP method settings* on page 296.

Edit service handler options

You also edit options for the **Service Handler** for the web service. Right-click the Web Service Resolver node, and select **Quick-Edit Policy** to display a dialog that enables you to configure the following options:

- **Validation**:

  To use a dedicated validation policy for all messages sent to the web service, select this check box, and click the browse button to configure a policy in the dialog. For example, this enables you to delegate to a custom validation policy used by multiple web services.

- **Routing**:

  To use a dedicated routing policy to send all messages on to the web service, select this check box, and click the browse button to configure a policy in the dialog. For example, this enables you to delegate to a custom routing policy used by multiple web services.

- **WSDL Access Options**:

  Select whether to make the WSDL for this web service available to clients. The **Allow the API Gateway to publish WSDL to clients** check box is selected by default. The published WSDL represents a virtualized view of the web service. Clients can retrieve the WSDL from the API Gateway, generate SOAP requests, and send them to the API Gateway, which routes them on to the web service.

These options enable you to configure the underlying autogenerated Service Handler (**Web Service Filter**) without navigating to it in the **Policies** tree. These are the most commonly modified **Web Service Filter** options after importing a WSDL file. Changes made in this dialog are visible in the underlying **Web Service Filter**. For more details, see "Web service filter" in the *API Gateway Policy Developer Filter Reference*.

Check URI path syntax in incoming HTTP requests

You can configure whether to enable strict checking of URI syntax for incoming HTTP requests. If this is enabled, requests with invalid URI path syntax are rejected (the default). If this is disabled, the API Gateway finds the best path match for the URI string.

The main use case for disabling this setting is for compatibility with legacy API Gateway behavior. If you already accept and process client URLs that are not strictly correct, you might want to continue. For example, the Microsoft IIS Web server accepts URLs that contain a backslash character (for example, `http://myserver.com\resource`). You might want to put an API Gateway in front of your IIS server, and continue to accept these non-standard URLs.

To configure this setting, perform the following steps:

1. Edit the following file:

```
INSTALL-DIR/apigateway/system/conf/jvm.xml
```

2. Configure the following setting to `true` or `false` as appropriate:

```
<VMArg name="-Dcom.vordel.strictUriSyntaxChecking=true"/>
```

The default and recommended setting is `true`.

Configure rate limiting

You can configure limits to the rate at which message requests pass through the API Gateway. The **Throttling** filter enables you to limit the number of requests that pass through API Gateway instances over a specified time period. For example, this enables you to enforce a specified message quota or *rate limit* on a client application, and to protect a back-end service from message flooding. This is especially useful under high volume and in elastic deployments, where dynamic topology changes can destabilize back-end servers.

The **Throttling** filter provides the following rate limit algorithms:

- **Smooth Rate Limiting**: This algorithm smooths out the traffic by dividing per second limits into regular millisecond intervals. For example, a setting of 500 requests per second results in 1 request being accepted every 2 milliseconds. It provides most protection to back-end servers, and is especially suitable for back-end server throttling in elastic environments, but can also be used in on-premise deployments.

  The Smooth Rate Limiting algorithm distributes rate limits among running API Gateways evenly (round robin) or dynamically (based on past traffic) to match your load balancing strategy. It also keeps track of the number of running API Gateways and dynamically updates the limits for each API Gateway when there is a change in the number of running API Gateways.

  The smooth rate limits are stored in an Apache Cassandra database. If you want to use this algorithm, you must first configure an Apache Cassandra connection. For more details, see "Cassandra Settings" in the *API Gateway Administrator Guide*
- **Floating Time Window**: This algorithm is provided for backwards compatibility with previous API Gateway versions. It does not include any traffic smoothing, and is suitable for lower traffic levels, over longer time intervals (for example, 10 transactions per minute or 100 transactions per hour). This means that if a rate limit is set to 100 requests per minute, all 100 can arrive in the first 10 seconds, and will be served. But any requests in next 50 seconds will be rejected. Floating time window is the default algorithm. Its rate limits are stored in a local or distributed cache.

To configure the rate limiting, you must include a **Throttling** filter in your API Gateway policy, and select the desired algorithm: **Smooth Rate Limiting** or **Floating Time Window**.

Further information

For details on how to configure all available settings in the **Throttling** filter for each rate limiting algorithm, see the following:

- "Throttling" in the *API Gateway Policy Developer Filter Reference*

For details on how to deploy API Gateway in elastic and classic environments, see the following:

- *API Gateway Container Deployment Guide*
- *API Gateway DevOps Deployment Guide*

Configure WebSocket connections

WebSocket protocol overview

The WebSocket protocol provides an extension to the HTTP 1.1 protocol to establish persistent, bidirectional communication between a client and a server.

The WebSocket protocol can be summarized as follows:

1. To establish a communication channel between a client and a server, the client needs to send an HTTP `Upgrade` request to the server. This is known as the WebSocket protocol handshake.
2. If the server is capable and willing to upgrade the connection, it sends a HTTP `101` response to the requesting client. At this point the handshake is considered successful and the connection between the server and the client is upgraded to the WebSocket protocol.

   Note As soon as the client receives the HTTP `101` response the connection is no longer considered an HTTP connection.

3. Messages can now flow bidirectionally between the server and the client over the WebSocket connection.
4. Any participant in the data exchange can request the WebSocket connection be terminated by sending a `Close` request to the other participant.

For a detailed description of the protocol, see RFC 6455.

Configure a WebSocket connection

API Gateway can act as a WebSocket proxy, whereby it is deployed in front of a WebSocket capable web server (for example, Jetty or Apache Tomcat) and provides governance (security, monitoring, and so on) on the WebSocket traffic flowing between the client, API Gateway, and the web server.

For a complete example of creating and testing a WebSocket connection, see *WebSocket connection example* on page 311.

WebSocket configuration settings

Configure the following fields on the **WebSocket configuration** dialog:

Enable this path resolver:
Select or deselect the check box to enable or disable the WebSocket handler. It is enabled by default.

Policies settings

You can assign specific policies on this tab to specific URIs that define the WebSocket endpoints. For example, you might need to handle frames being exchanged between a client and `ws://example.org/echo` differently to frames being exchanged between a client and `ws://example.org/voip`.

Note In the above scenario, different sets of policies need to be defined for each URI (`/echo` and `/voip`). This requires different relative paths. For more information on relative paths, see *Configure relative paths* on page 293.

When a request arrives that matches the path:
Enter the path on which WebSocket connections are to be accepted. This defines the URI of the WebSocket endpoint. A relative path resolver for this path must already exist.

Default MIME type for message body:
Enter the MIME type of the messages. The default is `application/json`.

When messages are flowing bidirectionally between a WebSocket server and client, they are no longer HTTP messages and as such they do not contain a Content type header. For API Gateway to process the content of the message it needs to know what type of content the message is. This field enables you to specify the type of message being exchanged between the server and the client.

On Upgrade request from client:
Click the browse button to select a policy to be used by API Gateway when an `Upgrade` request is received.

This policy is executed when a connection is being upgraded from HTTP to WebSocket. For example, you might statically route all WebSocket requests to `ws://example.org` to `wss://example.com` by using a policy. A policy for an HTTP connection must be provided and this policy must provide a mechanism to connect to the remote server.

For example, to route all requests to `ws://example.org` to `wss://example.com`, you can use the **Static Router** filter. Similarly, for dynamic routing you can use the **Dynamic Router** or **Connect to URL** filters. In the latter case, the remote host that the client is attempting to connect to can be extracted from the Host header of the request. This value can then be passed to the **Connect to URL** filter and used by that filter to establish a connection to the remote host.

Note In the routing filter, you must use `http://` or `https://` URLs as API Gateway does not recognize `ws://` and `wss://` URLs. For an example, see *WebSocket connection example* on page 311.

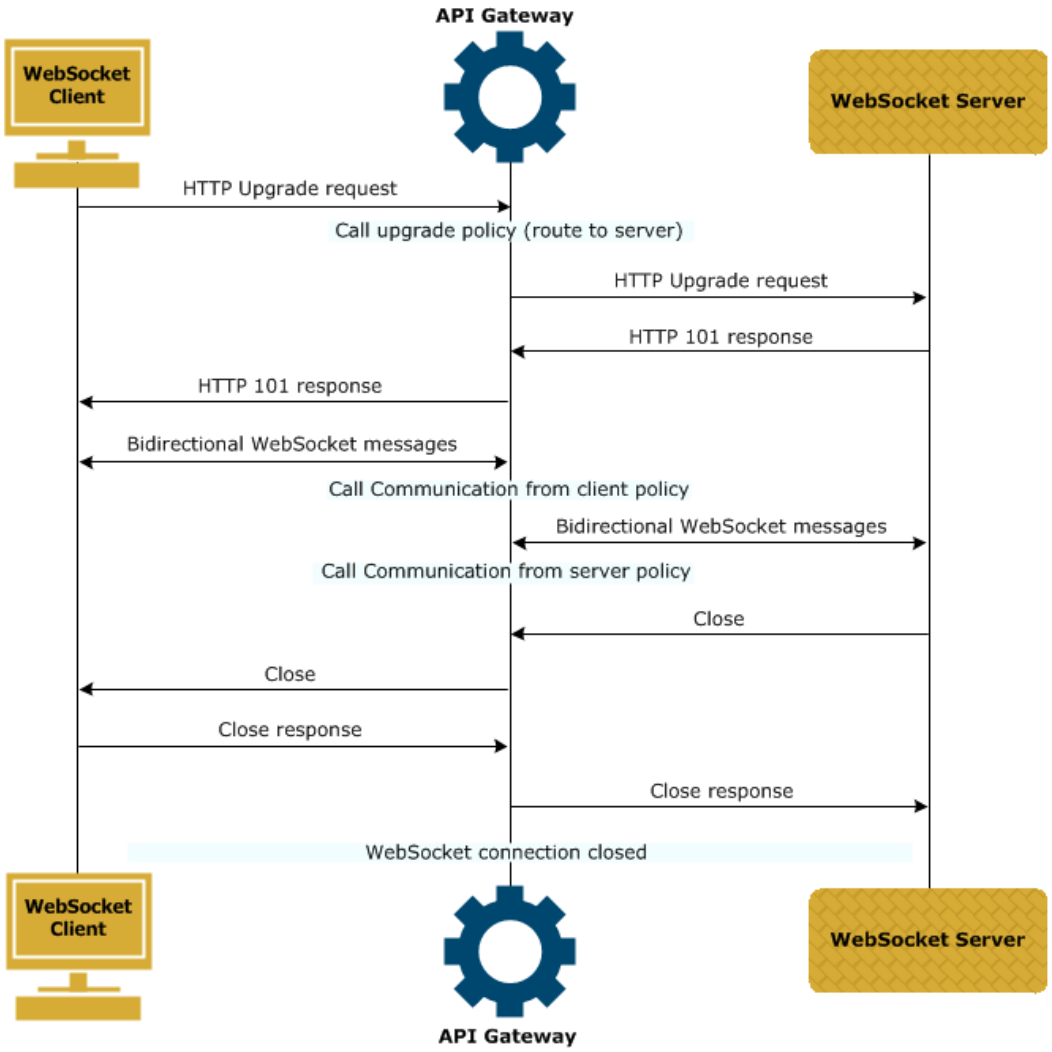
The on upgrade policy is also a good place to perform authentication and authorization of the requesting client.

On WebSocket communication from client:
Click the browse button to select a policy to be used by API Gateway when frames are received from the WebSocket client.

On WebSocket communication from server:
Click the browse button to select a policy to be used by API Gateway when frames are received from the WebSocket server.

Note After a successful upgrade request, the context used to upgrade the connection from HTTP to WebSocket protocol is accessible to the policies called for specific frames. This context is not shared between different WebSocket connections and it is destroyed when the connection is closed. The context contains all the message attributes (including authorization and authentication data) used by the upgrade request. This context should be accessed by the server and client policies in read-only mode.

The following figure shows the flow of messages between the client, API Gateway, and the server in a typical WebSocket communication. It also shows at which point each of the API Gateway policies are called.



Connection expire time:
Enter a numerical value and choose the units from the list. The available options are seconds, minutes, hours, and days. The default value is `0`, which means unlimited.

This is the absolute time for the connection to be active. For example, if this value is set to 1 hour, then after 1 hour the connection is dropped by API Gateway even if the connection is still active (frames are being sent).

Tip You can define an idle timeout for the connection as a part of the remote host configuration.

Advanced settings

For details on the fields on this tab, see *Advanced settings* on page 297.

CORS settings

For details on the fields on this tab, see *CORS settings* on page 297.

Monitor a WebSocket connection

You can use the API Gateway Manager web console to monitor WebSocket traffic. You can view the initial HTTP `Upgrade` request and response on the **Traffic > HTTP** tab. You can view all the WebSocket frames processed by API Gateway on the **Traffic > WebSockets** tab.

To view all the WebSocket frames processed by API Gateway in API Gateway Manager, follow these steps:

1. Click the **Traffic** button at the top of the window.
2. Click the **WebSockets** tab. A view of all WebSocket frames sent from or received by API Gateway is displayed.

   Tip A message might consist of one or more frames.

3. Click a message frame to see a detailed view of the content. For example, if you click a frame sent from the client to the server, the origin, opcode (interpretation of the payload data), duration, length of the data, key used to mask the data, and payload itself are shown.

For more information on traffic monitoring using the API Gateway Manager web console, see the *API Gateway Administrator Guide*.

WebSocket connection example

This example shows you how to create and test a WebSocket connection in API Gateway. API Gateway acts as a proxy for WebSocket services.

To create and test a WebSocket connection, perform the following steps:

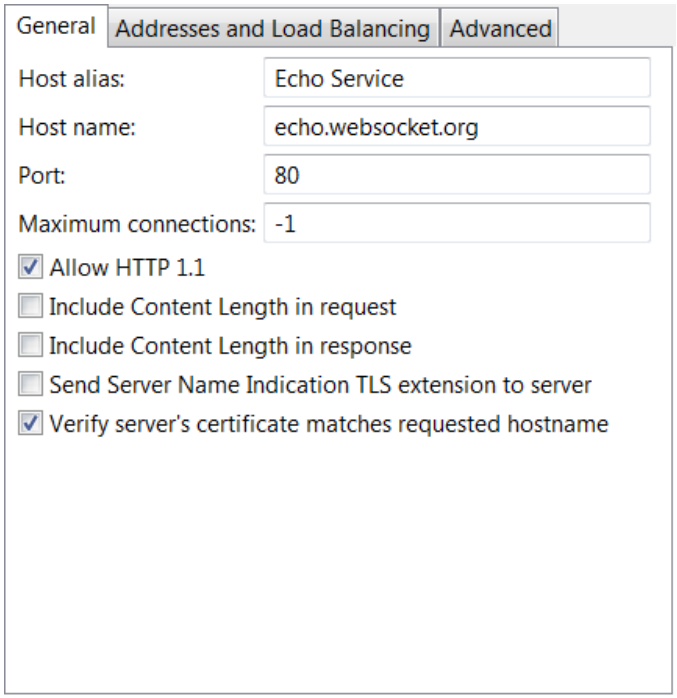
1. Create a HTTP/1.1 remote host for the back-end WebSocket service. See *Create a HTTP/1.1 remote host* on page 311.
2. Create a policy to handle the initial `Upgrade` request from the client. See *Create a policy to handle the Upgrade request* on page 312.
3. (Optional) Create policies to handle data frames from the client and server. See *Create policies to handle data frames* on page 314.
4. Create a WebSocket handler. See *Create a WebSocket handler* on page 314.
5. Test the connection using a WebSocket client. See *Test the WebSocket connection* on page 315.

Create a HTTP/1.1 remote host

Each WebSocket server that API Gateway is routing to must be defined as an HTTP/1.1 remote host.

To add a remote host to an API Gateway instance, right-click the API Gateway instance under **Environment Configuration** > **Listeners** in the Policy Studio tree, and select **Add Remote Host**. For more information, see *Configure remote host settings* on page 268.

In this example, the back-end WebSocket service is provided by a public echo service on the URL `http://echo.websocket.org/`. The remote host configuration is as follows:



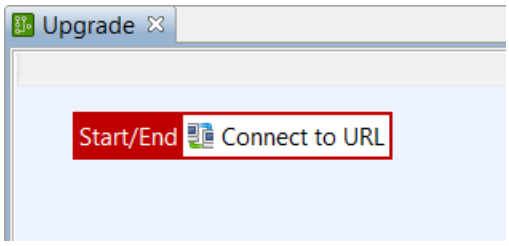
When configuring the remote host:

- Select the **Allow HTTP 1.1** check box.
- Set **Maximum connections** to `-1` (unlimited connections).

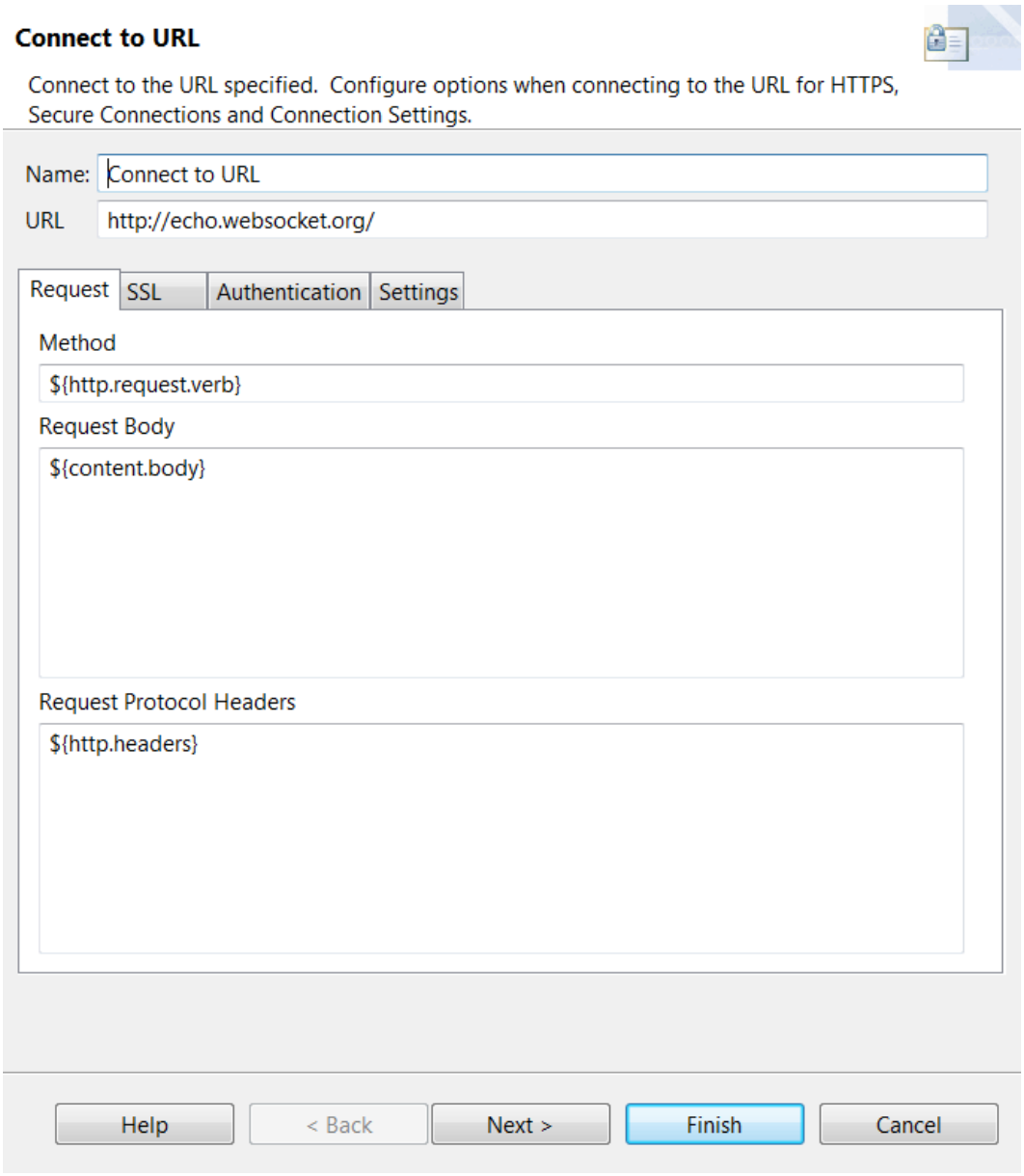
Create a policy to handle the Upgrade request

This policy handles the `Upgrade` request from the client. You can also perform authentication and authorization of the requesting client in this policy.

In this example, the sample Upgrade policy contains a simple **Connect to URL** filter:



The **Connect to URL** filter is configured as follows:



Note The filter points to the URL `http://echo.websocket.org/` and not `ws://echo.websocket.org`. In the URL `http://` is used instead of `ws://` (and similarly `https://` instead of `wss://`) as API Gateway does not recognize `ws://` and `wss://` URLs. This works because the upgrade connection is an HTTP call until the WebSocket server returns `101 - switching protocols`.

Alternatively, you could use a **Static Router** or **Dynamic Router** filter in combination with a **Connection** filter for the Upgrade policy.

Create policies to handle data frames

These are optional policies that trigger on data frames from the client or server. The policies that trigger on data frames from the client and server do not need to make any connections, as the connection is already established. They can modify the WebSocket frames found in `content.body` to alter the frame being sent. To access the HTTP headers from the original Upgrade request, you can find a copy of the `HeaderSet` object inside the `websock.context` attribute that is available on each invocation (the `websock.context` attribute should be considered read-only).

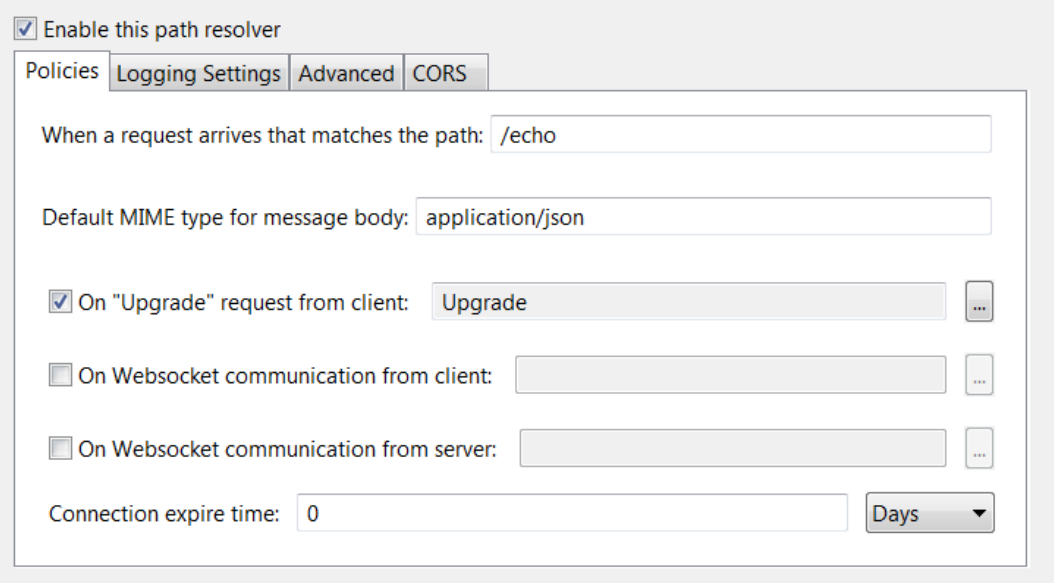
This example does not use any policies to handle data frames.

Create a WebSocket handler

To enable API Gateway to accept an HTTP `Upgrade` request from a client you must add a WebSocket handler to your API Gateway configuration and configure it with the HTTP path that the upgrade can be expected on.

To add a WebSocket handler, follow these steps:

1. In the Policy Studio tree, select a list of relative paths (for example, **Environment Configuration** > **Listeners > API Gateway > Default Services > Paths**).
2. In the **Resolvers** window on the right, click **Add > WebSocket** to display the **WebSocket configuration** dialog. For more information, see *WebSocket configuration settings* on page 308.
3. Configure the WebSocket as follows:

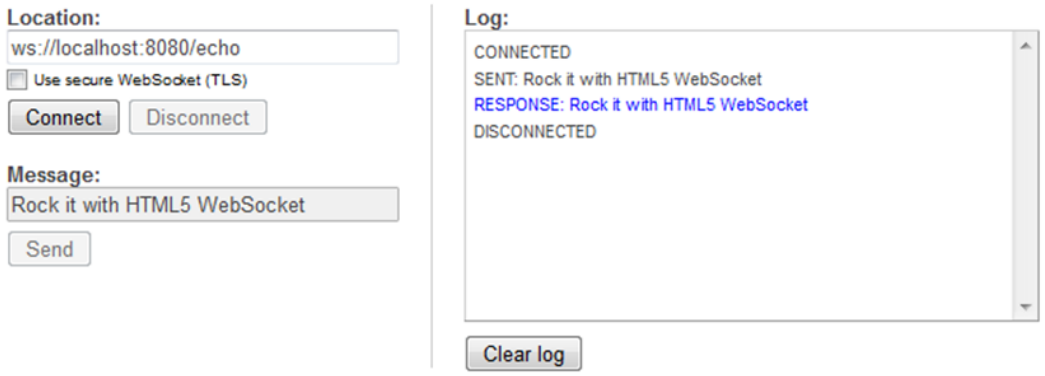


Note Ensure that you have not set `gzip` or `deflate` compression, as it is not compatible for WebSocket connections. For more information, see *Compressed content encoding* on page 430.

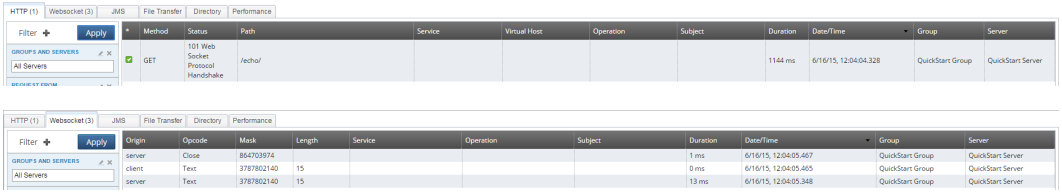
Test the WebSocket connection

To test the WebSocket connection, follow these steps:

1. Deploy the configuration on the API Gateway instance. Click the Deploy button on the toolbar in Policy Studio or press **F6**.
2. Go to https://www.websocket.org/echo.html (a test WebSocket client).
3. Enter the address of your API Gateway WebSocket proxy in the **Location** field and follow the instructions to connect and send messages. For example:



4. View the traffic in API Gateway Manager. For example:



Configure virtual hosts

Overview

A virtual host is a server, or pool of servers, that can host multiple domain names (for example, `company1.api.example.com` and `company2.api.example.com`). This enables you to run more than one website, or set of REST APIs, on a single host machine (for example, `192.0.2.11`). Each domain name can have its own host name, paths, APIs, and so on. For example:

```
https://company1.api.example.com:8080/api/v1/test
https://company2.api.example.com:8080/api/v2/test
https://company3.api.example.com:8080/api/v2/test
```

The API Gateway implements *name-based* virtual hosting, in which the client HTTP `Host` header is used as the routing criteria during path resolution (for example, `Host company1.api.example.com`). This means that you can have multiple domains running on the same hardware (IP address), while this is not apparent to the client or end user.

For example, the following URL invokes the `company2.api.example.com` virtual host:

```
https:/company2.api.example.com/api/v1/test
```

This results in the following message at runtime:

```
POST /api/v1/test HTTP/1.1
Host:company2.api.example.com
Content-Type:application/x-www-form-urlencoded
client_id=SampleConfidentialApp&client_secret=......
```

Note To support name-based virtual hosts, you must first ensure that your Domain Name System (DNS) server has been updated to map each host name to the correct IP address (for example, `*.example.com` is mapped to `192.0.2.11`). For more details on configuring a DNS service with wildcards for virtual hosting, see the *API Gateway Administrator Guide* .

When your DNS server has been updated to map each host name to the correct IP address, you can then configure the API Gateway for virtual hosting.

Configure virtual hosts for HTTP services

You can configure virtual hosts at the HTTP service level. This means that these settings are applied to the HTTP service and to any child resolvers. To configure a virtual host at the HTTP service level, perform the following steps:

1. In the Policy Studio tree, select an HTTP service (for example, **Environment Configuration** > **Listeners** > **API Gateway** > **Default Services** > **Virtual Hosts**).
2. Right-click, and select **Add a Virtual Host**.
3. Configure the following settings in the **Virtual Host** dialog:
    - **Name**:

      Enter a unique name of the virtual host.
    - **Enabled**:

      Select whether the virtual host processing is enabled. This is enabled by default.
    - **Hosts**:

      Specify the list of domains that you wish to host under this HTTP service. To add a host, click **Add** at the bottom right, and enter the domain name (for example `company1.api.example.com`).

You can also specify domain names using wildcards already configured in your DNS (for example `*.example.com:/8080` or `company3.api.example.com.*`). For details on configuring DNS wildcards, see the *API Gateway Administrator Guide*.

Configure child resolvers

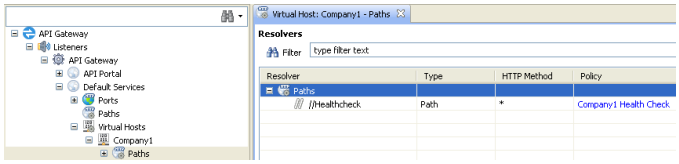
When you have configured a virtual host at the HTTP service level, you can also configure the following child resolvers:

- Relative path
- Static content provider
- Static file provider
- Servlet application

To configure a child resolver, perform the following steps:

1. In the Policy Studio tree, select the **Paths** node under the virtual host, and right-click to add a resolver (for example, **Add relative path**).
2. In the **Resolve path to policies** dialog, click the browse button beside the **Path Specific Policy** field, and select a policy to run on this path.

For example, if an inbound request to `/Healthcheck` matches on `company1.api.*`, the **Company1 Health Check** policy is executed. Otherwise, the global **Health Check** policy for the HTTP service is executed (in this case, **Default Services**).



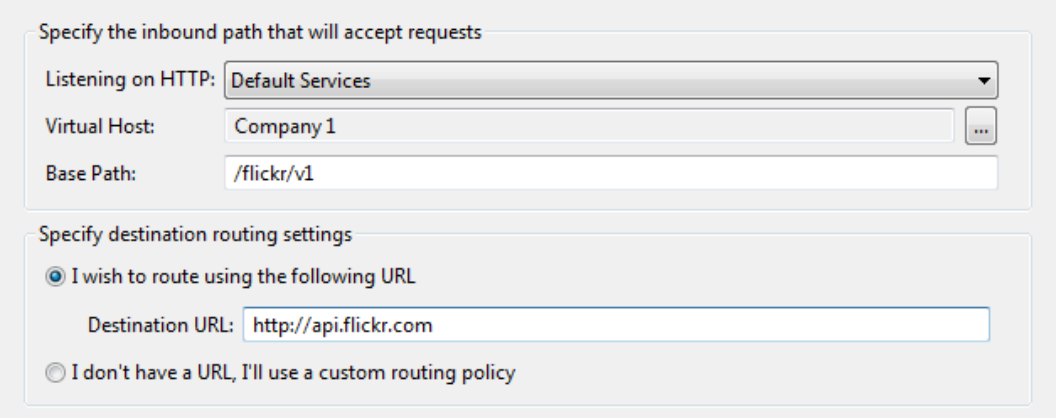
Configure virtual hosts for REST APIs

When API Manager has been installed, you can also configure virtual hosts for specific REST APIs. By default, the HTTP service-level profile is used, but you can override the virtual host at the REST API level.

For example, when adding a new REST API in the **New Rest API** dialog, you can specify virtual hosts on the **Exposure** tab. In the **Virtual Host** field, click the browse button to select a virtual host in the dialog.

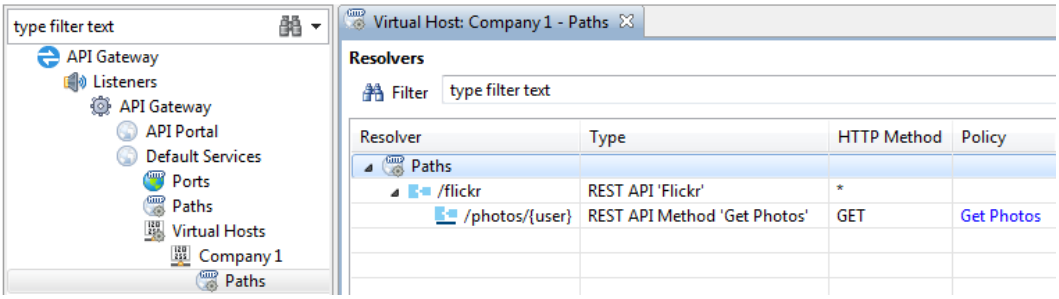
Tip The virtual hosts listed are automatically filtered based on the HTTP service selected on the **Exposure** window (for example, **Default Services**).

If no virtual host has already been configured, right-click the HTTP service node (for example, **Default Services**), and select **Add a Virtual Host**. For more details on virtual host settings, see *Configure virtual hosts for HTTP services* on page 316.



In addition, when editing an existing REST API, you can also specify a virtual host in the **Edit Rest API** dialog.

Finally, when the API has been created, you can view it in the **Resolver** screen for its virtual host. The following shows a simple Flickr API example:



For more details on configuring REST APIs and methods, see the *API Manager User Guide*.

Configure SMTP services

Overview

The API Gateway provides support for Simple Mail Transfer Protocol (SMTP), which enables the API Gateway to receive email and to act as a mail relay. The API Gateway can accept incoming email messages using the SMTP protocol, and then forward them on to a configured mail server. You can also use Policy Studio to configure optional policies for specific SMTP commands (for example, `HELO/EHLO`, `AUTH`, `MAIL FROM`, and so on).

When an SMTP command is configured in Policy Studio, each time the SMTP command is accepted by the API Gateway, the appropriate policy is executed. When the policy completes successfully, the SMTP conversation resumes. This topic shows how to configure SMTP services, interfaces, and handler policies using Policy Studio.

Add an SMTP service

To add an SMTP service to enable the API Gateway to accept SMTP connections, perform the following steps in Policy Studio:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select an API Gateway instance node (for example, the default **API Gateway**).
2. Right-click, and select **Add SMTP Services**.
3. In the **SMTP Services** dialog, specify a unique name for the SMTP service in the **Name** field.
4. In the **Outgoing Server Settings** section, complete the following settings:
    - **Host**

      Host name or IP address of the remote mail server. This is the server to which the API Gateway forwards incoming SMTP commands (for example, `smtp.gmail.com`). You can also specify a mail server running locally on the same machine as the API Gateway using an address of `localhost` or `127.0.0.1`.
    - **Port**

      Port on which to connect to the remote mail server. Defaults to port `25`.
5. In the **Security** section, complete the following settings:
    - **Connection Security**

      Select the type of security used for the connection to the remote mail server. Defaults to `None`. Other possible values are `SSL` and `STARTTLS`.
    - **Trusted Certificates**

      Use this tab to select the trusted CA certificates used in the security handshake for the connection to the remote mail server. This field is mandatory if SSL or STARTTLS connection security is selected.
    - **Client SSL Authentication**

      Use this tab to specify the trusted client certificates used in the security handshake for the connection to the remote mail server. This field is optional if SSL or STARTTLS connection security is selected.
    - **Advanced**

      Use this tab to specify a list of ciphers to use during the security handshake for the connection to the remote mail server. Defaults to `DEFAULT`. For more details, see the OpenSSL ciphers man page. This field is optional if SSL or STARTTLS connection security is selected.

6. In the **Authentication** section, complete the following settings:
   - **Username**

     Specify the user name used to authenticate the API Gateway with a remote SMTP server using the `AUTH` SMTP command. For more details, see *SMTP authentication* on page 327.
   - **Password**

     Specify the password used to authenticate the API Gateway with a remote SMTP server using the `AUTH` SMTP command. For more details, see *SMTP authentication* on page 327.
7. Select the **Include in real time monitoring** check box to monitor the SMTP services using the web-based API Gateway Manager monitoring console.
8. **Click OK**. This creates a tree node for the SMTP service under the selected instance in the **Services** tree.

Add an SMTP interface

When you have configured the outbound SMTP protocol, you must then set up an inbound interface to accept client connections. You can choose from the following interface types:

TCP	Non-secure connection. All traffic is sent in-the-clear.
SSL	SSL handshake is performed at connection time, so the entire SMTP conversation is secure.
STARTTLS	Initial connection is in the clear. The API Gateway advertises STARTTLS during the initial SMTP `HELO/EHLO` handshake. If the client supports this, it can send a STARTTLS command to the API Gateway, which in turn promotes connection security, and upgrades the connection to SSL/TLS.

Because the SSL and STARTTLS interface types have the potential to be secure (STARTTLS starts off non-secure, but can be upgraded during the SMTP conversation), a common configuration window is used for both protocols in Policy Studio.

To configure an inbound interface, perform the following steps in Policy Studio:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select the SMTP node under the instance.
2. Right-click, and select **Add Interface** > interface type (**TCP**, **SSL**, or **STARTTLS**).
3. Complete the settings on the relevant dialog. For full details on these settings, see *Configure HTTP services* on page 275.
4. **Click OK**.

Configure policy handlers for SMTP commands

You can use Policy Studio to configure optional policy handlers for each of the following SMTP commands:

- `HELO/EHLO`
- `AUTH`
- `MAIL FROM`
- `RCPT TO`
- `DATA`

The next sections explain how to configure policy handlers for each command.

Add an HELO/EHLO policy handler

The **HELO/EHLO** policy handler is invoked when a `HELO/EHLO` SMTP command is received from a client. This handler enables you to modify the `HELO/EHLO` greeting and the client domain. You can configure the greeting message sent back to the client from the API Gateway during the `HELO/EHLO` handshake as required. You can also configure a policy to replace the value of `smtp.helo.greeting`. The domain specified by the connected client in the `HELO/EHLO` command can be modified before forwarding on to the remote mail server. You can also configure a policy to replace the value of `smtp.helo.domain`.

To configure a policy handler for the `HELO/EHLO` command, perform the following steps:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select the SMTP node under the instance.
2. Right-click, and select **Add Policy Handler** > **HELO/EHLO**.
3. In the **Configure HELO Host** dialog, specify the **Greeting** to be sent back to the client as part of the `HELO/EHLO` handshake. Defaults to `Hello ${smtp.helo.domain}`.
4. In the **Policy** tree, select the policy that you wish to handle the `HELO/EHLO` command.
5. **Click OK**.

Message attributes
The following message attributes are generated during processing:

Message Attribute	Description
`smtp.helo.domain`	The client domain specified in the `HELO/EHLO` SMTP command received from the client.

Message Attribute	Description
`smtp.helo.greeting`	The HELO greeting to be sent back to the client after `HELO/EHLO` processing is performed. The default value is `Hello ${smtp.helo.domain}`.
`message.source`	The inbound port on which SMTP traffic is received by the API Gateway.
`message.protocol.type`	The protocol used for the connection. This can be `smtp-tcp` or `smtp-ssl`.
`monitoring.enabled`	Set to `true` if monitoring is enabled for the protocol, otherwise `false`.

Add an AUTH policy handler

The **AUTH** policy handler is invoked when an `AUTH` SMTP command is received from a client. You can use the **AUTH** handler to run a policy to perform user authentication checks. For example, during the Authentication phase of the SMTP conversation, the client-supplied username and password can be verified against an Authentication Repository using a policy containing an **Attribute Authentication** filter. For details on possible authentication scenarios, see *SMTP authentication* on page 327.

To configure a policy handler for the `AUTH` command, perform the following steps:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select the **SMTP** node under the instance.
2. Right-click, and select **Add Policy Handler** > **AUTH**.
3. In the **Configure AUTH** dialog, in the **Policy** tree, select the policy that you wish to handle the `AUTH` command.
4. **Click OK**.

Message attributes
The following message attributes are generated during processing:

Message Attribute	Description
`authentication.subject.id`	The user name supplied by the client.
`authentication.subject.password`	The password supplied by the client.
`message.source`	The inbound port on which SMTP traffic is received by the API Gateway.

Message Attribute	Description
`message.protocol.type`	The protocol used for the connection. This can be `smtp-tcp` or `smtp-ssl`.
`monitoring.enabled`	Set to `true` if monitoring is enabled for the protocol, otherwise `false`.

Add a MAIL policy handler

The **MAIL** policy handler is invoked when a `MAIL FROM` SMTP command is received from a client. Emails can be rejected based on wildcard matching of the supplied sender address in the `MAIL FROM` SMTP command. For example, email addresses containing `GMAIL.COM` (`fromAddress` of `*@gmail.com`) as the domain could be accepted using a simple **True** filter. Whereas, email addresses containing `YAHOO.COM` (`fromAddress` of `*@yahoo.com`) could be rejected using a simple **False** filter.

To configure a policy handler for the `MAIL FROM` command, perform the following steps:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select the SMTP node under the instance.
2. Right-click, and select **Add Policy Handler** > **MAIL**.
3. In the **Configure MAIL Address** dialog, you must specify the **From Address**. This is an email address used to filter addresses specified in the `MAIL FROM` SMTP command. You can specify this as a wildcard. The following are some example values:

From Address	Description
`*`	Runs the policy for any email address received.
`*@gmail.com`	Runs the policy for all email addresses with the `gmail.com` domain.
`S*@axway.*`	Runs the policy for all email addresses with any `axway` domain, and beginning with the letter `s`.

   The policy selection is performed on a best-match basis.
4. In the **Policy** tree, select the policy that you wish to handle the `MAIL FROM` command.
5. **Click OK**.

You can configure multiple **MAIL** handlers so that different policies are executed, depending on the received mail address.

Message attributes
The following message attributes are generated during processing:

Message Attribute	Description
`smtp.helo.domain`	The client domain specified in the `HELO/EHLO` SMTP command received from the client.
`smtp.mail.from`	The email address specified in the `MAIL FROM` SMTP command received from the client.
`message.source`	The inbound port on which SMTP traffic is received by the API Gateway.
`message.protocol.type`	The protocol used for the connection. This can be `smtp-tcp` or `smtp-ssl`.
`monitoring.enabled`	Set to `true` if monitoring is enabled for the protocol, otherwise `false`.

Add a RCPT policy handler

The **RCPT** policy handler is invoked when a `RCPT TO` SMTP command is received from a client. You can use this handler to filter addresses specified in the `RCPT TO` SMTP command. Recipients can be rejected based on wildcard matching of the supplied recipient address in the `RCPT` SMTP command.

For example, recipient addresses containing `GMAIL.COM` (`toAddress` of `*@gmail.com`) as the domain could be accepted using a simple **True** filter. Whereas, addresses containing `YAHOO.COM` (`toAddress` of `*@yahoo.com`) could be rejected using a simple **False** filter. You can configure multiple **RCPT** handlers so that different policies are executed, depending on the received email address.

To configure a policy handler for the `RCPT TO` command, perform the following steps:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select the SMTP node under the instance.
2. Right-click, and select **Add Policy Handler** > **RCPT**.
3. In the **Configure Recipient Address** dialog, you must specify the **To Address**. This is an email address used to filter addresses specified in the `RCPT TO` SMTP command. You can specify this as a wildcard. The following are some example values:

To Address	Description
`*`	Runs the policy for any email address received.
`*@axway.com`	Runs the policy for all email addresses with the `axway.com` domain.

To Address	Description
`d*@yahoo.*`	Runs the policy for all email addresses with any `yahoo` domain, and beginning with the letter `d`.

The policy selection is performed on a best-match basis.

4. In the **Policy** tree, select the policy that you wish to handle the `MAIL FROM` command.
5. **Click OK**.

Message attributes

The following message attributes are generated during processing:

Message Attribute	Description
`smtp.helo.domain`	The client domain specified in the `HELO/EHLO` SMTP command received from the client.
`smtp.mail.from`	The email address specified in the `MAIL FROM` SMTP command received from the client.
`smtp.rcpt.to`	The email address specified in the `RCPT TO` SMTP command received from the client. **Note** This is the current recipient being processed, whereas `smtp.rcpt.recipients` is the list of recipients processed so far.
`smtp.rcpt.recipients`	The list (collection of strings) of recipients (email addresses) received or processed *so far* by the SMTP transaction. This is read-only and updated by the API Gateway each time it receives a `RCPT TO` command from the client.
`message.source`	The inbound port on which SMTP traffic is received by the API Gateway.
`message.protocol.type`	The protocol used for the connection. This can be `smtp-tcp` or `smtp-ssl`.
`monitoring.enabled`	Set to `true` if monitoring is enabled for the protocol, otherwise `false`.

Add a DATA policy handler

The **DATA** policy handler is invoked when a `DATA` SMTP command is received from a client. For example, for emails that contain SOAP/XML content, you can add an XML signature to the XML data, stored in the `content.body` message attribute, using an **XML Signature Generation** filter. For emails containing attachments, the attached mail data can be run through one of the API Gateway anti-virus filters.

Alternatively, you can use **SMIME Encrypt** or **SMIME Decrypt** filters to encrypt or decrypt emails (including attachments) passing through the API Gateway. You can also digitally sign emails using an **SMIME Sign** filter, or verify signatures on already digitally signed emails using an **SMIME Verify** filter.

To configure a policy handler for the `DATA` command, perform the following steps:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select the **SMTP** node under the instance.
2. Right-click, and select **Add Policy Handler** > **DATA**.
3. In the **Policy** tree, select the policy that you wish to handle the `DATA` command.
4. **Click OK**.

Message attributes
The following message attributes are added during processing:

Message Attribute	Description
`smtp.helo.domain`	The client domain specified in the `HELO/EHLO` SMTP command received from the client.
`smtp.mail.from`	The email address specified in the `MAIL FROM` SMTP command received from the client.
`smtp.rcpt.recipients`	The full list (collection of strings) of recipients (email addresses) processed by the SMTP transaction.
`content.body`	The stream representing the body of the mail. **Note** The `content.body` does not include MIME headers.
`message.source`	The inbound port on which SMTP traffic is received by the API Gateway.
`message.protocol.type`	The protocol used for the connection. This can be `smtp-tcp` or `smtp-ssl`.

Message Attribute	Description
`monitoring.enabled`	Set to `true` if monitoring is enabled for the protocol, otherwise `false`.

SMTP authentication

The SMTP protocol supports Extended SMTP (ESMTP) PLAIN authentication. The following matrix shows the possible authentication scenarios and actions based on the SMTP Services configuration:

Scenario	AUTH handler	AUTH user name and password	Mail server advertises AUTH	API Gateway advertises AUTH	Proxy client AUTH	Authenticate API Gateway to server
1	No	No	No	No	No	No
2	No	No	Yes	Yes	Yes	No
3	No	Yes	No	No	No	No
4	No	Yes	Yes	No	No	Yes
5	Yes	No	No	Yes	No	No
6	Yes	No	Yes	Yes	No	No
7	Yes	Yes	No	Yes	No	No
8	Yes	Yes	Yes	Yes	No	Yes

These authentication scenarios are described as follows:

1. No authentication user name and password are specified so the API Gateway does not attempt to authenticate with the server. The server does not support authentication anyway. The mail server does not advertise authentication so the API Gateway does not advertise AUTH to the client. The client authentication is not proxied because the server does not support it.
2. No authentication user name and password are specified so the API Gateway does not attempt to authenticate with the server. The server does not support authentication anyway. The mail server advertises AUTH, so the API Gateway advertises AUTH to the client. No AUTH handler is configured, so the client authentication details are proxied to the server.
3. Same as 1 above.

4. The authentication user name and password are specified so the API Gateway authenticates with the server. The mail server advertises AUTH, but because a user name and password are specified, the API Gateway does not advertise AUTH to the client because the API Gateway authenticates with the server using the configured credentials. This also implies no client authentication proxying.
5. No authentication user name and password are specified so the API Gateway does not attempt to authenticate with the server. The server does not support authentication anyway. AUTH handler configured, which implies the API Gateway performs authentication, so advertise AUTH to the client.
6. Same as 5 above.
7. AUTH handler configured, which implies the API Gateway performs authentication, so advertise AUTH to the client. No proxying occurs because the API Gateway performs the authentication. No authentication is performed with the server because the server does not support it.
8. AUTH advertised to the client because the API Gateway performs authentication (and the mail server supports it). AUTH handler configured, which implies the API Gateway performs authentication. No proxying occurs because the API Gateway performs the authentication. Authentication is performed with the server because the server supports AUTH and a user name and password is configured.

SMTP Content-Transfer-Encoding

The SMTP protocol supports automatic Content-Transfer-Encoding/Decoding. For `DATA` SMTP commands, the content of the incoming mail body may be encoded. To enable policy filters to view and/or manipulate the raw body data, the contents are automatically decoded before policy execution, and re-encoded afterwards (before being forwarded on to the configured outbound mail server).

Supported encodings
The following encodings are supported:

- Base64
- 7-bit
- 8-bit
- quoted-printable
- binary

However, Base64 is the only encoding that results in decoding/re-encoding of the mail data.

Multipart MIME content, generally used for sending attachments in SMTP, is also supported. Each separate body in the multipart is checked for a Content-Transfer-Encoding, and the decoding/re-encoding is performed as appropriate.

Deployment example

This section provides a step-by-step example of how to configure and deploy SMTP services using the API Gateway. In this example, the API Gateway acts as a relay between a Thunderbird email client and the Google Gmail service.

Configure the API Gateway SMTP services

The API Gateway connects to the Gmail STARTTLS interface, which is available at `smtp.gmail.com`, and listening on port `587`.To configure the SMTP Services, perform the following steps in Policy Studio:

1. Under the **Environment Configuration** > **Listeners** node in the tree, select a Process node (for example, the default **API Gateway**).
2. Right-click, and select **Add SMTP Services**.
3. Enter `smtp.gmail.com` for the **Host**.
4. Enter `587` for the **Port**.
5. Select `STARTTLS` from the **Connection Security** drop-down list. This is selected because `smtp.gmail.com:587` exposes the Gmail STARTTLS SMTP interface.
6. Because STARTTLS has the potential to be upgraded to a secure connection, you must also select some **Trusted Certificates**.
7. Accept all other defaults, and click **OK** to add the SMTP services.

Configure the SMTP client interface

To configure a STARTTLS client interface, perform the following steps in Policy Studio:

1. Right-click the **SMTP** Services node, and select **Add Interface** > **STARTTLS**.
2. Enter a **Port** (for example, `8026`). This is the port on which the API Gateway's incoming SMTP traffic is accepted. You can enter any port that is not already in use.
3. Because STARTTLS has the potential to be upgraded to a secure connection, you must configure a trusted certificate. Click the **X.509 Certificate** button.
4. Select a certificate in the **Select Certificate** dialog.
5. **Click OK** to return to the **Configure STARTTLS Interface** dialog.
6. When the certificate has been configured, accept all other defaults, and click **OK** to add the incoming STARTTLS interface.

When the SMTP services and STARTTLS client interface have been configured, you must deploy the changes to the API Gateway.

Configure Thunderbird client settings

This example uses Thunderbird as the email client. However, you can use any standard email client that supports SMTP. Thunderbird is available as a free download from http://www.mozillamessaging.com/.

To configure a STARTTLS outgoing server in your Thunderbird client, perform the following steps:

1. Launch the Thunderbird email client.
2. From the main menu, select **Tools** > **Account Settings**.
3. Expand the **Local Folders** tree node in the left pane.
4. Select the **Outgoing Server** node to create a new outgoing server configuration.
5. Click **Add** to display the **SMTP Server** dialog.
6. Enter `Axway API Gateway [STARTTLS]` in the **Description** field.
7. Enter `localhost` (or the IP Address of the machine on which the API Gateway service is running) in the **Server Name** field.
8. Enter `8026` in the **Port** field. This sends SMTP traffic to the STARTTLS interface configured above, so the ports must match.
9. Select `STARTTLS` from the **Connection security** drop-down list. Traffic on this connection may be upgraded to secure during the SMTP conversation.
10. Select `Normal Password` from the **Authentication method** drop-down list. This indicates that Authentication is to be performed.
11. Enter a valid Gmail user-name for the **User Name**.
12. Click **OK** to add the new outgoing server configuration.

Configure certificates in Thunderbird

To enable Thunderbird to successfully negotiate the STARTTLS conversation with the API Gateway, you must import a CA certificate into Thunderbird. This is also because a certificate was already generated and imported into the API Gateway when configuring its STARTTLS client interface.

To configure a STARTTLS outgoing server in your Thunderbird client, perform the following steps:

1. From the Thunderbird main menu, select **Tools** > **Options**.
2. Select the **Certificates** tab.
3. Click the **View Certificates** button, to display the **Certificate Manager** dialog.
4. Click **Import**, and import the appropriate CA certificate.
5. Click **OK** when finished.

Test the STARTTLS client interface

To test the STARTTLS client interface using Thunderbird, perform the following steps:

1. Launch the Thunderbird email client, and create a new mail message.
2. Enter a valid Gmail address in the **To** field.
3. Enter `API Gateway Test` as the **Subject**.
4. Enter `This mail has been sent using Axway API Gateway` in the mail body.
5. To specify the appropriate outgoing mail server, select **Tools** > **Account Settings** from the main menu.
6. Select `Axway API Gateway [STARTTLS] - localhost` from the **Outgoing Server** drop-down list.
7. Click **OK**.
8. Send the mail.

The following example from the API Gateway trace shows the SMTP commands that occur. Commands marked in **bold text** shows traffic from the Thunderbird client to the API Gateway and vice versa. Commands marked in *italic text* shows traffic from the API Gateway to the Gmail server at `smtp.gmail.com:587`, and vice versa.

```
DEBUG 14:46:46:546 [14b4] incoming call on interface *:8026 from 127.0.0.1:1487
DEBUG 14:46:46:546 [14b4] new connection 08133248, settings source incoming
interface
(force 1.0=no, idleTimeout=60000, activeTimeout=60000)
DATA 14:46:46:546 [14b4] snd 0018: <220 doejAxway>
DATA 14:46:46:562 [14b4] rcv 18: <EHLO [127.0.0.1]>
DEBUG 14:46:46:562 [14b4] 080BE260: new connection cache set SMTP Client
DEBUG 14:46:46:562 [159c] idle connection monitor thread running
DEBUG 14:46:46:562 [14b4] new endpoint smtp.gmail.com:587
DEBUG 14:46:46:640 [14b4] Resolved smtp.gmail.com:587 to:
DEBUG 14:46:46:640 [14b4] 209.85.227.109:587
DEBUG 14:46:46:718 [14b4] connected to 209.85.227.109:587
DEBUG 14:46:46:718 [14b4] new connection 08135BA0, settings source service-wide
defaults (force 1.0=no, idleTimeout=15000, activeTimeout=30000)
DATA 14:46:46:765 [14b4] rcv 44: <220 mx.google.com ESMTP v11sm7979387weq.40>
DATA 14:46:46:765 [14b4] snd 0018: <ehlo [127.0.0.1]>
DATA 14:46:46:812 [14b4] rcv 125: <250-mx.google.com at your service,
[87.198.245.194]
250-SIZE 35651584
```

```
250-8BITMIME
250-STARTTLS
250 ENHANCEDSTATUSCODES>
DATA 14:46:46:812 [14b4] snd 0010: <starttls>
DATA 14:46:46:843 [14b4] rcv 30: <220 2.0.0 Ready to start TLS>
DEBUG 14:46:46:843 [14b4] push SSL protocol on to connection
DEBUG 14:46:46:906 [14b4] No SSL host name provided: using default certificate for
interface
DEBUG 14:46:46:906 [14b4] verifyCert: preverify=1, depth=2, subject /C=US/O=Equifax/
OU=Equifax Secure Certificate Authority, issuer /C=US/O=Equifax/OU=Equifax Secure
Certificate Authority
DEBUG 14:46:46:906 [14b4] ca cert? 1
DEBUG 14:46:46:906 [14b4] verifyCert: preverify=1, depth=1, subject /O=Google
Inc/CN=Google Internet Authority, issuer /C=US/O=Equifax/OU=Equifax Secure
Certificate
Authority
DEBUG 14:46:46:906 [14b4] verifyCert: preverify=1, depth=0, subject
/C=US/ST=California/L=Mountain View/O=Google Inc/CN=smtp.gmail.com,
issuer /C=US/O=Google Inc/CN=Google Internet Authority
DEBUG 14:46:46:952 [14b4] negotiated SSL cipher "RC4-MD5",session 00000000 (not
reused)
DATA 14:46:46:952 [14b4] snd 0018: <ehlo [127.0.0.1]>
DATA 14:46:46:999 [14b4] rcv 140: <250-mx.google.com at your service,
[87.198.245.194]
250-SIZE 35651584
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
250 ENHANCEDSTATUSCODES>
DATA 14:46:46:999 [14b4] snd 0109: <250-AxwayAPI Gateway Hello [127.0.0.1]
250-SIZE 35651584
250-8BITMIME
250-STARTTLS
250 ENHANCEDSTATUSCODES>
DEBUG 14:46:46:999 [14b4] delete transaction 0B95D2C0 on connection 08133248
DATA 14:46:46:999 [14b4] rcv 10: <STARTTLS>
DATA 14:46:46:999 [14b4] snd 0014: <220 Go ahead>
DEBUG 14:46:46:999 [14b4] push SSL protocol on to connection
DEBUG 14:46:46:999 [14b4] Servername CB: SSL host name: localhost, not in host map -
using default certificate for interface
DEBUG 14:46:47:031 [14b4] negotiated SSL cipher "AES256-SHA", session 00000000
(not reused)
DATA 14:46:47:031 [14b4] rcv 18: <EHLO [127.0.0.1]>
DATA 14:46:47:031 [14b4] snd 0018: <ehlo [127.0.0.1]>
DATA 14:46:47:077 [14b4] rcv 140: <250-mx.google.com at your service,
[87.198.245.194]
250-SIZE 35651584
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
```

```
250 ENHANCEDSTATUSCODES>
DATA 14:46:47:077 [14b4] snd 0124: <250-AxwayAPI Gateway Hello [127.0.0.1]
250-SIZE 35651584
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH
250 ENHANCEDSTATUSCODES>
DEBUG 14:46:47:077 [14b4] delete transaction 0B95D2C0 on connection 08133248
DATA 14:46:47:077 [14b4] rcv 41: <AUTH PLAIN ADGzaHllaDe0SHF1ex2r82Su555=>
DATA 14:46:47:077 [14b4] snd 0041: <auth PLAIN ADGzaHllaDe0SHF1ex2r82Su555=>
DATA 14:46:47:718 [14b4] rcv 20: <235 2.7.0 Accepted>
DATA 14:46:47:718 [14b4] snd 0020: <235 2.7.0 Accepted>
DATA 14:46:47:718 [14b4] rcv 45: <MAIL FROM:<john.doe@axway.com> SIZE=444>
DATA 14:46:47:718 [14b4] snd 0036: <mail from:<john.doe@axway.com>>
DATA 14:46:47:765 [14b4] rcv 33: <250 2.1.0 OK v11sm7979387weq.40>
DATA 14:46:47:765 [14b4] snd 0033: <250 2.1.0 OK v11sm7979387weq.40>
DATA 14:46:47:765 [14b4] rcv 30: <RCPT TO:<test@gmail.com>>
DATA 14:46:47:765 [14b4] snd 0030: <rcpt to:<test@gmail.com>>
DATA 14:46:47:812 [14b4] rcv 33: <250 2.1.5 OK v11sm7979387weq.40>
DATA 14:46:47:812 [14b4] snd 0033: <250 2.1.5 OK v11sm7979387weq.40>
DATA 14:46:47:812 [14b4] rcv 6: <DATA>
DATA 14:46:47:812 [14b4] snd 0006: <data>
DATA 14:46:48:609 [14b4] rcv 34: <354 Go ahead v11sm7979387weq.40>
DATA 14:46:48:609 [14b4] snd 0008: <354 OK>
DATA 14:46:48:609 [14b4] rcv 447: <Message-ID: <4CB85B46.4060205@axway.com>
Date: Fri, 15 Oct 2010 14:46:46 +0100
From: John Doe <john.doe@axway.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.2.9)
Gecko/20100915
Thunderbird/3.1.4
MIME-Version: 1.0
To: test@gmail.com
Subject: API Gateway Test
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
This mail has been sent via an Axway API Gateway
.>
DATA 14:46:48:609 [14b4] snd 0442: <Message-ID: <4CB85B46.4060205@axway.com>
Date: Fri, 15 Oct 2010 14:46:46 +0100
From: John Doe <john.doe@axway.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.2.9)
Gecko/20100915
Thunderbird/3.1.4
MIME-Version: 1.0
To: test@gmail.com
Subject: API Gateway Test
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
This mail has been sent via an Axway API Gateway>
DATA 14:46:48:609 [14b4] snd 0005: <.>
DATA 14:46:49:874 [14b4] rcv 44: <250 2.0.0 OK 1287150409 v11sm7979387weq.40>
DATA 14:46:49:874 [14b4] snd 0044: <250 2.0.0 OK 1287150409 v11sm7979387weq.40>
```

```
DEBUG 14:46:49:874 [14b4] delete transaction 0B95D2C0 on connection 08133248
DATA 14:46:49:874 [14b4] rcv 6: <QUIT>
DATA 14:46:49:874 [14b4] snd 0006: <quit>
DATA 14:46:49:921 [14b4] rcv 49: <221 2.0.0 closing connection v11sm7979387weq.40>
DEBUG 14:46:49:921 [14b4] delete transaction 08040BD8 on connection 08135BA0
DATA 14:46:49:921 [14b4] snd 0049: <221 2.0.0 closing connection v11sm7979387weq.40>
DEBUG 14:46:49:921 [14b4] delete transaction 0B95D2C0 on connection 08133248
DEBUG 14:46:49:921 [14b4] delete connection 08133248, current transaction 00000000
```

Configure a file transfer service

Overview

The API Gateway can act as a file transfer service that listens on a port for remote clients to connect to it. The API Gateway file transfer service supports the following protocols:

- **FTP**: File Transfer Protocol
- **FTPS**: FTP over Secure Sockets Layer (SSL)
- **SFTP**: Secure Shell (SSH) File Transfer Protocol

For all file transfer protocols, you must configure a file upload policy and an authentication policy. For FTP and FTPS, you must configure a password authentication policy. While for SFTP, you can configure a password authentication policy or a public key authentication policy. The API Gateway can also restrict access to the server based on IP address.

When a file transfer service is configured, users are presented with a personal file system view when they log in. The root of this file system is specified in a configurable request directory. Any files they upload are processed by the file upload policy. If this policy succeeds, the output of the policy is stored in a configurable response directory. If the policy fails, the original file is moved to a configurable quarantine directory.

Configuring a file transfer service can be useful when integrating with Business-to-Business (B2B) partner destinations or with legacy systems. For example, instead of making drastic changes to either system, the other system can upload files to the API Gateway. The added benefit is that the file transfer can be controlled and secured using API Gateway policies designed to suit system needs.

Tip For details on how to use the API Gateway to poll a remote file server, to query and retrieve files to be processed, see *Configure an FTP poller* on page 342.

General settings

You can configure the following settings in the **General** section:

Name:
Enter an appropriate name for the file transfer service.

Service Type:
Select the file transfer protocol type for this service from the drop-down list (`ftp`, `ftps`, or `sftp`). Defaults to `ftp`.

Implicit:
This setting applies to FTPS only. When selected, security is automatically enabled as soon as the remote client makes a connection to the file transfer service. No clear text is passed between the client and server at any time. In this case, the file transfer service defines a specific port for the remote client to use for secure connections (`990`). This setting is not selected by default.

Explicit:
This setting applies to FTPS only. When selected, the remote client must explicitly request security from the file transfer server, and negotiate the required security. If the client does not request security, the file transfer server can allow the client to continue insecure or refuse and/or limit the connection. This setting is selected by default.

Binding Address:
Enter a network interface to bind to. Defaults to `*`, which means bind to all available network interfaces on the host on which the API Gateway is installed. You can enter an IP address to bind to a specific network interface.

Port:
Enter a file transfer port to listen on for remote clients to connect to. Defaults to `21`.

Note The API Gateway must execute with superuser privileges to bind to a port number less than `1024` on Linux.

File upload settings

For all file transfer protocols, you must specify a **File Upload** policy to be invoked with the file data. This enables files and directories to be uploaded to a user subdirectory of the **Request Directory**.

For example, files uploaded by user `fred` are placed in `${environment.VINSTDIR}/file-transfer/in/persistent/fred`, where `VINSTDIR` is the location of the running API Gateway instance. The specified policy is then invoked with the raw file data. If this policy returns true, the output is placed in the corresponding **Response Directory**. If this policy returns false or an exception is thrown, the uploaded file is moved to the **Quarantine Directory**.

You can configure the following settings in the **File Upload** section:

Request Directory:
Specifies the directory into which files and directories are uploaded. Defaults to `${environment.VINSTDIR}/file-transfer/in/`.

Delete File on Successful Response:
Select whether to delete the uploaded files from the **Request Directory** when successfully processed. This setting is not selected by default.

Response Directory:
Specifies the name of the directory in which files output by the API Gateway processing of uploaded files are placed if the **File Upload** policy returns true. Defaults to `out`. The original files remain in the **Request Directory**.

Response Suffix:
Specifies the file name suffix that is appended to the output files in the **Response Directory**. Defaults to `.resp.${id}`, which enables a unique suffix to be appended to the files.

Quarantine Directory:
Specifies the directory into which the uploaded files are moved if the **File Upload** policy returns false, or an exception is thrown. Defaults to `quarantine`.

Note The response and quarantine directories can be relative or absolute. Relative directories reside under the request directory. The user can manage the uploaded files using their file transfer session (for example, by accessing API Gateway file processing results). Absolute directories must reside outside of the request directory. The user cannot view or manage uploaded files using their file transfer session. Specifying absolute directories hides API Gateway file processing from the user.

In this way, the request directory can be seen as the user's home directory for the duration of the connection. Therefore, anything created under the same home directory is visible to the user. However, if the response is created outside this directory (for example, in `/tmp/response`), the files are not visible to the user.

Specify a file upload policy

You must specify a **File Upload** policy to be invoked with the raw file data. Perform the following steps:

1. Click the **Add** button to display the dialog.
2. In the **Pattern** field, select or enter a regular expression to match against the file name. For example, the following expression means that the configured policy is run against these files types only:

```
([^\s]+(\.(?i)(xml|xhtml|soap|wsdl|asmx))$)
```

3. In the **Policy** field, click the browse button on the right to select a policy, and click **OK**.
4. Click **OK** to display the configured pattern and policy in the table.

Message attributes

The **File Upload** policy uses the following message attributes:

- `content.body`: Raw message file content.
- `file.src.name`: File name of the uploaded file.
- `file.src.path`: Full file path of the uploaded file.

Secure services settings

On the **Secure Services** tab, you can configure the following **Client Authentication** policies:

Password Authentication Policy:
For FTP and FTPS, you must configure a **Password Authentication Policy**. Click the browse button on the right to select a configured policy. This policy uses the `authentication.subject.id` and `authentication.subject.password` message attributes, which store the user name and password entered by the client user.

Public Key Authentication Policy:
For SFTP, you can configure a **Public Key Authentication Policy** or **Password Authentication Policy**. Click the browse button on the right to select a configured policy. This policy uses the `authentication.subject.id` and `authentication.subject.public.key` message attributes, which store the user name and public key used by the client.

You can configure the following **Server Authentication** settings:

Server Certificate:
For FTPS or SFTP, click the **Signing Key** button to specify a server certificate. You can select a certificate in the dialog, or click to create or import a certificate. The selected certificate must contain a private key. For more details, see *Manage X.509 certificates and keys* on page 221. Alternatively, you can specify a certificate to bind to at runtime using an environment variable selector (for example, `${env.serverCertificate}`). For details on setting external environment variables for API Gateway instances, see the *API Gateway DevOps Deployment Guide*.

Server Key Pair:
For SFTP, click the button on the right, and select a previously configured key pair that the file transfer service must present from the tree. To add a key pair, right-click the **Key Pairs** node, and select **Add**. Alternatively, you can import key pairs under the **Environment Configuration** > **Certificates and Keys** node in the Policy Studio tree. For more details, see *Manage X.509 certificates and keys* on page 221.

Command settings

For FTP and FTPS, the **Commands** tab enables you to specify commands that can be enabled for this service (other FTP and FTPS commands are enabled by default). The following commands are specified in the table:

- `DELE`: Allow user to delete files
- `PASV`: Support passive mode
- `REST`: Support restart mode
- `RMD`: Allow user to remove directories
- `STOU`: Support unique file name

To enable an existing command, click **Edit**, select **Enabled**, and click **OK**. The command is displayed as enabled in the table.

Add new commands

To add a new command, perform the following steps:

1. Click the **Add** button to display the dialog.
2. Enter the command **Name** (for example, `STAT`).
3. Select the file transfer protocol **Type** from the drop-down list (for example, `ftp` or `ftp(s)`).
4. Enter the command **Description**.
5. Select whether the command is **Enabled**.
6. Click **OK** to display the new command in the table.

Supported commands

For a full list of supported commands, see http://mina.apache.org/ftpserver-project/ftpserver_commands.html.

Access control settings

The **Access Control** tab enables you to restrict or block access to the file transfer service based on IP address. All IP addresses are allowed by default.

Restrict Access to the following IP Ranges:
To restrict access to specified IP addresses, perform the following steps:

1. Click the **Add** button to display the dialog.
2. Enter an **IP Address** (for example, `192.168.0.16`).
3. Enter a **Net Mask** for the specified IP address. For example, if you wish to restrict access to the specified IP address only, enter `255.255.255.255`. Alternatively, you can restrict access to a range of IP addresses by entering a value such as `255.255.255.253`, which restricts access to `192.168.0.16`, `192.168.0.17`, and `192.168.0.18`.
4. Click **OK** to display the IP address details in the table.

Block Access to the following IP Ranges:
To block access to specified IP addresses, perform the following steps:

1. Click the **Add** button to display the dialog.
2. Enter an **IP Address** (for example, `192.168.0.16`).

3. Enter a subnet **Mask** for the specified IP address. For example, if you wish to block access to the specified IP address only, enter `255.255.255.255`. Alternatively, you can block access to a range of IP addresses by entering a value such as `255.255.255.253`, which blocks access to `192.168.0.16`, `192.168.0.17`, and `192.168.0.18`.
4. Click **OK** to display the IP address details in the table.

Message settings

For FTP and FTPS, the you can specify a welcome message and a goodbye message on the **Messages** tab. These are output to the user at the start and at the end of each session.

Welcome Message:
Enter a short welcome text message for the file transfer service (for example, `Connected to FTP Test Service...`).

Goodbye Message:
Enter a short goodbye text message for the file transfer service (for example, `Leaving FTP Test Service...`).

Directory settings

You can configure the following settings on the **Directory** tab:

Directory Type:
Select one of the following directory types:

- **Persistent**: This is a sharable directory created per user, which persists between connections (for example, `${environment.VINSTDIR}/file-transfer/in/persistent/fred`). This is the default directory type.
- **Transient**: This is an isolated directory created per connection, which is destroyed after the connection (for example, `${environment.VINSTDIR}/file-transfer/in/2/fred`).

Note Some clients, notably FileZilla, generate multiple client connection sessions. For these clients, files uploaded in one session are not available in the viewing session.

Directory expiry in seconds:
Specifies how long the directory remains on the system. Defaults to 3600 seconds (1 hour). A setting of 0 seconds means that the directory never expires.

Directory expiry check period in seconds:
Specifies how often the API Gateway checks to see if a directory has expired. Defaults to 1800 seconds (30 minutes). A setting of 0 seconds means that it never checks.

Logging settings

The **Logging Settings** tab enables you to configure the logging level for the file transfer service, and to configure when message payloads are logged.

Logging Level
You can configure the following settings on all filters executed on the file transfer service:

Logging Level	Description
Fatal	Logs Fatal log points that occur on all filters executed.
Failure	Logs Failure log points that occur on all filters executed. This is the default logging level.
Success	Logs Success log points that occur on all filters executed.

For more details on logging levels, and on how to configure logging for a specific filter, see "Set transaction log level and log message" in the *API Gateway Policy Developer Filter Reference*.

Payload Level
You can configure the following settings for the file transfer service payload:

Payload Logging	Description
On receive request from client	Log the message payload when a request arrives from the client.
On send response to client	Log the message payload before the response is sent back to the client.
On send request to remote server	Log the message payload before the request is sent using any **Connection** or **Connect to URL** filters deployed in policies.
On receive response from remote server	Log the message payload when the response is received using any **Connection** or **Connect to URL** filters deployed in a policies.

For details on how to log message payloads at any point in a specific policy, see "Log message payload" in the *API Gateway Policy Developer Filter Reference*.

Include in real time monitoring
Select whether to monitor traffic for the file transfer service. This means that traffic for this service is monitored in the API Gateway Manager and API Gateway Analytics web-based consoles. This setting is selected by default. For more details, see:

- *API Gateway Administrator Guide*
- *API Gateway Analytics User Guide*

Traffic monitor settings

The **Traffic Monitor** tab enables you to configure traffic monitoring settings for the file transfer service. To override the system-level traffic monitoring settings, select **Override system-level settings**, and configure the relevant options.

For more details on traffic monitoring settings, see the *API Gateway Administrator Guide*.

Configure passive transfer mode

In *active transfer mode*, the client actively supplies an IP address and port number to open the client connection, which is the default mode. In *passive transfer mode*, the client passively waits for the server to supply an IP address and port number for the client to create the connection. To use passive transfer mode, you must configure the passive transfer properties in the following file:

```
INSTALL_DIR/apigateway/system/conf/jvm.xml
```

The passive transfer properties include the port range, address, and external address, for example:

```
<SystemProperty name="ftpPassivePorts" value="10000-11000"/>
<SystemProperty name="ftpPassiveAddress" value="<internal-ip>"/>
<SystemProperty name="ftpPassiveExternalAddress" value="<external-ip>"/>
```

These properties are explained as follows:

Property	Description
`ftpPassivePorts`	Ports or range of ports on which the server accepts passive data connections (for example, `123`, or `123,124`, or `123-126`).
`ftpPassiveAddress`	IP address on which the server listens for passive data connections.
`ftpPassiveExternalAddress`	IP address on which the server claims to be listening on in the `PASV` reply. For example, this is useful when the server is behind a NAT firewall and the client sees a different server address.

Configure an FTP poller

Overview

The FTP poller enables you to query and retrieve files to be processed by polling a remote file server. When the files are retrieved, they can be passed to the API Gateway core message pipeline for processing. For example, this is useful where an external application drops files on to a remote file server, which can then be validated, modified, and potentially routed on over HTTP or JMS by the API Gateway.

This kind of protocol mediation is useful when integrating with Business-to-Business (B2B) partner destinations or with legacy systems. For example, instead of making drastic changes to either system, the API Gateway can download the files from a remote file server, and route them over HTTP to another back-end system. The added benefit is that messages are exposed to the full complement of API Gateway message processing filters. This ensures that only properly validated messages are routed on to the target system.

The FTP poller supports the following file transfer protocols:

- **FTP**: File Transfer Protocol
- **FTPS**: FTP over Secure Sockets Layer (SSL)
- **SFTP**: Secure Shell (SSH) File Transfer Protocol

To add a new FTP poller, in the Policy Studio tree, under the **Environment Configuration** > **Listeners** node, right-click the instance name (for example, `API Gateway`), and select **FTP Poller** > **Add**. This topic describes how to configure the fields on the **FTP Poller Settings** dialog.

Tip For details on how to configure the API Gateway to act as a file transfer service that listens on a port for remote clients, see *Configure a file transfer service* on page 334.

General settings

This filter includes the following general settings:

Name:
Enter a descriptive name for this FTP poller.

Enable Poller:
Select whether this FTP poller is enabled. This is selected by default.

Host:
Enter the host name of the file transfer server to connect to.

Port:
Enter the port on which to connect to the file transfer server. Defaults to `21`.

User name:
Enter the user name to connect to the file transfer server.

Password:
Specify the password for this user.

Scan settings

The fields configured in the **Scan details** tab determine when to scan, where to scan, and what files to scan:

Poll every (ms):
Specifies how often in milliseconds the API Gateway scans the specified directory for new files. Defaults to `60000`. To optimize performance, it is good practice to poll often to prevent the number of files building up.

Look in directory on FTP server:
Enter the path of the target directory on the FTP server to scan for new files. For example, `outfiles`.

For files that match the pattern:
Specifies to scan only for files based on a pattern in a regular expression. For example, to scan only for files with a particular file extension (for example, `.xml`), enter an appropriate regular expression. Defaults to:

```
([^\s]+(\.(?i)(xml|xhtml|soap|wsdl|asmx))$)
```

Establish new session for each file found:
Select whether to establish a new file transfer session for each file found. This is selected by default.

Limit the number of files to be processed:
Select this option to limit the number of files that the FTP poller can will process on each poll of the FTP server. This option is not selected by default.

Specify the max number of files to be processed:
Enter the maximum number of files to be processed on each poll of the FTP server. The default is `100`.

Process file with following policy:
Click the browse button to select the policy to process each file with. For example, this policy may perform tasks such as validation, threat detection, content filtering, or routing over HTTP or JMS. You can select what action to take after the policy processes the file in the **On Policy Success** and **On Policy Failure** fields.

On Policy Success:
This field enables you to choose the behavior if the policy passes. Select one of the following options:

- Do Nothing — Take no action.
- Delete File — Delete the file.

- Move File — Move the file to a new location. Enter the directory path in the **Move to directory on FTP server** field. This path is relative to the **Look in directory on FTP server** entered above. For example, if **Look in directory** is `outfiles` and **Move to directory** is `processed`, then files are moved to `outfiles/processed` on the FTP server. The **Move to directory** is created if it does not exist.

On Policy Failure:
This field enables you to choose the behavior if the policy fails. Select one of the following options:

- Do Nothing — Take no action.
- Delete File — Delete the file.
- Move File — Move the file to a new location. Enter the directory path in the **Move to directory on FTP server** field. This path is relative to the **Look in directory on FTP server** entered above.

Connection type settings

The fields configured in the **Connection Type** tab determine the type of file transfer connection. Select the connection type from the list:

- FTP — File Transfer Protocol
- FTPS — FTP over SSL
- SFTP — SSH File Transfer Protocol

FTP and FTPS connections

The following general settings apply to FTP and FTPS connections:

Passive transfer mode:
Select this option to prevent problems caused by opening outgoing ports in the firewall relative to the file transfer server (for example, when using *active* FTP connections). This is selected by default.

Note To use passive transfer mode, you must perform the steps described in *Configure passive transfer mode* on page 341.

File Type:
Select **ASCII** mode for sending text-based data or **Binary** mode for sending binary data over the connection. Defaults to **ASCII** mode.

FTPS connections

The following security settings apply to FTPS connections only:

SSL Protocol:
Enter the SSL protocol used (for example, `SSL` or `TLS`). Defaults to `SSL`.

Implicit:
When this option is selected, security is automatically enabled as soon as the FTP poller client makes a connection to the remote file transfer service. No clear text is passed between the client and server at any time. In this case, the client defines a specific port for the remote file transfer service to use for secure connections (`990`). This option is not selected by default.

Explicit:
When this option is selected, the remote file transfer service must explicitly request security from the FTP poller client, and negotiate the required security. If the file transfer service does not request security, the client can allow the file transfer service to continue insecure or refuse and/or limit the connection. This option is selected by default.

Trusted Certificates:
To connect to a remote file server over SSL, you must trust that server's SSL certificate. When you have imported this certificate into the Certificate Store, you can select it on the **Trusted Certificates** tab.

Client Certificates:
If the remote file server requires the FTP poller client to present an SSL certificate to it during the SSL handshake for mutual authentication, you must select this certificate from the list on the **Client Certificates** tab. This certificate must have a private key associated with it that is also stored in the Certificate Store.

SFTP connections

The following security settings apply to SFTP connections only:

Present following key for authentication:
Click the button on the right, and select a previously configured key to be used for authentication from the tree. To add a key, right-click the **Key Pairs** node, and select **Add**. Alternatively, you can import key pairs under the **Environment Configuration** > **Certificates and Keys** node in the Policy Studio tree. For more details, see *Manage X.509 certificates and keys* on page 221.

SFTP host must present key with the following finger print:
Enter the fingerprint of the public key that the SFTP host must present (for example, `43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8`).

Configure a directory scanner

The Directory Scanner enables you to scan a specified directory on the file system for files containing messages (for example, in XML or JSON format). When the messages have been read, they can be passed into the core message pipeline, where the full range of message processing filters can act on them. The Directory Scanner is typically used in cases where an external application is dropping files (perhaps by FTP) on to the file system so that they can be validated, modified, and potentially routed on over HTTP or JMS. Alternatively, they can be stored to another directory where the application can pick them up again.

This sort of protocol mediation is very useful in cases where legacy systems are involved. For example, instead of making drastic changes to the legacy system by adding an HTTP engine, the API Gateway can pull the files from the file system, and route them on over HTTP to another back-end system. The added benefit is that messages can be exposed to the full range of message processing filters in the API Gateway. This ensures that only properly validated messages are routed on to the target system.

To add a new Directory Scanner, in the Policy Studio tree, under the **Environment Configuration** > **Listeners** node, right-click the name of the API Gateway instance (for example, `API Gateway`), and select the **Directory Scanner** > **Add** menu option. This topic describes how to configure the fields on the **Directory Scanner Settings** dialog.

Input settings

The fields in this section configure where to scan for files, what files to scan, and when to scan.

Input Directory:
Enter or browse to the input directory that the API Gateway scans for files. This setting is required.

Match File:
Select one of the following options to specify how inbound files are filtered:

- **By Name**:
  You can specify a comma-separated list of specific names, wildcarded names, or leave this field blank to accept all files by name. For example, a value of `"*.xml,*.xsl"` only accepts XML and XSL files in the input directory.
- **By Extension**:
  You can specify a comma-separated list of file extensions (excluding the `.` character), or leave this field blank to accept all files by extension. For example, `"xml,xsl"` only accepts XML and XSL files in the input directory.
- **By Pattern (regex)**:
  If you wish to scan only for files based on some pattern, you can specify this as a regular expression. For example, if you wish to scan only for files with a particular file extension, such as `.xml` , you can enter a regular expression such as the following:

  ```
  ^[a-zA-Z\s]*.xml
  ```

  Similarly, if a particular naming scheme is used when dropping the files into the configured directory, you can enter a regular expression to scan for these files only. For example, the following regular expression scans only for files named using a `yyyy-mm-dd` date format:

  ```
  ^(\d{4})[- /.]((1[012])|(0?[1-9]))[- /.]((3[01])|([012]?[1-9])|([12]0))$
  ```

Poll Rate (ms):
Specifies the amount of time (in milliseconds) between each scan of the **Input Directory** for new files. Defaults to `60000`.

Processing settings

The fields configured in this section determine what processing is performed on the input files, and where files are placed before and after processing.

Processing Directory:
Enter or browse to the directory to which the input file is copied prior to processing. This field is optional. If it is not specified, the input file is moved to the processing directory specified by the processing policy.

Response Directory:
Enter or browse to the directory to which the response file is copied. This field is optional. If this is not specified, the response file is not written to disk.

Processing Policy:
Select the policy executed on the input file. For example, the policy could perform message validation, routing, virus checking, or XSLT transformation. This setting is required.

File Type:
Specifies how the input file is interpreted. Select one of the following options:

- **Raw**:
  Assumes a content-type of `application/octet-stream`. This is the default.
- **Treat as HTTP Message (including headers)**:
  Assumes the inbound file contains an HTTP request (optionally with HTTP headers).
- **Infer content-type from extension**:
  Performs a lookup on configured MIME types to determine the content-type of the file based on its extension.
- **Use Content-type**:
  Enables you to specify a content-type in the text box.

Sort files:
Select whether files are sorted, and select one of the following sort types from the list:

- `Date last modified`
- `Extension`
- `Extension (case-sensitive)`
- `Name`
- `Name (case-sensitive)`
- `Size`

The default is to sort by `Name`.

Select one of the following sort directions:

- `Ascending`
- `Descending`

The default is `Ascending`.

Process files:
Select whether to process files in sequence or in parallel. Select **In sequence** to process files in a particular order (queued). To process files in parallel, select **In parallel, max workers** and enter the maximum number of threads processing scanned files simultaneously.

To limit the number of files queued for processing during each directory scan, select **Restrict max number of files processed on each run to** and enter the maximum value. The default is 1000.

This option is useful if you need to process very large files, or process files in a particular order (when used in conjunction with the **Sort Files** option).

On completion settings

You can specify what to do when the file processing has completed. Select one of the following options:

- **Do Nothing**:
  The input file remains in the **Input Directory** or **Processing Directory**. This is the default.
- **Delete Input File**:
  The input file is deleted from the **Input Directory** or **Processing Directory**.
- **Move Input File**:
  The input file is moved (archived) to the directory specified in the **To Directory** field. You can also specify an optional **File Prefix** or **File Suffix** for the archived file.

Traffic monitor settings

The **Traffic Monitor** tab enables you to configure traffic monitoring settings for the directory scanner. To override the system-level traffic monitoring settings, select **Override system-level settings**, and configure the relevant options. For more details, see the *API Gateway Administrator Guide*.

Configure a POP client

Overview

The API Gateway POP Client enables you to poll a Post Office Protocol (POP) mail server and read email messages from it. When the messages have been read, they can be passed into the core message pipeline where the full collection of message processing filters can act on them.

Configuration

You can configure a POP client by right-clicking an API Gateway instance node under the **Environment Configuration** > **Listeners** node in the Policy Studio tree, and selecting the **POP Client** > **Add** menu option. Complete the following fields on the **POP Mail Server** dialog:

Server Name:
Enter the host name or IP address of the POP mail server.

Port:
Enter the port on which the POP server is listening. By default, POP servers listen on port 110.

Connection Security:
Select the security used to connect to the POP server (`SSL`, `TLS`, or `NONE`). Defaults to `NONE`.

User Name:
Enter the user name of a configured mail user for this POP server.

Password:
Enter the password for this user.

Poll Rate:
Enter the rate at which the instance polls the mail server in milliseconds.

Delete Message from Server:
Specifies whether the POP server deletes email messages after they have been read by the instance. This setting is selected by default.

Email Debugging:
Select this setting to find out more information about errors encountered by the API Gateway when polling the POP server. All trace files are written to the `/trace` directory of your API Gateway installation. This setting is not selected by default.

Policy to Use:
Select the policy to use to process messages that have been read from the POP server.

TIBCO integration

Overview

The API Gateway ships with in-built support for TIBCO Rendezvous. The API Gateway can both produce and consume messages for TIBCO Rendezvous. This topic describes how to integrate TIBCO Rendezvous.

TIBCO Rendezvous integration

The API Gateway can act as a producer and a consumer of TIBCO Rendezvous messages. In both cases, a Rendezvous daemon must be configured. This is responsible for communicating with other Rendezvous programs on the network.

Producing TIBCO Rendezvous Messages:
You must perform the following steps to produce messages and send them to another Rendezvous program:

1. Configure a Rendezvous daemon. For more information, see *Configure TIBCO Rendezvous daemons* on page 415.
2. Use a filter to route messages to TIBCO Rendezvous. For more information, see the **TIBCO Rendezvous** filter in the *API Gateway Policy Developer Filter Reference*.

Consuming TIBCO Rendezvous Messages:
A TIBCO Rendezvous listener can be configured at the API Gateway instance level to consume Rendezvous messages. You must perform the following steps to consume Rendezvous messages:

1. Configure a Rendezvous daemon. For more information, see *Configure TIBCO Rendezvous daemons* on page 415.
2. Configure a Rendezvous listener. For more information, see *TIBCO Rendezvous listener* on page 350.

TIBCO Rendezvous listener

Overview

TIBCO Rendezvous is a low latency messaging product for real-time high throughput data distribution applications. A message can be sent from the TIBCO daemon running on the local machine to a single TIBCO daemon running on a separate host machine or it can be broadcast to several daemons running on multiple machines. Each message has a subject associated with it, which acts as the *destination* of the message.

A listener, which is itself a TIBCO daemon, can declare an interest in a subject on a specific daemon. Whenever a message is delivered to this subject on the daemon, the message is delivered to the listening daemon.

The API Gateway can act as a listener on a specific subject at a TIBCO daemon, in which case it said to be acting as a *consumer* of TIBCO messages. Similarly, it can also send messages to a TIBCO daemon, effectively acting as a *producer* of messages. For more information on how to send messages to other TIBCO Rendezvous programs, see the **TIBCO Rendezvous** filter in the *API Gateway Policy Developer Filter Reference*.

Configuration

You can configure a TIBCO Rendezvous Listener at the API Gateway instance level in the Policy Studio. To add a listener, perform the following steps:

1. In the Policy Studio tree, expand the **Environment Configuration** > **Listeners** node.
2. Right-click the API Gateway instance, and select **TIBCO** > **Rendezvous Listener** > **Add**.
3. Configure the following fields on the **TIBCO Rendezvous Listener** dialog:
    - **TIBCO Settings tab**:
      Enter the name of the subject that you want this consumer to listen for in the **Rendezvous Subject** field. Only messages addressed with this subject are consumed by the listener.

      Click the button next to the **TIBCO Rendezvous Daemon to use** field, and select a previously configured TIBCO Rendezvous Daemon to communicate with other TIBCO programs. To add a TIBCO Rendezvous Daemon, right-click the **TIBCO Rendezvous Daemons** tree node, and select **Add a TIBCO Rendezvous Daemon**. For more details, see *Configure TIBCO Rendezvous daemons* on page 415.
    - **Policy to Use**:
      When messages with the specified subject have been consumed they must be passed into a policy where they can be processed accordingly. Select the policy that you want to use to process consumed messages from the tree.

Configure Amazon SQS queue listener

Overview

Amazon Simple Queue Service (SQS) is a hosted message queuing service for distributing messages amongst machines. You can configure API Gateway to poll an Amazon SQS queue at a set rate. Any message found on the SQS queue in this interval can be sent to a policy for processing. For more information on Amazon SQS, go to http://aws.amazon.com/sqs/.

To add a new Amazon SQS queue listener, in the Policy Studio tree, under the **Environment Configuration** > **Listeners** node, right-click the instance name (for example, **API Gateway**), and select **Amazon Web Services > Add SQS Queue Listener**.

General settings

You can configure the following general settings for the Amazon SQS queue listener:

Name:
Enter a suitable name for this SQS listener.

AWS settings

AWS Credential:
Click the browse button to select your AWS security credentials (API key and secret) to be used by API Gateway when connecting to Amazon SQS.

Region:
Select the region appropriate for your deployment. You can choose from the following options:

- US East (Northern Virginia)
- US West (Oregon)
- US West (Northern California)
- EU (Ireland)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)
- South America (Sao Paulo)
- US GovCloud

Client settings:
Click the browse button to select the AWS client configuration to be used by API Gateway when connecting to Amazon SQS. For more details, see *Configure AWS client settings* on page 353.

Poll settings

Poll the queue <queueName> every <pollRate> milliseconds:
Enter the name of the queue to be polled and the rate at which the queue is to be polled, in milliseconds. The default queue name is `requestQueue` and the default poll rate is `10000` milliseconds (10 seconds).

Call policy:
Click the browse button to select a policy to send the message to for processing. When a message has been consumed from the queue it is passed to the selected policy for processing.

Process the retrieved messages as body with following Content Type:
Select this option to create a body from the message, and enter a content type. The default is `text/xml`.

Store the content of the retrieved messages in the following attribute:
Select this option to store the content of the message in an attribute, and enter the attribute name. The default attribute name is `sqs.body`.

Delete message on completion:
Select this option to delete the message from the queue when processing is complete. This is selected by default.

Maximum number of messages to read from queue:
Enter the maximum number of messages to return. Amazon SQS never returns more messages than this value but it might return less. The default value is `10`.

Visibility timeout for other consumers:
Enter the duration (in seconds) that the received messages are hidden from subsequent retrieve requests after being retrieved by API Gateway. The default value is `1000` seconds.

Response settings

Write a response message:
Select this option to write the contents of an attribute to a response queue. This is not selected by default.

Response queue name:
If you selected the **Write a response message** option, enter the name of the response queue in this field. The default name is `responseQueue`.

Use content body as the response:
Select this option to use the content body as the response.

Use the following attribute as the response:
Select this option to use an attribute as the response, and enter an attribute selector for the attribute containing the response in this field (for example, `${sqs.body}`). For more details on selectors, see *Select configuration values at runtime* on page 421.

Configure AWS client settings

API Gateway is a client of AWS, and as such you can define a client profile by which API Gateway connects to AWS. When you click the browse button on the **Client settings** field for an Amazon SQS queue listener, a **Send to Amazon SQS** filter, an **Upload to Amazon S3** filter, or an Amazon Simple Notification Service (SNS) alert destination, you can edit the configuration of the AWS client profile.

To edit the configuration, right-click the **Default AWS Client Configuration** and select **Edit**. Configure the following settings on the dialog:

Name:
Enter a suitable name for this client configuration.

Connection settings

Maximum number of open HTTP connections:
Enter the maximum number of open HTTP connections. The default value is `50`.

Socket timeout:
Enter the amount of time to wait (in milliseconds) for data to be transferred over an established, open connection before the connection is timed out. The default value is `50000` milliseconds. A value of 0 means infinity, and is not recommended.

Connection timeout:
Enter the amount of time to wait (in milliseconds) when initially establishing a connection before giving up and timing out. The default value is `50000` milliseconds. A value of 0 means infinity, and is not recommended.

Maximum number of retries:
Enter the maximum number of retry attempts for failed requests (that can be retried). The default value is `3`.

User agent:
Enter the HTTP user agent header to send with all requests.

Protocol:
Select the protocol (HTTP or HTTPS) to use when connecting to AWS.

Proxy settings

You can optionally configure the following proxy settings:

Proxy Host:
Enter the proxy host to connect through.

Proxy port:
Enter the port on the proxy host to connect through.

User name:
Enter the user name to use when connecting through a proxy.

Password:
Enter the password to use when connecting through a proxy.

NTLM Proxy support:
To configure Windows NT LAN Manager (NTLM) proxy support, enter the Windows domain name in the **Windows domain** field and enter the Windows workstation name in the **Windows workstation name** field.

Advanced settings

You can optionally configure these settings to tune low level TCP parameters to try and improve performance.

Note These settings are for advanced users only.

Size hint (in bytes) for the low level TCP send buffer:
Enter the size hint (in bytes) for the low level TCP send buffer.

Size hint (in bytes) for the low level TCP receive buffer:
Enter the size hint (in bytes) for the low level TCP receive buffer.

Further information

For more detailed information on Amazon Web Services integration, see the *AWS Integration Guide* available from Axway Support.

Cryptographic acceleration

Overview

The API Gateway uses OpenSSL to perform cryptographic operations, such as encryption and decryption, signature generation and validation. OpenSSL exposes an *Engine API*, which makes it possible to plug in alternative implementations of some or all of the cryptographic operations implemented by OpenSSL. When configured appropriately, OpenSSL calls the engine's implementation of these operations instead of its own.

For example, a particular engine might provide improved implementations of the asymmetric operations RSA and DSA. This engine can then be plugged into OpenSSL so that whenever OpenSSL needs to perform either an RSA or DSA operation, it calls out to the engine's implementation of these algorithms rather than call its own.

Typically, OpenSSL engines provide a hardware implementation of specific cryptographic operations. The hardware implementation usually offers improved performance over its software-based counterpart, which is known as *cryptographic acceleration*.

You can configure cryptographic acceleration at the instance level in the API Gateway. To configure the API Gateway instance to use an OpenSSL engine instead of the default OpenSSL implementation, perform the following steps:

1. In the Policy Studio tree, expand **Environment Configuration** > **Listeners**.
2. Right-click the API Gateway instance, and select **Cryptographic Acceleration > Add OpenSSL Engine**.

General configuration

The OpenSSL Engine Configuration dialog displays the name of the engine, the algorithms that it implements, together with any initialization and cleanup commands required by the engine. Complete the following fields:

Name:
Enter an appropriate name for the engine in this field.

Provides:
Enter a comma-separated list of cryptographic operations to be performed by the engine instead of OpenSSL. The engine must implement the listed operations, otherwise the default OpenSSL operations are used. The following operations are available:

`RSA`	RSA (Rivest Shamir Adleman) asymmetric algorithm
`DSA`	DSA (Digital Signature Algorithm) asymmetric algorithm
`RAND`	Random number generation
`DH`	Diffie-Hellman anonymous key exchange algorithm
`ALL`	Engine's implementation of all cryptographic algorithms

For example, to configure the API Gateway to use the engine's implementation of the RSA, DSA, and DH algorithms only, enter the following in the **Provides** field:

```
RSA, DSA, DH
```

Commands:
The OpenSSL engine framework allows a number of control commands to be invoked at various stages in the loading and unloading of a specific engine library. These commands can be issued before or after the initialization of the engine, and also before or after the engine is uninitialized. Control commands are based on text name-value pairs.

Typical uses for control commands include specifying the path to a driver library, logging configuration information, a password to access protected devices, a configuration file required by the engine, and so on.

OpenSSL control commands can be added by clicking the **Add** button. Enter the name of the command in the **Name** field, and its value in the **Value** field. This command *must* be supported by the engine.

Use the **When** drop-down list to select when the command is to be run. The options available are as follows:

`preInit`	Command is run before the engine is initialized (before the call to `ENGINE_init()`).
`postInit`	Command is run after the engine is initialized (after the call to `ENGINE_init()`).
`preShutdown`	Command is run before the engine shuts down (before the call to `ENGINE_finish()`).
`postShutdown`	Command is run after the engine shuts down (after the call to `ENGINE_finish()`).

Conversations for crypto engines

A Hardware Security Module (HSM) protects the private keys that it holds using a variety of mechanisms, including physical tokens, passphrases, and other methods. When use of the private key is required by an agent, it must authenticate itself with the HSM, and be authorized to access this data.

For information on how the API Gateway interacts with the HSM, see *Cryptographic acceleration conversation: request-response* on page 357. For more information on storing private keys on OpenSSL engines, see *Manage X.509 certificates and keys* on page 221.

Cryptographic acceleration conversation: request-response

Hardware Security Modules (HSM) protect the private keys they hold using a variety of mechanisms, including physical tokens, passphrases, and other methods. When use of the private key is required by some agent, it must authenticate itself with the HSM, and be authorized to access this data.

Whatever the mechanism protecting the keys on the HSM, this commonly requires some interaction with the agent. The most common form of interaction required is for the agent to present a passphrase. The intent is generally that this is carried out by a real person, rather than produced mechanically by the agent. Other forms of interaction may include prompting the operator to insert a specific card into a card reader.

However, the requirement for an operator to enter a passphrase renders automated startup of services using the HSM impossible. Although weaker from a security standpoint, the server can conduct an automated dialog with a HSM when it requires access to a private key, presenting specific responses to specific requests, including feeding passphrases to it. Of course, this is futile if the dialog calls for the insertion of a physical token in a device.

The dialog for different keys on the same device is often the same. For example, a number of keys on a Thales nShield Solo HSM may require the server to present an operator passphrase for a preinserted card in a card-reader. The specific dialogs are therefore associated with the cryptographic engine.

Each dialog consists of a set of expected request-generated response pairs. The expected request takes the form of a regular expression. When the cryptographic device prompts for input, the text of this prompt is compared against each expected request in the conversation, until a match is found. When matched, the corresponding generated response is delivered to the HSM.

In the simplest case, consider a HSM producing the following prompt:

```
Enter passphrase for operator card Operator1:
```

You can identify this, for example, with the following regular expression:

```
"passphrase.*Operator1"
```

In the configured conversation, you can make the expected response to this prompt the passphrase for the specific card, for example:

```
"tellNoOne"
```

The server is somewhat at the mercy of the HSM for how this dialog continues. If the HSM continues to prompt for requests, the server can only attempt to respond. You may set the maximum expected challenge setting on the conversation to indicate a maximum number of prompts to expect from the HSM, at which point the server does its best to terminate the conversation, almost certainly failing to load the affected key.

External connections 9

This section describes how to configure external connections.

External connections

API Gateway can leverage your existing Identity Management infrastructure, thus avoiding the need to maintain separate silos of user information. For example, if you already have a database full of user credentials, API Gateway can authenticate requests against this database, rather than using its own internal user store. Similarly, API Gateway can authorize users, look up user attributes, and validate certificates against third-party Identity Management servers.

You can add a connection to an external system as a global *external connection* in Policy Studio so that it can be reused across all filters and policies. For example, if you create a policy that authenticates users against an LDAP directory, and then validates an XML signature by retrieving a public key from the same LDAP directory, it makes sense to create a global external connection for that LDAP directory. You can then *select* the LDAP connection in both the authentication and XML signature verification filters, rather than having to reconfigure them in both filters.

You can also use external connections to configure a group of related URLs. This allows you to round-robin between a number of related URLs to ensure high availability. When API Gateway is configured to use a *URL connection set* (instead of a single URL), it round-robins between the URLs in the set.

To configure external connections, right-click the appropriate node in Policy Studio tree (for example, **Environment Configuration > External Connections > Database Connections**). This topic introduces the different types of external connection and shows where to obtain more details.

Authentication repository profiles

API Gateway can authenticate users against external databases and LDAP repositories, in addition to its own local user store. You can also use a number of bespoke authentication connectors to enable API Gateway to authenticate against specific third-party Identity Management products.

Connection details for these authentication repositories are configured at a global level, making them available for use across authentication (and authorization) filters. This saves the administrator from reconfiguring connection details in each filter.

For example, the available authentication repository types include the following:

- CA SiteMinder Repositories
- Database Repositories
- Entrust GetAccess Repositories
- LDAP Repositories
- Local Repositories (for example, Local User Store)
- Oracle Access Manager Repositories
- Oracle Entitlements Server Repositories
- RADIUS Repositories (deprecated)
- RSA Access Manager Repositories
- Tivoli Repositories
- Sun Access Manager Repositories (deprecated)

For details on how to configure the various authentication repository types, see *Configure authentication repositories* on page 365.

Connection sets

Connection sets are used by API Gateway to round-robin between groups of external servers (for example, RSA Access Manager). You can reuse these global groups when configuring connections to external servers in Policy Studio. For this reason, connection sets are available under the **External Connections** node according to the filter from which they are available. For example, connection sets under the **RSA Access Manager Connection Sets** node are available in the **RSA Access Manager** filter.

At runtime, API Gateway can round-robin between the servers in the group to ensure that if one of the servers becomes unavailable, API Gateway can use one of the other servers in the group.

To add a connection set for a particular category of filters, right-click the appropriate node under the **Connection Sets** node under the **External Connections** node. Select **Add a Connection Set** to display the **Connection Group** dialog. For more details, see *Configure connection groups* on page 433.

Client credentials

Client credentials allow you to configure client authentication settings at a global level. You can configure a client credential profile for the following authentication options:

- API keys as a client
- HTTP basic
- Kerberos
- OAuth 2.0 as a client

After configuring a client credential profile globally, you can select that profile for use at the filter level (for example, in the **Connection** or **Connect To URL** filters).

To add a client credential profile for a particular authentication mechanism, right-click the appropriate node under the **Client Credentials** node under the **External Connections** node. For more details, see *Configure client credentials* on page 383.

Database connections

API Gateway typically connects to databases to authenticate or authorize users using API Gateway's numerous Authentication and Authorization filters. Similarly, API Gateway can retrieve user attributes from a database (for example, which can then be used to generate SAML attribute assertions later in the policy).

You can configure database connections globally under the **External Connections** node, making them available to the various filters that require a database connection. This means that an administrator can reuse the same database connection details across multiple authentication, authorization, and attribute-based filters.

API Gateway maintains a JDBC pool of database connections to avoid the overhead of setting up and tearing down connections to service simultaneous requests. This pool is implemented using *Jakarta DBCP (Database Connection Pools)*. The settings in the **Advanced** section of the **Configure Database Connection** dialog are used internally by API Gateway to initialize the connection pool. The table at the end of this section shows how the fields correspond to specific configuration DBCP settings.

To configure details for a global database connection, right-click the **External Connections > Database Connections** node. Select the **Add a Database Connection** menu option, and configure the fields on the **Configure Database Connection** dialog. For details on configuring these fields, see *Configure database connections* on page 387.

ICAP servers

The Internet Content Adaptation Protocol (ICAP) is a lightweight HTTP-based protocol used to optimize proxy servers, which frees up resources and standardizes how features are implemented. For example, ICAP is typically used to implement features such as virus scanning, content filtering, ad insertion, or language translation in the HTTP proxy cache.

When an ICAP Server is configured under the **External Connections** node, you can then select it in multiple ICAP filters. For details on how to configure an ICAP server, see *Configure ICAP servers* on page 392.

Sentinel servers

You can send events from API Gateway to Axway Sentinel. When a Sentinel server is configured under the **External Connections** node, you can then select it in multiple Sentinel monitoring filters. For details on how to configure a Sentinel server, see *Configure Sentinel servers* on page 394.

JMS services

The Java Message Service (JMS) is a Java message-oriented middleware API for sending messages between two or more clients. When a JMS Service is configured under the **External Connections** node, it is available for selection in multiple JMS-related configuration screens. This enables you to share JMS configuration across multiple filters.

For more details on configuring JMS services, see *Configure messaging services* on page 151.

Kerberos connections

You can configure global **Kerberos Clients**, **Kerberos Services**, and **Kerberos Principals** under the **External Connections** node. When a Kerberos item is configured, it is available for selection in all Kerberos-related configuration screens that require this item. This enables you to share Kerberos configuration items across multiple filters.

For more details, see the following topics:

- *Configure Kerberos clients* on page 395
- *Configure Kerberos services* on page 402
- *Configure Kerberos principals* on page 400

See also the *API Gateway Kerberos Integration Guide*.

LDAP connections

In the same way that database connections can be configured globally in Policy Studio (and then reused across individual filters), LDAP connections are also managed globally in Policy Studio. LDAP connections are used by authentication, authorization, and attribute filters. Filters that require a public key (from a public-private key pair) can also retrieve the key from an LDAP source.

When a filter that uses an LDAP directory is run for the first time, it binds to the LDAP directory using the connection details configured on the **Configure LDAP Server** dialog. Usually the connection details include the user name and password of an administrator user who has read access to all users in the LDAP directory for whom you wish to retrieve attributes or authenticate.

For details on how to configure a global LDAP connection, see *Configure LDAP directories* on page 406.

Proxy servers

You can configure proxy servers under the **External Connections** node, which can then be specified in the **Connection** and **Connect To URL** filters. When configured, the filter connects to the proxy server, which routes the message to the destination server.

To configure a proxy server, click the **External Connections** node, and select **Proxy Servers > Add a Proxy Server**. For details on how to configure the settings the **Proxy Server Settings** dialog, see *Configure proxy servers* on page 409.

RADIUS clients

Note This feature has been deprecated and will be removed in a future release.

The Remote Authentication Dial In User Service (RADIUS) protocol provides centralized authentication and authorization for clients connecting to remote services.

To configure a client connection to a remote server over the RADIUS protocol, click the **External Connections** node, and select **RADIUS Clients > Add a RADIUS Client**. For details on how to configure the settings the **RADIUS Client** dialog, see the *Configure RADIUS clients* on page 410.

For details on how to configure a RADIUS Authentication Repository, see *Configure authentication repositories* on page 365.

SiteMinder and SOA Security Manager

To add a CA SiteMinder or CA SOA Security Manager connection, right-click the **SiteMinder/SOA Security Manager** node under the **External Connections** node, and select **Add a SiteMinder Connection** to display the **SiteMinder Connection Details** dialog. For details on configuring the fields on this dialog, see *Configure SiteMinder/SOA Security Manager connections* on page 411.

SMTP servers

You can configure a Simple Mail Transfer Protocol (SMTP) server as a global configuration item under the **External Connections** node. The **SMTP** filter in the **Routing** category can then reference this SMTP server. To configure an SMTP server, right-click the **External Connections > SMTP Servers** node, and select **Add an SMTP Server**. For more details, see *Configure SMTP servers* on page 414.

Syslog servers

You can configure syslog servers globally, and then select them as a customized logging destination for an API Gateway instance. Right-click the **External Connections > Syslog Servers** node, and select **Add a Syslog Server**. Complete the following fields on the **Syslog Server** dialog:

Name:
Enter an appropriate name for the syslog server.

Host:
Enter the host and UDP port on which the syslog daemon is running. Enter in the format of `HOST_OR_IP_ADDRESS:UDP_PORT` (for example, `192.0.2.0:514`). Alternatively, you can enter `HOST_OR_IP_ADDRESS` only, and the default port of `514` is used.

Facility:
Select the syslog facility to log to (for example, `local0`).

Note For details on how to configure the API Gateway instance to enable logging to this remote syslog server, see the *API Gateway Administrator Guide*.

TIBCO

You can add a connection to TIBCO Rendezvous daemon. To add a connection, right-click the **External Connections > TIBCO Rendezvous Daemon** node in the tree, and select **Add a TIBCO Rendezvous Daemon**. For details on configuring the fields on the dialog, see *Configure TIBCO Rendezvous daemons* on page 415.

Tivoli

You can create a connection to an IBM Tivoli server to enable integration between the API Gateway and Tivoli Access Manager for e-business 6.1. Tivoli connections can then be used by API Gateway's Tivoli filter to delegate authentication and authorization decisions to Tivoli Access Manager, and to leverage existing Tivoli Access Manager policies.

To add a Tivoli connection, right-click the **External Connections > Tivoli Connections** node in the tree, and select **Add a Tivoli Connection**. For more details on configuring the fields on the **Tivoli Configuration** dialog, see *Configure Tivoli connections* on page 417.

URL connection sets

URL connection sets are used by API Gateway filters to round-robin between groups of external servers (for example, Entrust GetAccess, SAML PDP, or XKMS). These global groups can then be reused when configuring these filters in Policy Studio. For this reason, URL connection sets are available under the **External Connections** node in the tree, according to the filters from which they are available. For example, URL sets under the **XKMS URL Sets** node are only available from the **XKMS Certificate Validation** filter.

At runtime, API Gateway can round-robin between the servers in the group to ensure that if one of the servers becomes unavailable, API Gateway can use one of the other servers in the group.

To add a URL connection set for a particular category of filters, right-click the appropriate node under the **External Connections > URL Connection Sets** node in the tree. Select the **Add a URL Set** option to display the **URL Group** dialog. For more details, see *Configure URL groups* on page 443.

XKMS connections

API Gateway can also validate certificates against an XKMS (XML Key Management Service) responder or group of responders. An XKMS consists of a group of XKMS responders to validate certificates against, coupled with the signing key to use for signing requests to each of the responders in the group.

To add a global XKMS connection, right-click the **External Connections > XKMS Connection** node in the tree, and select the **Add an XKMS Connection** option to display the **Certificate Validation - XKMS** dialog. For more details, see *Configure XKMS connections* on page 418.

All global XKMS Connections are available for selection when configuring the **Certificate Validation - XKMS** filter. This saves the administrator from reconfiguring XKMS connection details across multiple filters.

Configure authentication repositories

API Gateway supports a wide range of common authentication schemes, including SSL, XML signatures, WS-Security Username tokens, and HTTP authentication. With SSL, the client authenticates to API Gateway using a client certificate. With XML signatures, the client is authenticated by validating the signature contained within the XML message. However, when API Gateway attempts to authenticate a client using a user name and password (for example, WS-Security Username tokens or HTTP authentication), it must compare the user name and password presented by the client to those stored in the *authentication repository*.

The authentication repository acts as a repository for *users*. Users serve many roles in API Gateway. For example, clients whose user name and password combinations are stored in the authentication repository can authenticate to API Gateway using that user name and password combination. For more information on users, see *Manage API Gateway users* on page 233.

The authentication repository can be maintained in API Gateway's local configuration store, in an LDAP directory, or in a range of third-party Identity Management products and services. When a user has been successfully authenticated against one of these repositories, API Gateway can use any one of that user's stored attributes (for example, distinguished name (DName), email address, user name) to authorize that same user in a subsequent authorization filter.

For example, this *credential mapping* is useful in cases where your client-base uses user name and password combinations for authentication (*authentication attributes*), but their access rights must be looked up in an authorization server using the client's DName (*authorization attribute*). In this way, the client possesses a single *virtual identity* within API Gateway. The client can use one identity for authentication, and another for authorization, yet API Gateway sees both identities as representing the same client.

You can add a new repository under the **Environment Configuration > External Connections** node in Policy Studio node tree. Right-click the appropriate node (for example, **Database Repositories**), and select **Add a new Repository**. Similarly, you can edit an existing repository. Right-click the repository node (for example, the default **Local User Store**), and select **Edit Repository**. You can reuse the repositories added under the **External Connections** node in multiple filters.

Axway PassPort repositories

API Gateway can integrate with Axway PassPort to authenticate and authorize users for resources. To authenticate users against an Axway PassPort repository, right-click **Axway PassPort Repositories**, and select **Add a new Repository**.

For more details on configuring a connection to an Axway PassPort repository, see *Configure Axway PassPort authentication repositories* on page 379. For details on integrating with Axway PassPort using an authorization filter, see the *API Gateway Policy Developer Filter Reference*.

CA SiteMinder repositories

If the user profiles are stored in an existing CA SiteMinder server, API Gateway can query SiteMinder to authenticate users.

To authenticate users against a SiteMinder repository, right-click **CA SiteMinder Repositories**, and select **Add a new Repository**. Complete the following fields on the Authentication Repository dialog:

Repository Name:
Enter a suitable name for this repository.

Agent Name:
Select a previously configured SiteMinder agent name from the list. To register a new agent, see *API Gateway Authentication and Authorization Integration Guide*.

Resource:
Enter the name of the protected resource for which the user must be authenticated. Alternatively, you can enter a selector for a message attribute, which is looked up and expanded to a value at runtime. Message attribute selectors have the following format:

```
${message.attribute}
```

For example, by default API Gateway specifies the original path the end user requested as the resource:

```
${http.request.uri}
```

Action:
The user must be authenticated for a specific action on the protected resource. By default, API Gateway takes this action from the HTTP verb used in the incoming request. You can use the following selector to get the HTTP verb:

```
${http.request.verb}
```

Alternatively, you can enter any user-specified value. For more details on selectors, see *Select configuration values at runtime* on page 421.

Single Sign-On Token:
By default, when an end user has been authenticated for a given resource, SiteMinder generates a *single sign-on token* (a session cookie). API Gateway stores this cookie in a user-specified message attribute. API Gateway returns the cookie to the end user along with the response. The end user can then pass this cookie with future requests to API Gateway. When API Gateway receives such a request, it can validate the session cookie using the **CA SiteMinder Session Validation** filter. The client stays authenticated for the entire lifetime of the session cookie. As long as the session cookie is valid, API Gateway does not need to re-authenticate the end user against SiteMinder for every request. This increases throughput and performance considerably.

Put Token in Message Attribute:
Enter the name of the message attribute where you wish to store the session cookie. By default, the cookie is stored in the `siteminder.session` attribute.

For more details how to integrate API Gateway with SiteMinder, see *API Gateway Authentication and Authorization Integration Guide*.

Database repositories

API Gateway can store its authentication repository in an external database. This option makes sense if your organization already has a silo of user profiles stored in the database and you do not want to duplicate this store within API Gateway's local configuration storage.

To authenticate users against a database repository, right-click **Database Repositories**, and select **Add a new Repository**. Complete the following fields on the Authentication Repository dialog:

Repository Name:
Enter an appropriate name for the database in the **Repository Name** field.

Database:
There are two basic configuration items required to retrieve a user's profile from the database:

- **Database Location**:
  To configure connection details for the database, click **Add**, and complete the Database Connection dialog. For details on configuring the fields on this dialog, see *Configure database connections* on page 387. To edit or remove previously configured database connections, select them from the drop-down list and click **Edit** or **Delete**.
- **Database Query**:
  Database Query retrieves the profile of a specific user from the database to enable API Gateway to authenticate the user. after a successfull authentication, you can select an attribute of this user to use for the authorization filter later in the policy. **Database Query** can be an SQL statement, stored procedure, or function call. For details on how to configure **Database Query**, see *Configure database queries* on page 390.

Format Password Received From Client:
If the user sends up a clear-text password to API Gateway, but that user's password is stored in a hashed format in the database, API Gateway must hash the password before performing the authentication step.

- **Hash Client Password**:
  Depending on whether you wish to hash the user's submitted password, select the appropriate option.
- **Hash Format**:
  If you have selected to hash the client's password, you must specify the format of the hashed password. The most typical formats have been listed, but you can also enter another format. Formats should be entered in terms of message attribute selectors. The following formats are available from the **Hash Format** list.

  ```
  ${authentication.subject.id}:${authentication.subject.realm}:${authentication.subject.password}
  ${authentication.subject.password}
  ```

  The first option combines the username, authentication realm, and password respectively. This combination is then hashed. The second option simply creates a hash of the user's password.
- **Hash Algorithm**:
  Select either **MD5** or **SHA1** as the digest algorithm to use when creating the hash.

For more details on selectors, see *Select configuration values at runtime* on page 421.

Query Result Processing:
You can use these options to provide API Gateway with some meta information about the result **Database Query** returns. You can identify the name of the database table column or row that contains the user's password, as well as the name of the column or row that contains the attribute that is to be used for the authorization filter.

- **Password Column**:
  Specify the name of the database table column that contains the user's password. The contents of this column are compared to the password the user submits.
- **Password Type**:
  Depending on how the user's password has been stored in the database, select either **Clear Password** or **Digest Password**.

- **Authorization Attribute Column**:
  When **Database Query** is run, all of the user's attributes are returned. Only the user's user name and password are used for the authentication event. In addition, you can use one of the other user's attributes for authorization at a later stage in the policy. The additional authorization attribute must be either a user name or an X.509 DName. Enter the name of the column containing the user name or the DName here, but only if this value is required for authorization purposes.
- **Authorization Attribute Format**:
  API Gateway's authorization filters are all based on a user name or DName. All authorization filters evaluate if a user identified by a user name or DName is allowed to access a specific resource. Select the appropriate format from the drop-down list depending on what type of user credential is stored in the database table column entered above.

Entrust GetAccess repositories

Entrust GetAccess provides Identity Management and access control services for web resources. It centrally manages access to web applications, enabling users to benefit from a single sign-on capability when accessing the applications that they are authorized to use.

You can configure API Gateway to connect to a group of GetAccess servers in a round-robin fashion. This provides the necessary failover capability if one or more GetAccess servers are not available. When API Gateway successfully authenticates to a GetAccess server, it obtains authorization information about the end user from the GetAccess SAML PDP. The authorization details are returned in a SAML authorization assertion thatAPI Gateway then validates to determine whether the request should be allowed.

To authenticate users against an Entrust GetAccess repository, right-click **Entrust GetAccess Repositories**, and select **Add a new Repository**. Configure the following fields on the **Authentication Repository** dialog:

Repository Name:
Enter an appropriate name for this repository.

Request:
Configure the following request settings:

- **URL Group**:
  Select a URL group from the drop-down list. This group consists of a number of GetAccess Servers to which API Gateway round-robins connection attempts. To add URL groups, in the node tree, click **Environment Configuration > External Connections > URL Connection Sets**, right-click **Entrust GetAccess URL Sets**, and select **Add a URL Set**. For more details on adding and editing URL groups, see *Configure URL groups* on page 443.
- **WS-Trust Attribute Field Name**:
  Specify the field name for the `Id` field in the WS-Trust request. The default is `Id`.

Response:
Configure the following response settings:

- **SOAP Actor/Role**:
  To add the SAML authorization assertion to the response message, select a SOAP actor/role to

indicate the WS-Security block where the assertion is added. If you leave this field blank, the assertion is not added to the message.

- **Drift Time**:
  The specified time is used to allow for the possible difference between the time on the GetAccess SAML PDP and the time on the machine hosting API Gateway. This comes into effect when validating the SAML authorization assertion.

For details on using an authorization filter to integrate API Gateway with Entrust GetAccess, see the *API Gateway Policy Developer Filter Reference*.

Local repositories

The authentication repository can be maintained in the same database that API Gateway uses to store all its configuration information. To edit the default user store, select **Local Repositories > Local User Store > Edit Repository**. Alternatively, to create a new user store, select **Local Repositories > Add a new Repository**.

You can enter an appropriate name for the repository in the **Repository Name** field. The **Authorization Attribute Format** field enables administrators to specify whether to use the client's **X.509 Distinguished Name** or **User Name** in subsequent authorization filters. If **User Name** is selected, the user name the client uses to authenticate to API Gateway is used in any configured authorization filters. If **X.509 Distinguished Name** is selected, the X.509 DName API Gateway has stored for that user is used for subsequent authorization.

For example, if the administrator selects **User Name** from the **Authorization Attribute Format** list, `admin` (the **User Name** field) is used for authorization. Alternatively, if the administrator selects **X.509 Distinguished Name**, the X.509 DName is used for authorization (for example, `O=Company, OU=comp, EMAIL=emp@company.com, CN=emp`).

For more information on adding and configuring users to the authentication repository, see *Manage API Gateway users* on page 233.

LDAP repositories

In cases where an organization stores user profiles in an LDAP directory, it does not make sense to re-enter these profiles into the default API Gateway user store. Instead, API Gateway can leverage an existing LDAP directory by querying it for user profile data. If a user's profile can be retrieved, and you can bind to the LDAP directory as that user, the user is authenticated.

Authentication with LDAP

When a filter is configured to authenticate a user against an LDAP repository using a user name and password combination, the flow is as follows:

1. A pooled LDAP connection to the repository selected in the **LDAP Directory** field is retrieved.
2. A search filter is run using the retrieved connection (for example, `(&(objectClass={User})(sAMAccountName={c05vc}))`). Attributes configured in the **Login Authentication Attribute** and **Authorization Attribute** fields are retrieved in this search.

   For example, if you select `Distinguished Name` from the drop-down list, the user's DName is retrieved from the LDAP directory. This uniquely identifies the user in the LDAP directory, and is used to bind to the directory so the user's password can be verified. The attribute specified in **Login Authentication Attribute** is used when you bind as any user. The value of the attribute specified in **Authorization Attribute** is stored in `authentication.subject.id`, and can be used by subsequent filters in the policy (for example, Authorization filters that authorize the authenticated user).
3. If no results are returned from the search, the user is not found in the directory. It is important that the administrator user configured on the **Configure LDAP Server** window has the ability to see the user that you are attempting to authenticate.
4. If multiple users are returned from the search, an attempt is made to bind to the directory using each **Login Authentication Attribute** value retrieved from the search, together with the password from the message.
5. If more than one user is authenticated correctly, an error is returned because you only want to authenticate a single user.
6. If no user is authenticated, an error is returned.
7. If a single user's **Login Authentication Attribute** value and password binds successfully to the directory, authentication has succeeded.
8. Any successful bind is immediately closed.

Create an LDAP repository

To create a new LDAP repository, right-click **LDAP Repositories**, and select **Add a new Repository**. The details entered on the Authentication Repository dialog depend on the type of LDAP directory that you are using. Policy Studio has default entries for some of the more common LDAP directories, which are available from the drop-down lists. However, you can also connect to alternative LDAP directories.

The following subsections demonstrate how to configure this window for typical user searches on three common LDAP servers:

- Oracle Directory Server
- Microsoft Active Directory Server
- IBM Directory Server

Oracle Directory Server

To configure the Authentication Repository dialog for Oracle Directory Server (formerly iPlanet and Sun Directory Server), use the following settings:

- **Repository Name**:
  Enter a suitable name for this user store.
- **Directory Name**:
  Click **Add/Edit** to add details of your Oracle Directory Server. For more details, see *Configure LDAP directories* on page 406.

The **User Search Conditions** section instructs API Gateway to search the LDAP tree according to the following conditions:

- **Base Criteria**:
  Enter where API Gateway should begin searching the LDAP directory (for example, `cn=Users, dc=qa, dc=vordel, dc=com`).
- **User Class**:
  Enter or select the name given by the particular LDAP directory to the *User* class. For Oracle Directory Server, select `'inetorgperson' LDAP Class`.
- **User Search Attribute**:
  The value entered depends on the type of LDAP directory to which you are connecting. When a user is stored in an LDAP directory, a number of user *attributes* are stored with that user. One of these attributes corresponds to the user name presented by the client for authentication. However, different LDAP directories use different names for that user attribute. For Oracle Directory Server, select `cn` from the drop-down list.
- **Allow Blank Passwords**:
  Select this to allow the use of blank passwords.

In the next section, you must specify the following:

- **Login Authentication Attribute**:
  In an LDAP directory tree, there must be one user attribute that uniquely distinguishes any one user from all the others. In Oracle Directory Server, the Distinguished name is referred to as the *entrydn* or Entry Domain Name. Select `Entry Domain Name` to uniquely identify the client authenticating to API Gateway.
- **Authorization Attribute**:
  When the client has been successfully authenticated, you can use any one of that user's stored attributes in a subsequent authorization filter. In this case, you want to use the user's Entry Domain Name (DName) for an authorization filter, so enter `entrydn` in the text box. However, you can enter any user attribute as long as the subsequent authorization filter supports it. The value of the LDAP attribute specified is stored in the `authentication.subject.id` message attribute.
- **Authorization Attribute Format**:
  Because any user attribute can be specified in the **Authorization Attribute** above, you must inform API Gateway of the type of this attribute. API Gateway uses this information internally in subsequent authorization filters. Select `X.509 Distinguished Name` from the drop-down list.

Microsoft Active Directory Server

This subsection describes how to configure the Authentication Repository dialog for Microsoft Active Directory Server. The values enter here differ from those entered when interfacing to the Oracle Directory Server:

- **Repository Name**:
  Enter a suitable name for this search.
- **LDAP Directory**:
  Click **Add/Edit** to add details of your Active Directory Server. For more details, see *Configure LDAP directories* on page 406.

The **User Search Conditions** instruct API Gateway to search the LDAP tree according to certain criteria. The values specified are different from those selected for Oracle Directory Server, because MS Active Directory Server uses different attributes and classes to Oracle Directory Server:

- **Base Criteria**:
  The base criteria specify the base object under which to search for the user's profile (for example, `cn=Users, dc=qa, dc=vordel, dc=com`).
- **User Class**:
  In Active Directory Server, the user class is called *User*, so select `'User' LDAP Class`.
- **User Search Attribute**:
  This specifies the name of the user attribute whose value corresponds to the user name entered by the client during a successful authentication process. With Active Directory Server, this attribute is called *givenName*, which represents the name of the user. Enter `givenName` in this field.
- **Login Authentication Attribute**:
  Enter the name of the user attribute that uniquely identifies the user in the LDAP directory. This attribute is the DName and is called *distinguishedName* in Active Directory Server. Select `Distinguished Name` from the drop-down list to uniquely identify the user. API Gateway authenticates the user name and password presented by the client against the values stored for the user identified in this field.
- **Authorization Attribute**:
  When the client has been successfully authenticated, API Gateway can use any of that user's stored attributes in subsequent authorization filters. Because most authorization filters require a DName, enter `Distinguished Name` in the text box. However, any user attribute could be entered here, as long as the subsequent authorization filter supports it.
- **Authorization Attribute Format**:
  API Gateway needs to know the format of the **Authorization Attribute**. Select `X.509 Distinguished Name` from the drop-down list.

IBM Directory Server

The configuration details for IBM Directory Server provide an example of a directory server that does not return a full DName as the result of a standard LDAP user search. Instead, it returns a *contextualized DName*, which is relative to the specified **Base Criteria**. In such cases, API Gateway can build up the full DName by combining the **Base Criteria** and the returned name. The following example shows how this works in practice.

If `C=IE` is specified as the **Base Criteria**, the IBM Directory Server returns `CN=niall, OU=Dev`, instead of the full DName, which is `C=IE, CN=niall, OU=Dev`. To enable API Gateway to do this, leave the **Login Authentication Attribute** field blank. API Gateway then automatically concatenates the specified **Base Criteria** (`C=IE`) with the contextualized DName returned from the directory server (`CN=niall, OU=Dev`) to obtain the fully qualified DName (`C=IE, CN=niall, OU=Dev`).

You can also leave the **Authorization Attribute** field blank to enable API Gateway to automatically use the fully qualified DName for subsequent authorization filters. You should select `X.509 Distinguished Name` from the **Authorization Attribute Format** drop-down list.

Oracle Access Manager repositories

You can authorize an authenticated user for a particular resource against an Oracle Access Manager (OAM) repository. After successful authentication, OAM issues a Single Sign On (SSO) token, which can then be used instead of the user name and password.

To authenticate users against an OAM repository, right-click **Oracle Access Manager Repositories**, and select **Add a new Repository**. Configure the following fields on the Authentication Repository dialog:

Repository Name:
Enter an appropriate name for this repository.

Resource Request:
Configure the following settings for the resource request:

- **Resource Type**:
  Enter the type of the resource for which you are requesting access. For example, for access to a Web-based URL, enter `http`.
- **Resource Name**:
  Enter the name of the resource for which the user is requesting access. By default, this field is set to `//hostname${http.request.uri}`, which contains the original path requested by the client.
- **Operation**:
  In most access management products, users are authorized for a limited set of actions on the requested resource. For example, users with management roles may be permitted to write (`HTTP POST`) to a certain web service, but users with junior roles might only have read access (`HTTP GET`) to the same service. Use this field to specify the operation to which you want to

grant the user access on the specified resource. By default, this is set to the `http.request.verb` message attribute, which contains the HTTP verb used by the client to send the message to API Gateway (for example, `HTTP POST`).

- **Include query string**:
  Select whether the query string parameters are used by the OAM server to determine the policy that protects this resource. This setting is optional if the policies configured do not rely on the query string parameters.
- **Client location**:
  If the client location must be passed to OAM for it to make its decision, you can enter a valid DNS name or IP address to specify this location.
- **Optional Parameters**:
  You can add optional additional parameters to be used in the authentication decision. The available optional parameters include the following:

`ip`	IP address, in dotted decimal notation, of the client accessing the resource.
`operation`	Operation attempted on the resource (for HTTP resources, one of `GET`, `POST`, `PUT`, `HEAD`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`, or `OTHER`).
`resource`	The requested resource identifier (for HTTP resources, the full URL).
`targethost`	The host (`host:port`) to which resource request is sent.

Note One or more of these optional parameters may be required by certain authentication schemes, modules, or plugins configured in the OAM server. To determine which parameters to add, see your OAM server configuration and documentation.

Single Sign On:
Configure the following settings for single sign on:

- **Create SSO Token**:
  Select whether to create an SSO token. This is selected by default.
- **Store SSO Token in User Attribute**:
  Enter the name of the message attribute that contains the user's SSO token. This attribute is populated when authenticating to OAM using the HTTP Basic or HTTP Digest filter (see "HTTP basic authentication" in the *API Gateway Policy Developer Filter Reference* and "HTTP digest authentication" in the *API Gateway Policy Developer Filter Reference*). By default, the SSO token is stored in the `oracle.sso.token` message attribute.
- **Add SSO Token to User Attributes**:
  Select whether to add the SSO Token to user message attributes. This is selected by default.

OAM Access Server SDK Directory:
Enter the path to your OAM Access Server SDK directory. For more details on the OAM Access Server SDK, see your OAM documentation.

For more details on using filters to integrate API Gateway with Oracle Access Manager, see the *API Gateway Policy Developer Filter Reference*.

Oracle Entitlements Server 10g repositories

You can authenticate and authorize a user for a particular resource against an Oracle Entitlements Server (OES) 10g repository.

For example, API Gateway can extract credentials from the message sent by the client, and delegate authentication to OES 10g. When the client has been authenticated, API Gateway queries OES 10g to see if the client is permitted to access the web service resource. When authentication and authorization have passed, the message is trusted and forwarded to the target web service.

To authenticate and authorize users against an OES 10g repository, right-click **Oracle Entitlements Server 10g Repositories**, and select **Add a new Repository**. Configure the following fields on the Authentication Repository dialog:

Repository Name:
Enter an appropriate name for this repository.

Oracle SSM Settings:
Click **Configure** to launch the **Oracle Security Service Module Settings** dialog. For details on configuring these settings, see *Oracle Security Service Module settings (10g)* on page 200.

RADIUS repositories

Note This feature has been deprecated and will be removed in a future release.

You can configure API Gateway to authenticate users in a Remote Authentication Dial In User Service (RADIUS) repository. RADIUS is a client-server network protocol that provides centralized authentication and authorization for clients connecting to remote services.

To authenticate users against a RADIUS repository, perform the following steps:

1. Right-click **RADIUS Repositories**, and select **Add a new Repository**.
2. In the Authentication Repository dialog, enter the RADIUS **Repository Name**.
3. On the **Client** tab, select the RADIUS clients that you wish to authenticate to the repository. For details on how to add clients to this list, see *Configure RADIUS clients* on page 410.
4. On the **Attributes** tab, click **Add** to add a RADIUS attribute. This is a name-value pair used to determine how access is granted. Examples include `User-Name`, `User-Password`, `NAS-IP-Address`, or `NAS-Port`).
5. In the RADIUS Attributes dialog, enter a **Name** for the attribute (for example, `User-Name`). You can select standard RADIUS attributes from the drop-down list, or enter a custom attribute.
6. Enter a **Value** for the attribute, and click **OK**.
7. Click **OK** in the Authentication Repository dialog when you have finished adding attributes.

RSA Access Manager repositories

RSA Access Manager (formerly known as RSA ClearTrust) provides Identity Management and access control services for web applications. It centrally manages access to web applications, ensuring that only authorized users are allowed access to resources. Integration with RSA Access Manager requires RSA Access Manager SDK version 6.2, or server installation libraries.

To authenticate users against an RSA Access Manager repository, right-click **RSA Access Manager Repositories**, and select **Add a new Repository**. Configure the following fields on the Authentication Repository dialog:

Repository Name:
Enter an appropriate name for this repository.

Connection Details:
API Gateway can connect to a group of Access Manager *authorization servers* or *dispatcher servers*. When multiple Access Manager authorization servers are deployed for load-balancing purposes, API Gateway first connects to a dispatcher server, which returns a list of active authorization servers. An attempt is made to connect to one of these authorization servers using round-robin DNS. If the first dispatcher server in the connection group is not available, API Gateway attempts to connect to the dispatcher server with the next highest priority in the group, and so on.

If a dispatcher server has not been deployed, API Gateway can connect directly to an authorization server. If the authorization server with the highest priority in the connection group is not available, API Gateway attempts to connect to the authorization server with the next highest priority, and so on. You can select the type of the connection group using the **Authorization Server** or **Dispatcher Server** radio button. All servers in the group must be of the same type.

Connection Group:
Select the **Connection Group** to use for authenticating clients. You can add connection groups under the **Environment Configuration > External Connections** node in Policy Studio. Expand the **Connection Sets** node, right-click **RSA Access Manager Connection Sets**, and select **Add a Connection Set**. For more details on adding and editing connection groups, see *Configure connection groups* on page 433.

Authentication Type:
Select one of the following authentication types for the connection:

- HTTP Basic
- Windows NT
- RSA SecureID
- LDAP
- Certificate Distinguished Name

For more details on prerequisites and on using an authorization filter to integrate API Gateway with RSA Access Manager, see the *API Gateway Policy Developer Filter Reference*.

Sun Access Manager repositories

Note This feature has been deprecated and will be removed in a future release. See *Oracle Access Manager repositories* on page 374 instead.

API Gateway can authenticate and authorize users against Sun Access Manager using *Access Manager Repositories*. When a user has been successfully authenticated, they receive a Single Sign On (SSO) token, which they can use to obtain access to resources that are protected by Access Manager.

To authenticate users against a Sun Access Manager repository, right-click **Sun Access Manager Repositories**, and select **Add a new Repository**. Configure the following fields on the Authentication Repository dialog:

Repository Name:
Enter a descriptive name for this repository.

Login Module Name:
Enter the name of the Access Manager *Module Instance* that is responsible for logging in users. The module entered must be registered with the **Realm** specified below.

Realm:
Specify the realm that the Access Manager *Module Instance* specified above belongs to.

Create SSO Token:
Select this check box if you want Sun Access Manager to create an SSO (Single Sign-On) token when a user has successfully authenticated. The user can then use this SSO token to gain access to other resources that are protected by Access Manager.

Store SSO Token in Attribute Named:
Enter the name of API Gateway message attribute in which to store the SSO token. Most of API Gateway's Access Manager integration filters require the name of this attribute to retrieve the SSO token and process it.

Add SSO Token to User Attributes:
If this option is selected, the SSO token that is retrieved after successfully authenticating to Sun Access Manager is stored in the `attribute.lookup.list` message attribute, which contains a list of user attributes.

Axway does not recommend selecting this option if you wish to insert a SAML attribute statement into the message (using one of the SAML injection filters) later in the policy. If this option is unselected, the SSO token is inserted into the SAML attribute statement, which may not always be desirable. For this reason, the decision is left to the user on whether or not to add the SSO token to the `attribute.lookup.list` of user attributes depending on their specific requirements.

Tivoli repositories

API Gateway can integrate with Tivoli Access Manager to authenticate users. To authenticate users against a Tivoli repository, right-click **Tivoli Repositories**, and select **Add a new Repository**.

For more details on how to configure API Gateway to communicate with a Tivoli server, see the *API Gateway Authentication and Authorization Integration Guide*.

Configure Axway PassPort authentication repositories

Overview

Axway PassPort provides a central repository, identity broker, and security audit point for your Axway Business-to-Business Integration (B2Bi) or Managed File Transfer (MFT) solutions. Axway PassPort centralizes and simplifies provisioning and management for your entire online ecosystem, enabling secure collaboration between applications, divisions, customers, suppliers, and regulatory bodies.

To authenticate users against an Axway PassPort repository, in the Policy Studio tree, select **Environment Configuration** > **External Connections** > **Authentication Repository Profiles**. Right-click **Axway PassPort Repositories**, and select **Add a new Repository**. This topic explains how to configure the Axway PassPort repository settings, and provides details on Axway PassPort repository registration.

Configuration

Complete the following fields to configure an Axway PassPort repository:

Repository Name:
Enter a descriptive name for this repository.

Hostname:
Enter the host name or IP address of the server running PassPort.

Shared Secret:
Enter the PassPort shared secret. This is specified during PassPort installation.

CSD Name:
Enter the name of the Axway Component Security Descriptor (CSD) file to use when registering with PassPort. Defaults to `csd.xml`.

Note The CSD file must be deployed in the API Gateway group's `conf` directory in your API Gateway installation:

```
INSTALL_DIR/apigateway/groups/GROUP_ID/conf
```

PassPort Certificates—HTTPS:
Select the certificate used for PassPort SSL communication. To export this certificate from PassPort, perform the following steps:

1. In the PassPort user interface, click **Administration** >**Server Security Settings**.
2. Note the certificate used for **Default_HTTPS**.
3. Click **Security** >**Certificates**.
4. Select the certificate noted in step 2 (defaults to `CN=PassPortSecured,O=Axway,C=FR`), and click **Export Certificate**.
5. In the **Export Certificate** dialog, select a **File Extension** of `.cer`.
6. Click **OK**, and select a location to save the certificate.

To import this certificate into the API Gateway, perform the following steps:

1. In the API Gateway **Authentication Repository** dialog, click **Select**.
2. In the **Select Certificate** dialog, click **Create/Import**.
3. In the **Configure Certificate and Private Key** dialog, click **Import Certificate**.
4. Select the certificate that was exported from PassPort.
5. Give the certificate an **Alias Name** manually, or click **Use Subject**.
6. Click **OK**.
7. Select the certificate from the list, and click **OK**.

PassPort Certificates—HTTPS Client Authentication (Optional):
You can configure PassPort to use a different certificate for its client authentication protocol. To do this, repeat the steps for the HTTPS certificate, except when exporting from PassPort in step 2, make a note of the **Default_HTTPS_Client_Auth** certificate.

Ports—HTTPS:
Enter the HTTPS port that PassPort is using. In PassPort, this is found under **Administration** > **Server Ports Configuration**. Defaults to `6453`.

Ports—HTTPS Client Authentication:
Enter the HTTPS client authentication port that PassPort is using. In PassPort, this is found under **Administration** > **Server Ports Configuration**. Defaults to `6666`.

Authentication—Domain:
Enter the PassPort domain that this repository is using for authentication and authorization. Defaults to `Synchrony`.

Axway PassPort repository registration

The external connection to Axway PassPort requires that communication with the Axway PassPort server is performed over a secure connection using two-way SSL authentication. This means that the PassPort server must be able to identify and trust the client connection, and this trust is established by registration.

The first connection from the API Gateway to PassPort initiates registration. A public-private key pair is created and a Certificate Signing Request (CSR) is submitted to PassPort. This is where the

Shared Secret and **HTTPS** port values are used. While the CSR is pending, the repository is unable to process any requests. However, registration is a once off event, and when complete, it does not need to be repeated.

When registration is complete, the key and signed certificate are stored in a Java Key Store file in the following directory of your API Gateway installation:

```
INSTALL_DIR/apigateway/groups/GROUP_ID/conf
```

Troubleshooting registration issues

In the unlikely event that automatic registration fails, you should check the following:

- Ensure the time on the API Gateway is synchronized with the time on the PassPort machine. When PassPort processes the CSR, it sets the **Valid From** date to the current time. If the PassPort time is ahead of the API Gateway time, the API Gateway is unable to use the certificate because it is not yet valid. The error in the trace log is as follows:

  ```
  java.security.cert.CertificateNotYetValidException:java.security.cert.Certificat
  eNotYetValidException
  ```

- By default, PassPort blocks for up to 2 seconds waiting for the CSR to be processed. You can configure this value in the PassPort administration user interface under **Administration** > **System Properties** (`am.registration.cert.signature.wait.time`). If the signing request takes longer than this, one of the following errors may be logged:

  ```
  Authentication exception when authenticating system:

  com.axway.passport.am.api.v2.service.external.PassportConnectionException:Regist
  ration is still in progress.
  Certificate Signing Request has not yet been validated, please try again later.
  [Status:Waiting Validation]
  Authentication exception when authenticating system:

  com.axway.passport.am.api.v2.service.external.PassportConnectionException:Regist
  ration is still in progress.
  Certificate Signing Request has not yet been signed, please try again later.
  [Status:Waiting Signing]
  ```

  This is generally a transient error that may be generated when the initial registration is in progress. Resubmitting the request should succeed. If the error persists, check in the PassPort administration user interface for the reason why the signing request has been delayed.

- If the registration request has been refused by the PassPort administrator, the following error is displayed:

  ```
  Registration has been refused.
  ```

```
Please contact the PassPort Administrator for further information.
[Status:Validation Refused]
```

- If CSR processing fails for some other reason, the following error is logged:

  ```
  Registration has failed and API Gateway is unable to communicate with PassPort.
  Please contact the PassPort Administrator or try manually re-triggering
  registration.
  [Status:...]
  ```

  To resolve this, contact the PassPort administrator to see why the signing request failed. To retry registration, you need to manually re-trigger registration, as explained in the next subsection.

Retrigger registration manually

When registration is complete, the API Gateway does not repeat the process, the generated Java Key Store (JKS) is used for all subsequent connection attempts. However, if for any reason the key pair in the JKS is no longer trusted by PassPort, or if registration is not being processed, you can trigger the registration procedure manually.

The simplest way to retrigger registration is to change the repository name and redeploy. However, if this is not an option, you can manually remove the keystores.

The format of the JKS file names is as follows:

```
keystore_REPOSITORYNAME_HOSTNAME_HTTPSCLIENTAUTHPORT.jks
```

All non-alphanumerics are replaced with an underscore (_).

For example, given the following repository details:

- **Repository Name**: `PassPort - local`
- **Hostname**: `passport-host`
- **HTTPS Client Authentication**: `6666`

The keystore name is `keystore_PassPort___local_passport_host_6666.jks`.

To trigger registration, perform the following steps:

1. Back up the JKS associated with the Axway PassPort Authentication Repository being reset.
2. Delete this JKS.
3. Restart the API Gateway instance. The deleted file is recreated and registration is initiated.

Note If registration has not been completed when the registration is being retriggered, you must delete the following additional temporary files to ensure clean registration:

```
keystore_REPOSITORYNAME_HOSTNAME_HTTPSCLIENTAUTHPORT.jks.csrid
keystore_REPOSITORYNAME_HOSTNAME_HTTPSCLIENTAUTHPORT.jks.pub
keystore_REPOSITORYNAME_HOSTNAME_HTTPSCLIENTAUTHPORT.jks.key
```

In addition, to avoid future confusion, it is good practice to ensure that all redundant certificates and signing requests are removed from PassPort using the PassPort administration user interface.

Configure client credentials

Overview

Client credentials enable you to globally configure client authentication settings for the following authentication options:

- API keys as a client
- HTTP basic
- Kerberos
- OAuth 2.0 as a client

Note For more information on configuring OAuth 2.0 client credentials, see the *API Gateway OAuth User Guide*.

You can configure settings for client credentials under the **Environment Configuration > External Connections > Client Credentials** node in the Policy Studio tree, which you can then specify at the filter level, for example, in the **Connection** and **Connect To URL** filters.

Configure API key client credential profiles

API key client credential profiles enable you to globally configure authentication settings for API keys as a client. API keys are supplied by client users and applications calling REST APIs to allow the API service provider to track and control how the APIs are used (for example, to meter access and prevent abuse or malicious attack).

To configure API key client credential profiles, you must first configure a provider. The following Amazon Web Services provider configurations are included in an out-of-the-box installation of API Gateway:

- Amazon AWS V2 Signing
- Amazon AWS V4 Signing

Add API keys

To add an API key for an existing API key provider, click an API key client credential node (for example, **Amazon AWS V2 Signing**), and click the **Add** button on the **API Key Credential Profiles** tab of the **API Key Credential Profile** window. Complete the following fields on the **Add API Key** dialog:

Name:
Enter a suitable name for this API key.

API Key ID:
Enter an identifier for this API key. This is the Amazon client ID associated with your Amazon account.

API Key Secret:
Enter a secret for this API key. This is the Amazon secret associated with your Amazon account.

Tip To sort the list view of client credential profiles, click the column heading.

After you have configured your API key client credentials globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see "Connection" in the *API Gateway Policy Developer Filter Reference* and "Connect to URL" in the *API Gateway Policy Developer Filter Reference*.

Add API key providers

To configure a new API key provider, right-click **API Keys**, and select **Add API Key Client Credential**. Complete the following fields on the **API Key Service Provider Configuration** dialog:

Name:
Enter a suitable name for this API key service provider configuration.

Sign the request using:
Select an option to specify how the request is signed. The options are:

- Amazon Access Key Signing V2
- Amazon Access Key Signing V4
- No Signing

Put the API key in:
Select an option to specify where to put the API key in the request and enter a name in the **named** field. The options are:

- Header – The parameter is added to the header of the request.
- Query String/Form Body – If the incoming request is a GET request, the parameter is added to the query string. If the incoming request is a POST request, the parameter is added to the content body.

Note Version 2 of the Amazon Signing API only supports the GET operation and only recognizes a limited set of query string parameters. Any unsupported query string parameters result in a `HTTP 400 Bad Request` response from the V2 Amazon web service.

For example, to put the API key in the header of the request in a field named `AWSKeyId`, choose **Header** and enter `AWSKeyId`.

Tip To change the configuration of an existing API key service provider, click the API key client credential node, and edit the settings on the **API Key Configuration** tab of the **API Key Credential Profile** window.

Configure HTTP basic/digest client credential profiles

A client can authenticate to the API service provider with a user name and password combination using HTTP basic authentication or HTTP digest authentication.

To add a HTTP basic/digest client credential profile, click the **HTTP Basic** node, and click the **Add** button on the **HTTP Basic/Digest Client Credentials** window. Complete the following fields on the **Add HTTP Authentication Profile** dialog:

Profile Name:
Enter a suitable name for this HTTP authentication profile.

Choose Authentication Type:
Select the **Basic** or **Digest** radio button to specify the type of HTTP authentication to use, and enter a user name and password in the **Username** and **Password** fields.

Automatically send credentials:
Select this option to automatically send credentials in the request. This is the equivalent of **Send token with first request** in Kerberos. This option is selected by default.

After you have configured your HTTP client credentials globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see "Connection" in the *API Gateway Policy Developer Filter Reference* and "Connect to URL" in the *API Gateway Policy Developer Filter Reference*.

Configure Kerberos client credential profiles

A Kerberos client can authenticate to a Kerberos service by sending a Kerberos service ticket in the HTTP request to that service.

Note You can also configure the API Gateway to authenticate to a Kerberos service by including the relevant Kerberos tokens inside the XML message. For more details, see "Kerberos client authentication" in the *API Gateway Policy Developer Filter Reference*. For more details on different Kerberos setups with API Gateway, see *API Gateway Kerberos Integration Guide*.

To add a credential profile for a Kerberos client, click **Kerberos**, and select **Add** in the **Kerberos Client Credentials** window.

Configure the following fields on the **Add Kerberos Profile** dialog:

Profile Name:
Enter a suitable name for this Kerberos authentication profile.

Kerberos Client:
Click the **...** button, and select a previously configured Kerberos client from the list. To add a Kerberos client, right-click **Kerberos Clients**, and select **Add Kerberos Client**. You can also add Kerberos clients globally under **Environment Configuration > External Connections** in the node tree.

The selected Kerberos client has two roles. First, it must obtain a Kerberos Ticket Granting Ticket (TGT). Second, it uses this TGT to obtain a service ticket for the **Kerberos Service Principal** selected in this profile.

For more details on configuring Kerberos clients, see *Configure Kerberos clients* on page 395.

Kerberos Service Principal:
Click the **...** button, and select a previously configured Kerberos service principal from the list. To add a Kerberos principal, right-click **Kerberos Principals**, and select **Add Kerberos Principal**. You can also add Kerberos principals under **Environment Configuration > External Connections** in the node tree.

The Kerberos client must obtain a ticket from the Kerberos Ticket Granting Server (TGS) for the selected Kerberos service principal. The TGS grants a ticket for the principal, and the Kerberos client can then send the ticket to the Kerberos service. The principal in the ticket must match the principal of the Kerberos service for the client to be successfully authenticated. A service principal name (SPN) can be used to uniquely identify the service in the Kerberos realm.

For more details on configuring Kerberos principals, see *Configure Kerberos principals* on page 400.

Send token with first request:
In some cases, a Kerberos client might not authenticate (send the `Authorization` HTTP header) to the Kerberos service on first request. The Kerberos service then responds with an HTTP 401 response code, instructing the client to authenticate to the server by sending the `Authorization` header. The Kerberos client sends a second request, this time with the `Authorization` header containing the relevant Kerberos token. Select this option to *always* send the `Authorization` HTTP header containing the Kerberos service ticket on the first request to the Kerberos service. This option is selected by default.

Send body only after establish context:
Select this option if you want the Kerberos client to only send the message body after the context with the Kerberos service has been fully established (the client has mutually authenticated with the service).

Pass when service returns 200 even if context not established:
In rare cases, a Kerberos service might return a `200 OK` response to the initial request from a Kerberos client even though the security context has not yet been fully established. This `200 OK` response might not contain the `WWW-authenticate` HTTP header. Select this option to send the

request with the `Authorization` header to the Kerberos service *even if* the context has not been established. The Kerberos service then decides whether to process the request depending on the status of the security context.

After you have configured your Kerberos client credentials profile globally, you can select the client credential profile to use for authentication on the **Authentication** tab of your filter (for example, in the **Connection** and **Connect To URL** filters). For more information, see "Connection" in the *API Gateway Policy Developer Filter Reference* and "Connect to URL" in the *API Gateway Policy Developer Filter Reference*.

Configure database connections

The details entered on the **Configure Database Connection** dialog specify how the API Gateway connects to the database. The API Gateway maintains a JDBC pool of database connections to avoid the overhead of setting up and tearing down connections to service simultaneous requests. This pool is implemented using *Apache Commons DBCP (Database Connection Pools)*.

The settings in the **Advanced - Connection pool** section of this window configures the database connection pool. For details on how the fields on this window correspond to specific DBCP configuration settings, see the table in *Database connection pool settings* on page 389.

Prerequisites

Before configuring a database connection, you must add the JDBC driver files for your chosen database to your API Gateway and Policy Studio installations. The following sections show how to add the third-party JDBC driver files for your database to API Gateway and Policy Studio.

Add third-party binaries to API Gateway

To add third-party binaries to API Gateway, perform the following steps:

1. Add the binary files as follows:
    - Add `.jar` files to the `INSTALL_DIR/apigateway/ext/lib` directory.
    - Add `.so` files to the `INSTALL_DIR/apigateway/<platform>/lib` directory.
2. Restart API Gateway.

Add third-party binaries to Policy Studio

To add third-party binaries to Policy Studio, perform the following steps:

1. Select **Window > Preferences > Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.

4. Click **OK**.
5. Restart Policy Studio with the `-clean` option. For example:

```
> cd INSTALL_DIR/policystudio/
> policystudio -clean
```

Configure the database connection

Configure the following fields on the **Configure Database Connection** window:

Name:
Enter a name for the database connection in the **Name** field.

URL:
Enter the fully qualified URL of the location of the database. The following table shows examples of database connection URLs, where `reports` is the name of the database and `DB_HOST` is the IP address or host name of the machine on which the database is running:

Database	Example Connection URL
Oracle	`jdbc:oracle:thin:@DB_HOST:1521:reports`
Microsoft SQL Server	`jdbc:sqlserver://DB_HOST:1433;DatabaseName=reports;integratedSecurity=false;`
MySQL/MariaDB	`jdbc:mysql://DB_HOST:3306/reports`
IBM DB2	`jdbc:db2://DB_HOST:50000/reports`

Note You can use the `jdbc:mysql://DB_HOST:3306/reports` URL with both MySQL and MariaDB databases.

User Name:
The user name to use to access the database.

Password:
The password for the user specified in the **User Name** field.

Initial pool size:
The initial size of the DBCP pool when it is first created.

Maximum number of active connections:
The maximum number of active connections that can be allocated from the JDBC pool at the same time. The default maximum is 8 active connections.

Maximum number of idle connections:
The maximum number of active connections that can remain idle in the pool without extra connections being released. The default maximum is 8 connections.

Minimum number of idle connections:
The minimum number of active connections that can remain idle in the pool without extra connections being created. The default minimum is 8 connections.

Maximum wait time:
The maximum number of milliseconds that the pool waits (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely. The default time is 10000ms, and a value of -1 indicates an indefinite time to wait.

Time between eviction:
The number of milliseconds to sleep between runs of the thread that evicts unused connections from the JDBC pool.

Number of tests:
The number of connection objects to examine from the pool during each run of the evictor thread. The default number of objects is 3.

Minimum idle time:
The minimum amount of time, in milliseconds, an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any).

Database connection pool settings

The table below shows the correspondence between the fields in the **Advanced - Connection pool** section of the window and the Apache Commons DBCP configuration properties:

Field Name	DBCP Configuration Property
URL	`url`
User Name	`username`
Password	`password`
Initial pool size	`initialSize`
Maximum number of active connections	`maxActive`
Maximum number of idle connections	`maxIdle`
Minimum number of idle connections	`minIdle`
Maximum wait time	`maxWait`

Field Name	DBCP Configuration Property
Time between eviction	`timeBetweenEvictionRunsMillis`
Number of tests	`numTestsPerEvictionRun`
Minimum idle time	`minEvictableIdleTimeMillis`

Connection validation

By default, when the API Gateway makes a connection, it performs a simple connection validation query. This enables the API Gateway to test the database connection before use, and to recover if the database goes down (for example, if there is a network failure, or if the database server reboots).

The API Gateway validates connections by running a simple SQL query (for example, a `SELECT 1` query with MySQL or MariaDB). If it detects a broken connection, it creates a new connection to replace it.

Test the connection

When you have specified all the database connection details, you can click the **Test Connection** button to verify that the connection to the database is configured successfully. This enables you to detect any configuration errors at design time instead at runtime.

Configure database queries

Overview

The **Database Statement** dialog enables you to enter an SQL query, stored procedure, or function call that the API Gateway runs to return a specific user's profile from a database.

Configuration

The following fields should be completed on this window:

Name:
Enter a name for this database query here.

Database Query:
Enter the actual SQL query, stored procedure, or function call in the text area provided. When executed, the query should return a single user's profile. The following are examples of SQL statements and stored procedures:

```
select * from users where username=${authentication.subject.id}
{ call load_user (${authentication.subject.id}, ${out.param}) }
{ call ${out.param.cursor} := p_test.f_load_user(${authentication.subject.id}) }
```

These examples show that you can use selectors in the query. The selector that is most commonly used in this context is `${authentication.subject.id}`, which specifies the message attribute that holds the identity of the authenticated user. For more details on selectors, see *Select configuration values at runtime* on page 421.

Statement Type:
The database can take the form of an SQL query, stored procedure, or function call, as shown in the above examples. Select the appropriate radio button depending on whether the database query is an SQL **Query** or a **Stored procedure/function call**.

Table Structure:
To process the result set that is returned by the database query, the API Gateway needs to know whether the user's attributes are structured as rows or columns in the database table.

The following example of a database table shows the user's attributes (Role, Dept, and Email) structured as table columns:

Username	Role	Dept	Email
Admin	Administrator	Engineering	admin@org.com
Tester	Testing	QA	tester@org.com
Dev	Developer	Engineering	dev@org.com

In the following table, the user's attributes have been structured as name-value pairs in table rows:

Username	Attribute Name	Attribute Value
Admin	Role	Administrator
Admin	Dept	Engineering
Admin	Email	admin@org.com
Tester	Role	Testing
Tester	Dept	QA

Username	Attribute Name	Attribute Value
Tester	Email	tester@org.com
Dev	Role	Developer
Dev	Dept	Engineering
Dev	Email	dev@org.com

If the user's attributes are structured as column names in the database table, select the **attributes as column names** radio button. If the attributes are structured as name-value pair in table rows, select the **attribute name-value pairs in rows** option.

Configure ICAP servers

Overview

The Internet Content Adaptation Protocol (ICAP) is a lightweight HTTP-based protocol used to optimize proxy servers, which frees up resources and standardizes how features are implemented. For example, ICAP is typically used to implement features such as virus scanning, content filtering, ad insertion, or language translation in the HTTP proxy server cache. *Content Adaptation* refers to performing a specific value-added service (for example, virus scanning) for a specific client request and/or response.

You can configure ICAP servers under the **Environment Configuration** > **External Connections** tree node, which you can then specify in an **ICAP** filter later. To configure an ICAP server, right-click the **ICAP Servers** node, and select **Add an ICAP Server** to display the **ICAP Server Settings** dialog.

Server settings

Configure the following settings on the **Server** tab:

Host	The machine name or IP address of the remote ICAP host. Defaults to the localhost (`127.0.0.1`).
Port	The port on which the ICAP server is listening. Defaults to `1344`.
Request Service	The path to the service (exposed by the ICAP server) that handles Request Modification (REQMOD) requests. The default value is `/request`.

Response Service	The path to the service exposed by the ICAP server that handles Response Modification (RESPMOD) requests. The default value is `/response`.
Options Service	The path to the service (exposed by the ICAP server) that handles OPTIONS requests. OPTIONS requests enable server capabilities to be queried. The default value is `/options`.

Security settings

The following settings on the **Security** tab enable you to secure the connection to the ICAP server:

Trusted Certificates:
When the API Gateway connects to the ICAP server over SSL, it must decide whether to trust the ICAP server's SSL certificate. You can select a list of CA or server certificates on the **Trusted Certificates** tab that are considered trusted by API Gateway when connecting to the ICAP server. The table displayed on the **Trusted Certificates** tab lists all certificates imported into the API Gateway Certificate Store. To *trust* a certificate for this particular connection, select the box next to the certificate in the table.

Client SSL Authentication:
In cases where the destination ICAP server requires clients to authenticate to it using an SSL certificate, you must select a client certificate on the **Client SSL Authentication** tab. Select the check box next to the client certificate that you want to use to authenticate to the ICAP server.

Advanced:
The **Ciphers** field enables you to specify the ciphers that API Gateway supports. The API Gateway sends this list of supported ciphers to the destination ICAP server, which then selects the highest strength common cipher as part of the SSL handshake. The selected cipher is then used to encrypt the data when it is sent over the secure channel.

Advanced settings

Select one of the following ICAP server modes on the **Advanced** tab:

Request Modification Mode (REQMOD)	Specifies that the **ICAP** filter in the API Gateway sends a Request Modification (REQMOD) request to the ICAP server. The ICAP server returns a modified version of the request, an HTTP response, or a 204 response code indicating that no modification is required. This mode is selected by default.
Response Modification Mode (RESPMOD)	Specifies that the **ICAP** filter in the API Gateway sends a Response Modification (RESPMOD) request to the ICAP server. For example, the API Gateway sends an origin server's HTTP response to the ICAP server. The response from the ICAP server can be an adapted HTTP response, an error, or a 204 response code indicating that no adaptation is required.

Further information

For example policies that show an **ICAP** filter configured in REQMOD and RESPMOD modes, see the *API Gateway Policy Developer Filter Reference*.

Configure Sentinel servers

Sentinel server overview

Axway Sentinel is a Business Activity Monitoring (BAM) product that collects, aggregates, correlates, and reports events from API Gateway and other Axway products, applications, and systems throughout your infrastructure. Sentinel is a separate product that you can buy from Axway or an authorized partner. This topic describes how you can configure API Gateway to send events to Axway Sentinel server.

Note A complete documentation set for Axway Sentinel is available on the Axway Support website: https://support.axway.com.

To add a new Sentinel server connection, in the Policy Studio tree, under the **Environment Configuration** > **External Connections** node, right-click the **Sentinel Servers** node, and select **Add a Sentinel Server**.

General settings

You can configure the following general settings for the Sentinel server:

Name:
Enter a suitable name for this Sentinel server.

Host:
Enter the host name (FQDN) or IP address of the Sentinel server.

Port:
Enter the port number that the Sentinel server is listening for events on.

Use overflow file:
Select this option to use an overflow file to store API Gateway event data when there is no connection between API Gateway and Sentinel.

Name:
Enter a suitable name for the overflow file.

Size (MB):
Enter the maximum size of the overflow file.

Encoding:
Enter the encoding type to use. The default is `utf-8`.

User agent settings:
By default API Gateway registers events against the API Gateway's name in the topology. To use another name, select the **Or the following name** option and enter an alternative name to use.

By default API Gateway registers events against the host name of the machine running the API Gateway. To use another host name, select the **Or the following name** option and enter an alternative host name to use.

Further information

For more detailed information, see the *API Gateway Sentinel Interoperability Guide* .

Configure Kerberos clients

Overview

API Gateway can act as a Kerberos client. For this, it must authenticate to the Kerberos Key Distribution Center (KDC) as a specific Kerberos principal, and use the Ticket Granting Ticket (TGT) granted for it to obtain tickets from the Ticket Granting Service (TGS) to authenticate to Kerberos services.

You can configure Kerberos clients globally under **Environment Configuration > External Connections** in the node tree. Right-click the **Kerberos Clients** node in the tree, and select **Add a Kerberos Client**. The following sections describe how to configure the different fields int he dialog.

Once finished, you can select the configured Kerberos client when configuring other Kerberos-related filters. Ensure the check box **Enabled** at the bottom of the window is selected.

For more details on different Kerberos setups with API Gateway, see *API Gateway Kerberos Integration Guide*.

Kerberos endpoint settings

Configure the following fields on the **Kerberos Endpoint** tab:

Ticket Granting Ticket Source

This option defines where to obtain the Kerberos client credentials and the session key used in communications with the TGS to request TGTs. A TGT can be retrieved from a cache created as part of a Java Authentication and Authorization Service (JAAS) login, from delegated credentials, or from the native GSS Library on Linux platforms.

Note Depending on the TGT source option option you select, the **Kerberos Principal**, **Password**, and **Keytab File** fields below might be required or disabled.

Load via JAAS Login:
By default, API Gateway performs a JAAS login to the Kerberos KDC. After the login, API Gateway caches the credentials, and they are used to acquire TGTs as needed.

The JAAS login acquires the credentials in one of the following ways:

- **Request from KDC** – Request a new TGT from the Kerberos KDC. The request is sent at server startup, server refresh (for example, when an update to configuration is deployed), and when the TGT expires.
- **Extract from Default System Ticket Cache** – If a TGT has already been obtained outside API Gateway and has been stored in the default system ticket cache, you can use this option to retrieve the TGT from the cache. On Windows 2000, the TGT is extracted from the cache using the Local Security Authority (LSA) API. On Linux, the assumed default location of the ticket cache is `/tmp/krb5cc_<uid>` , where `<uid>` is a numeric user identifier. If the ticket cache cannot be found in the default locations, or on a different Windows platform, API Gateway searches the cache in `{user.home}{file.separator}krb5cc_{user.name}`.

  Note A system ticket cache can only hold the credentials of a single Kerberos client. To load credentials for more than one Kerberos client from system ticket caches, select **Extract from System Ticket Cache** option.

- **Extract from System Ticket Cache** – Extract the TGT from an explicitly named system ticket cache instead of the default system ticket cache.To specify the cache, browse to the cache you want to use. You can populate ticket caches with client credentials using an external utility, such as `kinit`.

Load from Delegated Credentials:
The Kerberos client can use a delegated TGT that has been already retrieved from a **Kerberos Service** authentication filter. The TGT is extracted from message attributes (for example, `authentication.delegated.credentials` and `authentication.delegated.credentials.client.name`) that have been set by the Kerberos Service filter.

If you select this option, it is not necessary to configure the **Kerberos Principal** or **Secret Key** fields.

Load via Native GSS Library:
Select this option to use the Native GSS-API to acquire the client's credentials. The Native GSS-API expects that the credentials already are in a system ticket cache that it can access.

The **Load via Kinit** option determines the number of Kerberos clients you can use with API Gateway:

- If **Load via Kinit** *is* selected, API Gateway can support multiple Kerberos clients natively. API Gateway runs `kinit` and creates a ticket cache for each client in the `/conf/plugins/kerberos/cache` directory. The Native GSS-API can automatically acquire the client credentials from these caches.

- If **Load via Kinit** *is not* selected, API Gateway can support only one Kerberos client natively. The Kerberos client credentials must be in the default system ticket cache. API Gateway cannot support accessing credentials natively from the default system ticket cache and other system ticket caches.

Note To use the native GSS library and the `kinit` tool, you must select to use the native GSS library on the instance-level API Gateway **Kerberos Configuration** settings. For more details, see *Kerberos configuration* on page 204.

Kerberos Principal

A Kerberos principal is used to assign a unique identity for API Gateway to be used in the Kerberos environment.

To select which Kerberos principal to use, click the **...** button, and select a previously configured principal from the list. To add a Kerberos principal, right-click **Kerberos Principals**, and select **Add Kerberos Principal**. You can also add Kerberos principals under **Environment Configuration > External Connections** in the node tree.

For Kerberos constrained delegation (KCD), the Kerberos Principal selected here must be configured for credential delegation to a set of Kerberos services in the Kerberos KDC.

For more details, see *Configure Kerberos principals* on page 400.

Note The semantics of this field are slightly different depending on what TGT source you selected:

- If you selected to retrieve the TGT from the KDC, you are effectively asking the KDC to issue a TGT for the principal selected here.
- If you selected to retrieve the TGT from a system ticket cache, the principal selected here searches the cache to retrieve the TGT.
- If you selected to use the `kinit` utility, the name of the principal selected here is passed as an argument to `kinit`.
- If you selected to retrieve a TGT from delegated credentials, it is not necessary to specify any principal.

Secret Key

The Kerberos principal uses the secret key to communicate with the KDC's Authentication Service in order to acquire a TGT. The secret key can be generated from a password, or it can be acquired from the principal's keytab file. The options available depend on what you selected as the source of the TGT.

Enter Password:
You can only enter a password if you selected to request the TGT from the KDC. The password is used when generating the secret key. A secret key is not required if the TGT has been already retrieved either from a system ticket cache or from delegated credentials.

Note By default, the password entered here is stored in clear-text form in the underlying configuration data in API Gateway. If necessary, the password can be encrypted using a passphrase. For more details on encrypting sensitive API Gateway configuration data, see the *API Gateway Administrator Guide*.

Wildcard Password:
##enter stuff here

Keytab:
If you selected to request the TGT from the KDC, you can also extract the secret key for the principal from a keytab file, which maps principal names to encryption keys. Using the `kinit` tool also requires a keytab file.

To load the principal-to-key mappings into the table in the dialog, select **Load Keytab** and browse to the existing keytab file. To add a new keytab entry, select **Add Principal**. To delete a keytab entry, select the entry in the table and click **Delete Entry**. To export the entire contents of the keytab table in the dialog, click **Export Keytab**.

Note By default, the contents of the keytab table – derived from a keytab file or manually entered – stored in clear-text form in the underlying configuration data in API Gateway. If necessary, the contents of the keytab table can be encrypted using a passphrase. For more details on encrypting sensitive API Gateway configuration data, see the *API Gateway Administrator Guide*.

When API Gateway starts, it writes the stored keytab contents to the `/conf/plugin/kerberos/keytabs/` directory in your API Gateway installation. It is recommended to configure directory-based or file-based access control for this directory and its contents.

For more details on configuring the **Keytab Entry** dialog, see the *Kerberos keytab concepts* on page 405.

Kerberos Constrained Delegation settings

An end user application requesting a TGT for a back-end Kerberos service might be unable to authenticate to the Kerberos service with Kerberos credentials, for example, if accessing the service from outside the Kerberos domain. KCD enables the Kerberos principal you selected on the **Kerberos Endpoint** tab to act as a trusted Kerberos principal. The trusted Kerberos principal authenticates the end user application using some other authentication method than Kerberos credentials, then authenticates to the back-end Kerberos service as the end user application, and acquires a TGT in the name of the end user application.

To enable KCD, configure the following fields on the **Kerberos Constrained Delegation** tab:

Kerberos Principal to Impersonate:

Select the end user principal being impersonated. The principal to impersonate is normally configured using a selector, so that many end user principals can be impersonated. A typical setup might be that the principal on the **Kerberos Endpoint** tab might be, for example, `TrustedAPIGateway@AXWAY.COM` and the Kerberos Principal to Impersonate is `${authentication.subject.id}@AXWAY.COM`.

If you don't set this field, the Kerberos Client does not attempt to impersonate other Kerberos principals in Kerberos Constrained Delegation.

Select Cache for Impersonated Subjects:

Select the cache used to store impersonated user credentials. These will be refreshed automatically so that expired credentials are not sent to the service-side.

You must define this field for KCD.

Advanced settings

You can configure the following fields on the **Advanced** tab:

Mechanism:
Select the mechanism used to establish a context between the Kerberos client and the Kerberos service. The Kerberos service must use the same mechanism.

Mutual Authentication:
Select this to carry out mutual authentication . This means that the service authenticates back to the client during the context setup. For SPNEGO mechanism, you must select this.

Integrity:
Select to enable data integrity for GSS operations.

Confidentiality:
Select to enables data confidentiality for GSS operations.

Credential Delegation:
Select this to delegate the request initiator's credentials to the acceptor during context setup. If you select this option, the acceptor can assume the initiator's identity and authenticate to other Kerberos services on behalf of the initiator.

Anonymity:
Select to prevent disclosing the Kerberos client's identity to the Kerberos service.

Replay Detection:
Select to enable replay detection for the per-message security services after context establishment.

Sequence Checking:
Select to switch on sequence checking for the per-message security services after context establishment.

Synchronize to Avoid Replays Errors at Service:

If a Kerberos client is running under stress and is attempting to send many requests to a Kerberos service within a very short (millisecond) time frame, the sequential Kerberos Authenticator tokens the Kerberos client generates might contain identical values for the `ctime` (the current time on the client's host) and `cusec` (the microsecond portion of the client's timestamp) fields.

Since Kerberos service implementations often compare the `ctime` and `cusec` values on successive Kerberos Authenticator tokens to discover replay attacks, it is possible that the Kerberos service might reject Kerberos Authenticator requests in which the `ctime` and `cusec` fields have the same value.

To avoid this scenario, you can select this option to synchronize the creation of the Kerberos Authenticator requests using the **Pause Time** value.

Note On Linux, this setting is not required, so ensure it is deselected for better performance.

Pause Time:
Specify the time interval (in milliseconds) to wait before generating the client-side Kerberos Authenticator tokens when synchronizing to avoid over-zealous replay detection on the Kerberos service. You can only set this value if you have also selected the **Synchronize to Avoid Replays Errors at Service** option.

Note The default value of 15 milliseconds matches the clock resolution time of operating systems. For more details on the clock resolution on your target operating system, consult your operating system documentation .

Refresh credential when remaining validity is X secs:

Select this to enable refresh the Kerberos credentials shortly before they expire to avoid expiry failures on the service-side.

When API Gateway receives a request that uses a Kerberos client, API Gateway refreshes any cached Kerberos credentials used to process that request, if the time that this credential can be validly used is less than or equal to this number of seconds.

It is possible that a credential is only refreshed long after it has expired, as triggering the refresh requires a request that uses the expired credential.

Configure Kerberos principals

Overview

A Kerberos principal represents a unique identity in a Kerberos system to which Kerberos can assign tickets to access Kerberos-aware services. Principal names are made up of several components separated with the `/` separator. You can also specify a Kerberos realm as the last component of the name by using the `@` character. If no realm is specified, the principal is assumed to belong to the default realm, as configured in the `krb5.conf` file. For more details, see *Kerberos configuration on page 204*.

Typically, a principal name comprises three parts: the *primary*, the *instance*, and the *realm*. The format of a typical Kerberos v5 principal name is:

```
primary/instance@realm
```

- *Primary:*
  If the principal represents a user in the system, the primary is the user name of the user. Alternatively, for a host, the primary is specified as the `host` string.
- *Instance:*
  The instance can be used to further qualify the primary (for example, `user/admin@foo.abc.com`).
- *Realm:*
  This is your Kerberos realm, which is usually a domain name in upper case letters. For example, the `foo.abc.com` machine is in the `ABC.COM` Kerberos realm.

For more details on different Kerberos setups with API Gateway, see *API Gateway Kerberos Integration Guide*.

Configuration

You can configure Kerberos principals globally under the **Environment Configuration** > **External Connections** in the node tree. To configure a Kerberos principal, right-click the **Kerberos Principals** node, and select **Add a Kerberos Principal**.

Configure the following fields on the **Kerberos Principal** dialog:

Name:
Enter a friendly name for the Kerberos principal. This name is shown in the drop-down lists in other Kerberos-related configuration windows in the Policy Studio.

Principal Name:
Enter the name of the Kerberos principal in this field. The principal name consists of a number of components separated using the `/` separator. Specify the realm here if the principal belongs to either a non-default realm or if a default realm is not specified in the `krb5.conf` file.

Principal Type:
Select the type of principal specified in the field above. The following table lists the available principal types.

Note The principal name types and their corresponding OIDs are defined in the General Security Services API (GSS-API).

Principal name type	Explanation	OID
`NT_USER_NAME`	The principal name identifies a named user on the local system	`1.2.840.113554.1.2.1.1`
`KERBEROS_V5_PRINCIPAL_NAME`	The principal name represents a Kerberos version 5 principal.	`1.2.840.113554.1.2.2.1`

Principal name type	Explanation	OID
`NT_EXPORT_NAME`	The principal name represents an exported canonical byte representation of the name (for example, which can be used when searching for the principal in an Access Control List (ACL)).	`1.3.6.1.5.6.4`
`NT_HOSTBASED_SERVICE`	The principal name identifies a service associated with a specific host.	`1.3.6.1.5.6.2`

To add new principal types, click **Add**. The name entered in the **Name** field on the **Kerberos Principal Name OID** must correspond to one of the constant fields defined in the `org.ietf.jgss.GSSName` Java class. For other allowable name types, see the Javadocs for the GSSName class.

Similarly, the corresponding OID for this name type must be entered in the **OID** field of the dialog.

Note OIDs and principal type names must only be changed to reflect changes in the underlying GSS-API. Because of this, do not edit the existing **Principal Types** except under strict supervision by the Axway Support.

Configure Kerberos services

Overview

API Gateway can act as a Kerberos service. A Kerberos client must obtain a Ticket Granting Ticket (TGT) to authenticate to the Kerberos service exposed by API Gateway. The Kerberos client must present the TGT to API Gateway to be successfully authenticated, and for their requests to be processed. The Kerberos service is responsible for consuming the TGT.

You can configure Kerberos services globally under the **Environment Configuration > External Connections** in the node tree. To configure a Kerberos service, right-click **Kerberos Services**, and select **Add a Kerberos Service**. The following sections describe how to configure the various fields on the **Kerberos Service** dialog.

You can select globally configured Kerberos services by name when configuring a **Kerberos Service** filter. This filter is responsible for validating the TGTs consumed by the Kerberos service. To easily tell different Kerberos services apart, ensure you enter a descriptive name for the service in the **Name** when configuring a Kerberos service. For more details on configuring the Kerberos Service filter, see "Kerberos service authentication" in the *API Gateway Policy Developer Filter Reference*.

Once finished, you can select the configured Kerberos service when configuring other Kerberos-related filters. Ensure the check box **Enabled** at the bottom of the window is selected.

For more details on different Kerberos setups with API Gateway, see *API Gateway Kerberos Integration Guide*.

Kerberos endpoint settings

Configure the following fields on the **Kerberos Endpoint** tab:

Kerberos Principal

Select the name of the Kerberos principal you want to associate with API Gateway. A Kerberos client trying to authenticate to API Gateway *must* present a TGT containing a matching Kerberos principal name to API Gateway.

To select which Kerberos principal to use, click the **...** button, and select a previously configured principal from the list. To add a Kerberos principal, right-click **Kerberos Principals**, and select **Add Kerberos Principal**. You can also add Kerberos principals under **Environment Configuration > External Connections** in the node tree. For more details, see *Configure Kerberos principals* on page 400.

Secret Key

Select this to specify the location of the Kerberos service's secret key that is used to decrypt TGTs received from Kerberos clients.

Enter Password:
You can enter a password to generate the Kerberos service's secret key. This key is originally created for a specific principal on the KDC.

Wildcard Password:
##enter stuff here

Keytab:
Instead of generating a secret key for the Kerberos service in this configuration dialog, adding the Kerberos service to the KDC usually generates a keytab file containing a mapping between the Kerberos principal name and that principal's secret key. You can load the keytab file to API Gateway configuration using the fields in this section.

To load the principal-to-key mappings into the table in the dialog, select **Load Keytab** and browse to the existing keytab file. To add a new keytab entry, select **Add Principal**. To delete a keytab entry, select the entry in the table and click **Delete Entry**. To export the entire contents of the keytab table in the dialog, click **Export Keytab**.

Note By default, the contents of the keytab table – derived from a keytab file or manually entered – stored in clear-text form in the underlying configuration data in API Gateway. If necessary, the contents of the keytab table can be encrypted using a passphrase. For more details on encrypting sensitive API Gateway configuration data, see the *API Gateway Administrator Guide*.

When API Gateway starts, it writes the stored keytab contents to the `/conf/plugin/kerberos/keytabs/` directory in your API Gateway installation. It is recommended to configure directory-based or file-based access control for this directory and its contents.

For more details on configuring the **Keytab Entry** dialog, see the *Kerberos keytab concepts* on page 405.

Load via Native GSS Library:
If you have configured API Gateway to use Native GSS Library on the instance-level **Kerberos Configuration** settings, you must choose to load the Kerberos service's secret key from the preferred location. The native GSS library expects the Kerberos service's secret key to be in the default system keytab file. The location of this keytab file is specified in the `default_keytab_name` setting in the `krb5.conf` file that the native GSS library reads using the `KRB5_CONFIG` environment variable. This keytab can contain keys for multiple Kerberos services.

Advanced settings

Configure the following fields on the **Advanced** tab:

Mechanism:
Select the mechanism used to establish a context between the Kerberos service and the Kerberos client. The Kerberos client must use the same mechanism.

Extract Delegated Credentials:
A Kerberos client can set an attribute on the context established with the Kerberos service to indicate the Kerberos service can act on behalf of the Kerberos client in subsequent communications. This enables a Kerberos service (like API Gateway) to assume the identity of the Kerberos client when communicating with a back-end Kerberos service. The Kerberos client's credentials are propagated to the back-end Kerberos service, not API Gateway's credentials. This is called *credential delegation*.

If you have configured a Kerberos client to enable delegating its credentials to a Kerberos service, select **Extract Delegated Credentials** to set the Kerberos service to extract the delegated credentials from the context it establishes with the Kerberos client. The credentials are stored in the `gss.delegated.credentials` and `gss.delegated.credentials.client.name` message attributes.

To forward the extracted delegated credentials to the back-end Kerberos service on behalf of the Kerberos client, configure the Kerberos settings on the **Kerberos Client** filter, or use the **Connection** filter from the Routing category. If using the **Connection** filter, make sure to select **Extract from delegated credentials** on the **Kerberos Endpoint** tab to retrieve the TGT from the extracted delegated credentials for the **Kerberos Client** on the **Kerberos Authentication** tab of the **Connection** filter.

For more details on configuring these options, see the following topics:

- "Connection" in the *API Gateway Policy Developer Filter Reference*
- *Configure Kerberos clients* on page 395

Kerberos keytab concepts

The Kerberos keytab file contains mappings between Kerberos principal names and DES-encrypted keys that are derived from the password used to log into the Kerberos Key Distribution Center (KDC). The purpose of the keytab file is to allow the user to access distinct Kerberos services without being prompted for a password at each service. Furthermore, it allows scripts and daemons to log in to Kerberos services without the need to store clear-text passwords or the need for human intervention.

Note Anyone with read access to the keytab file has full control of all keys contained in the file. For this reason, it is imperative that the keytab file is protected using very strict file-based access control.

Each key entry in the Kerberos keytab file is identified by a Kerberos principal and an encryption type. For this reason, a keytab file can hold multiple keys for the same principal, each key belonging to a different encryption type. If the keytab file contains several keys for a principal, the Kerberos client or service uses the key with the strongest encryption type as agreed during the negotiation of previous messages with the KDC.

A keytab file can also contain keys for several different principals. In this case, at runtime, the Kerberos client or service only considers keys mapped to the Kerberos principal name you selected in the **Kerberos Principal** drop-down list when configuring that Kerberos client or service.

The **Keytab** table in the **Secret Key** section of the configuration window for a Kerberos client or Kerberos service is essentially a graphical interface to entries in a Kerberos keytab file. To generate a keytab entry, select **Keytab > Add Principal**. To remove an entry, select the entry and click **Delete Entry**. You can configure Kerberos clients and services under **Environment Configuration > External Connections** in the node tree.

For more details on different Kerberos setups with API Gateway, see *API Gateway Kerberos Integration Guide*.

Configuration

Configure the following fields on the **Keytab Entry** dialog:

Kerberos Principal:
Select an existing Kerberos principal from the drop-down list. To add a Kerberos principal, right-click **Kerberos Principals**, and select **Add Kerberos Principal**. You can also add Kerberos principals under **Environment Configuration > External Connections** in the node tree. For more details, see *Configure Kerberos principals* on page 400.

Password:
Enter the password to seed the encryption algorithms for the encryption types.

Key version number:
Set the version number for the encryption key.

Encryption Types:
Select the encryption types. The encryption types determine the algorithms used to generate the encryption keys stored in the keytab file. If the keytab file contains multiple keys for a Kerberos principal, the encryption type is used to select an appropriate encryption key.

To ensure maximum interoperability between Kerberos clients and Kerberos services configured in API Gateway and different KDCs, all encryption types are selected by default. This way, the generated keytab entry contains a separate encryption key for each encryption type listed here, and each key is mapped to the selected Kerberos principal name.

Note You must ensure that the required encryption types exist in the keytab as defined in Kerberos system settings in the `krb5.conf` file. For a Kerberos client to request a Ticket Granting Ticket (TGT), it must have at least one key that matches one of the encryption types listed in the `default_tkt_enctypes` setting in `krb5.conf`. A Kerberos service requires a key of a matching encryption type to be able to decrypt a TGT a Kerberos client presents.
For more details on the `krb5.conf` file, see *Kerberos configuration* on page 204.

By default, for Windows 2003 Active Directory, TGT is encrypted using the `rc4-hmac` encryption type. However, if the service user has enabled **Use DES encryption types for this account**, the `des-cbc-md5` encryption type is used.

Configure LDAP directories

A filter that uses an LDAP directory to authenticate a user or retrieve attributes for a user must have an LDAP directory associated with it. You can use the **Configure LDAP Server** dialog to configure connection details of the LDAP directory. Both LDAP and LDAPS (LDAP over SSL) are supported.

When a filter that uses an LDAP directory is run for the first time after a server restart, the server binds to the LDAP directory using the connection details configured on the **Configure LDAP Server** dialog. Usually, the connection details include the user name and password of an administrator user who has read access to all users in the LDAP directory for whom you wish to retrieve attributes or authenticate.

General configuration

Configure the following general LDAP connection settings:

Name:
Enter or select a name for the LDAP filter in the drop-down list.

URL:
Enter the URL location of the LDAP directory. The URL is a combination of the protocol (LDAP or LDAPS), the IP address of the host machine, and the port number for the LDAP service. By default, port `389` is reserved for LDAP connections, while port `636` is reserved for LDAPS connections. For example, the following are valid LDAP directory URLs:

```
ldap://192.168.0.45:389
```

```
ldaps://145.123.0.28:636
```

Cache Timeout:
Specifies the timeout for cached LDAP connections. Any cached connection that is not used in this time period is discarded. Defaults to 300000 milliseconds (5 minutes). A cache timeout of 0 means that the LDAP connection is cached indefinitely and never times out.

Cache Size:
Specifies the number of cached LDAP connections. Defaults to 8 connections. A cache size of 0 means that no caching is performed.

Authentication configuration

If the configured LDAP directory requires clients to authenticate to it, you must select the appropriate authentication method in the **Authentication Type** field. When the API Gateway connects to the LDAP directory, it is authenticated using the selected method. Choose one of the following authentication methods:

- None
- Simple
- Digest-MD5
- External

Note If any of the following authentication methods connect to the LDAP server over SSL, that server's SSL certificate must be imported into the API Gateway certificate store.

None

No authentication credentials need to be submitted to the LDAP server for this method. In other words, the client connects anonymously to the server. Typically, a client is only allowed to perform read operations when connected anonymously to the LDAP server. It is not necessary to enter any details for this authentication method.

Simple

Simple authentication involves sending a user name and corresponding password in clear text to the LDAP server. Because the password is passed in clear text to the LDAP server, it is recommended to connect to the server over an encrypted channel (for example, over SSL).

It is not necessary to specify a **Realm** for the Simple authentication method. The realm is only used when a hash of the password is supplied (for Digest-MD5). However, in cases where the LDAP server contains multiple realms, and the specified user name is present in more than one of these realms, it is at the discretion of the specific LDAP server as to which user name *binds* to it.

Select the **SSL Enabled** checkbox to force the API Gateway to connect to the LDAP directory over SSL.

To successfully establish SSL connections with the LDAP server, you must import the server's certificate into the JRE truststore of API Gateway. The default truststore is located under `INSTALL-DIR/platform/jre/lib/security/cacerts`. To add the server certificate, use a keytool command like the following:

```
keytool -import -trustcacerts -keystore INSTALL-
DIR/apigateway/platform/jre/lib/security/cacerts
-storepass TRUSTSTORE_PASSWORD -alias ldap_server -import -file CERTIFICATE_FILE
```

`TRUSTSTORE_PASSWORD` is the default `cacerts` password, which is available in the JRE documentation.

Digest-MD5

With *Digest-MD5* authentication, the server generates some data and sends it to the client. The client encrypts this data with its password according to the MD5 algorithm. The LDAP server then uses the client's stored password to decrypt the data and hence authenticate the user.

The **Realm** field is optional, but may be necessary in cases where the LDAP server contains multiple realms. If a realm is specified, the LDAP server attempts to authenticate the user for the specified realm only.

External

External authentication enables you to use client certificate-based authentication when connecting to an LDAP directory. When this option is selected, you must select a client certificate from the API Gateway certificate store. The **SSL Enabled** checkbox is selected automatically. Click the **Signing Key** button to select the client certificate to use to mutually authenticate to the LDAP server.

Note This means that you must specify the **URL** field using LDAPS (for example, `ldaps://145.123.0.28:636`). The user name, password, and realm fields are not required for external authentication.

Test the LDAP connection

When you have specified all the LDAP connection details, click the **Test Connection** button to verify that the connection to the LDAP directory is configured successfully. This enables you to detect any configuration errors at design time, rather than at runtime.

For LDAPS (LDAP over SSL) connections, the LDAP server's certificate must be imported into the Policy Studio JRE trusted store. Perform the following steps in Policy Studio:

1. Select the **Environment Configuration** > **Certificates and Keys** > **Certificates** node in the Policy Studio tree.
2. In the **Certificates** panel on the right, click **Create/Import**, and click **Import Certificate + Key**.
3. Browse to the LDAP server's certificate file, and click **Open**.

4. Click **Use Subject** on the right of the **Alias Name** field, and click **OK**.
   The LDAP server's certificate is now imported into the **Certificate Store**, and must be added to the **Java keystore**.
5. In the **Certificates** panel, select the certificates that you wish the JRE to trust.
6. Click **Keystore**, and browse to the `cacerts` file in the following directory:

   ```
   INSTALL_DIR\policystudio\jre\lib\security
   ```

7. Select the `cacerts` file.
8. You are prompted for a password. (This is the default `cacerts` password which is available in the JRE documentation)
9. Click **OK**.
10. On the **Keystore** window, click **Add to keystore**.
11. Select the LDAP certificate imported previously.
12. Click **OK**.
13. Restart Policy Studio.
14. You can now click **Test Connection** to test the connection to the LDAP directory server over SSL.

Additional JNDI properties

You can also specify optional JNDI properties as simple name-value pairs. Click the **Add** button to specify properties in the dialog.

Configure proxy servers

Overview

You can configure settings for individual proxy servers under the **Environment Configuration** > **External Connections** node in the Policy Studio tree, which you can then specify at the filter level (in the **Connection** and **Connect To URL** filters). When configured, the filter connects to the HTTP proxy server, which in turn routes the message on to the destination server named in the request URI.

For more details, see "Connection" in the *API Gateway Policy Developer Filter Reference* and "Connect to URL" in the *API Gateway Policy Developer Filter Reference*.

Note API Gateway does not support persistent SSL connections to back-end servers using proxy tunneling. The API Gateway connection caching mechanism is not designed for proxy tunnel connections.

Configuration

To configure a proxy server under the **Environment Configuration** > **External Connections** tree node, right-click the **Proxy Servers** node, and select **Add a Proxy Server**. You can configure the following settings in the dialog:

Proxy Server Setting	Description
Name	Unique name or alias for these proxy server settings.
Host	Host name or IP address of the proxy server.
Port	Port number on which to connect to the proxy server.
Username	Optional user name when connecting to the proxy server.
Password	Optional password when connecting to the proxy server.
Scheme	Specifies whether the proxy server uses the HTTP or HTTPS transport. Defaults to HTTP.

Tip These proxy server settings are different from the global proxy settings in the **Preferences** dialog in the Policy Studio, which apply only when downloading WSDL, XSD, and XSLT files from the Policy Studio. For more details, see *Policy Studio preferences* on page 180.

Configure RADIUS clients

Note This feature has been deprecated and will be removed in a future release.

The API Gateway provides support for integration with remote systems over the Remote Authentication Dial In User Service (RADIUS) protocol. RADIUS is a client-server network protocol that provides centralized authentication and authorization for clients connecting to remote services. For more details, see the RADIUS specification.

To configure a client connection to a remote server over the RADIUS protocol, under the **Environment Configuration > External Connections** tree node in the Policy Studio, select **RADIUS Clients > Add a RADIUS Client**. This topic explains how to configure the settings the **RADIUS Client** dialog.

For details on how to configure a RADIUS authentication repository, see *Configure authentication repositories* on page 365.

Configuration

Configure the following fields in the **RADIUS Client** dialog:

Name:
Enter an appropriate name for the RADIUS client to display in Policy Studio.

Host name:
Enter the host name used by the RADIUS client.

Client port:
Enter the port number used by the RADIUS client.

RADIUS servers:
This field displays a list of configured RADIUS servers. To add a server to the list, click **Add**, and complete the following fields:

Name	Name of the RADIUS server.
Port	Port number used by the RADIUS server.
Secret	Shared secret used to access the RADIUS server.
Response timeout (sec)	Response timeout in seconds before the connection to the server is closed.
Retransmit count	Number of times retransmission is attempted before the connection to the server fails.

Configure SiteMinder/SOA Security Manager connections

This topic explains how to create connections to CA SiteMinder and CA SOA Security Manager.

Before you start, you must register API Gateway as an agent in the CA SiteMinder Policy Server. For more details, see *API Gateway Authentication and Authorization Integration Guide*.

Prerequisites

The following prerequisites apply to SiteMinder/SOA Security Manager connections.

CA SiteMinder connections

Integration with CA SiteMinder requires CA SiteMinder SDK version 12.52-sp02 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

1. Ensure that any SiteMinder binaries you may have previously added to API Gateway have been deleted.
2. Install CA SiteMinder SDK.

3. Copy the following `jar` files from the `java` directory of the CA SDK :
    - `cryptoj.jar`
    - `smagentapi.jar`
    - `smjavasdk2.jar`
4. Add the files to the following directory on API Gateway:

    ```
    INSTALL_DIR/apigateway/<platform>/lib
    ```

5. Restart API Gateway.

CA SOA Security Manager connections

Integration with CA SOA Security Manager requires CA TransactionMinder SDK version 6.0 or later. You must add the required third-party binaries to your API Gateway and Policy Studio installations.

For details on obtaining the required third-party binaries, see your CA product documentation.

Add third-party binaries to API Gateway

To add third-party binaries to API Gateway, perform the following steps:

1. Add the binary files as follows:
    - Add `.jar` files to the `INSTALL_DIR/apigateway/ext/lib` directory.
    - Add `.so` files to the `INSTALL_DIR/apigateway/<platform>/lib` directory.
2. Restart API Gateway.

Add third-party binaries to Policy Studio

To add third-party binaries to Policy Studio, perform the following steps:

1. Select **Window > Preferences > Runtime Dependencies** in the Policy Studio main menu.
2. Click **Add** to select a JAR file to add to the list of dependencies.
3. Click **Apply** when finished. A copy of the JAR file is added to the `plugins` directory in your Policy Studio installation.
4. Click **OK**.
5. Restart Policy Studio with the `-clean` option. For example:

    ```
    > cd INSTALL_DIR/policystudio/
    > policystudio -clean
    ```

Add a new connection

To add a new connection, in the node tree, click **Environment Configuration > External Connections**, right-click the **SiteMinder/SOA Security Manager Connection** node, and select **Add a SiteMinder Connection** or **Add a SOA Security Manager Connection**.

To specify how API Gateway connects to SiteMinder, use the **SiteMinder Connection Details** dialog. To specify how API Gateway connects to CA SOA Security Manager, use the **SOA Security Manager Connection Details** dialog. The connection details to be configured are the same for both the SiteMinder and SOA Security Manager, with an additional setting for SOA Security Manager.

SiteMinder and SOA Security Manager connection settings

This section describes settings that are common to both CA SiteMinder and CA SOA Security Manager connections.

Agent Name:
Enter the name of the agent to connect to SiteMinder or SOA Security Manager in the **Agent Name** field. This name *must* correspond to the name of an agent previously configured in the CA SiteMinder Policy Server.

Agent Configuration Object:
The name entered must match the name of the Agent Configuration Object (ACO) configured in the CA SiteMinder Policy Server. The only feature represented by the ACO parameters that API Gateway currently supports is the `PersistentIPCheck` setting. If you set the `PersistentIPCheck` ACO parameter to `yes`, API Gateway compares the IP address from the last request (stored in a persistent cookie) with the IP address in the current request to see if they match. If the IP addresses do not match, API Gateway rejects the request. This check is disabled if you set this parameter to `no`.

SMHost.conf file created by smreghost:
API Gateway host machine must be registered as an agent with SiteMinder or SOA Security Manager. To register the host, you must use the `smreghost` tool on API Gateway machine. The tool creates a file called `SmHost.conf`, which you can then upload into API Gateway configuration using Policy Studio. For more details on creating the `SmHost.conf` file, see *API Gateway Authentication and Authorization Integration Guide*##(CSHID link straight to the topic?)

To add the `SmHost.conf` file to API Gateway, ensure the `SmHost.conf` file is copied to the same machine running Policy Studio, click **Browse**, and select the the `SmHost.conf` file. You can select whether to use an `SmHost.conf` or `SmHost.cnf` file in the dialog. You can also enter the file name as an environment variable selector (for example, `${env.SMHOST}`). For more details, see the *API Gateway DevOps Deployment Guide*.

After selecting the `SmHost.conf` configuration file, you can see the connection details in the text area.

SOA Security Manager connection settings

This section describes details that are specific to CA SOA Security Manager connections only. In addition to the fields already described in the previous section, you must also configure the following field on the **SOA Security Manager Connection Details** dialog.

XMLSDKAcceptSMSessionCookie:
This setting controls whether the CA SOA Security Manager authentication filter accepts a single sign-on token for authentication purposes. The single sign-on token must reside in the HTTP header field named `SMSESSION` to authenticate using this mechanism. This token is created and updated when the CA SOA Security Manager authorization filter runs successfully.

When this check box is selected, the authentication filter allows authentication using a single sign-on token.

Note If no single sign-on token is present in the message, the authentication filter authenticates fully by gathering credentials from the request in whatever manner has been configured in the CA SOA Security Manager. When this check box is unselected, the authentication filter authenticates fully (it never allows authentication using a single sign-on token).

Configure SMTP servers

Overview

You can configure the API Gateway to use a specified Simple Mail Transfer Protocol (SMTP) server to relay messages to an email recipient. You can configure the SMTP server as a global configuration item under the **Environment Configuration** > **External Connections** node. The SMTP server is then available for selection in the **SMTP** filter in the **Routing** category.

To configure an SMTP server, right-click the **Environment Configuration** > **External Connections > SMTP Servers** node, and select **Add an SMTP Server**. Alternatively, in the **SMTP** filter window, click the button beside the **SMTP Server Settings** field, right-click the **SMTP Servers** node, and select **Add an SMTP Server**.

Configuration

Configure the following fields in the **SMTP Server settings** dialog:

Name:
Enter an appropriate name for the SMTP server.

SMTP Server Settings
Specify the following fields:

- **SMTP Server Hostname**:
  Enter the host name or IP address of the SMTP server.

- **Port**:
  Enter the SMTP port on the specified server hostname. By default, SMTP servers run on port 25.
- **Connection Security**:
  Select the security required for the connection to the SMTP server (`SSL`, `TLS`, or `NONE`).
  Defaults to `NONE`.

Log on using
If you are required to authenticate to the SMTP server, specify the following fields:

- **User Name**:
  Enter the user name of a registered user that can access the SMTP server.
- **Password**:
  Enter the password for the user name specified.

Note The SMTP server connection supports a simple user name and password type of authentication. Microsoft Windows NT LAN Manager (NTLM) authentication is not supported.

Configure trusted certificates for SMTP connections

SMTP server connections use a Java keystore instead of the API Gateway certificate store. You must import the certificate chain into the following file:

`<`*`install-dir`*`>\apigateway\<`*`platform`*`>\jre\lib\security\cacerts`

The default password is `changeit`.

Configure TIBCO Rendezvous daemons

Overview

TIBCO Rendezvous is a low latency messaging product for real-time high throughput data distribution applications. A message can be sent from the TIBCO daemon running on the local machine to a single TIBCO daemon running on a separate host machine or it can be broadcast to several daemons running on multiple machines. Each message has a subject associated with it, which acts as the *destination* of the message.

A listener, which is itself a TIBCO daemon, can declare an interest in a subject on a specific daemon. Whenever a message is delivered to this subject on the daemon the message is delivered to the listening daemon.

The API Gateway can act as a listener on a specific subject at a TIBCO daemon, in which case it said to be acting as a *consumer* of TIBCO messages. Similarly, it can also send messages to a TIBCO daemon, effectively acting as a *producer* of messages. In both cases, the local TIBCO daemon must be configured to talk to the TIBCO daemons running on the remote machines.

For more information on consuming and producing messages to and from TIBCO Rendezvous, see the following topics:

- *TIBCO integration* on page 349
- *TIBCO Rendezvous listener* on page 350
- **TIBCO Rendezvous** filter in the *API Gateway Policy Developer Filter Reference*

The remainder of this topic describes how to configure a TIBCO Rendezvous daemon. For a more detailed description of how to configure the fields on this dialog, refer to your TIBCO Rendezvous documentation.

Configuration

You can configure TIBCO Rendezvous daemons under the **Environment Configuration** > **External Connections** tree node in the Policy Studio. Right-click the **TIBCO Rendezvous Daemons** node, and select **Add a TIBCO Rendezvous Daemon**. Configure the following fields on the **TIBCO Daemon Settings** dialog:

Name:
Enter a friendly name for this TIBCO Rendezvous daemon. When configured, this name is available for selection when configuring a **TIBCO Rendezvous Listener** and a **TIBCO Rendezvous** filter.

Service:
Communication between TIBCO Rendezvous daemons takes place using Pragmatic General Multicast (PGM) or Universal Datagram Protocol (UDP) services. The specified *service parameter* configures the local TIBCO Rendezvous daemon to use this type of service when sending or broadcasting messages to other TIBCO Rendezvous daemons who are also using this service.

You can specify the service in the following ways:

- **By Service Name**:
  If your network administrator has added an entry for TIBCO Rendezvous in a network database such as NIS (for example, `rendezvous 7500/udp`), you can enter the name of the service (for example, `rendezvous`) in this field.
- **By Port Number**:
  Alternatively, you can enter the port number on which the TIBCO Rendezvous daemon is listening (for example, `7500`).
- **Default Option**:
  If you leave this field blank, a default service name of `rendezvous` is assumed. For this reason, administrators should add an entry in the network database with this name (for example, `rendezvous 7500/udp`. This enables you to leave this field blank so that this default service is used.

Network:
If the machine on which the TIBCO Rendezvous daemon is running has more than one network interface, you can specify what interface to use for all communications with other daemons. Each TIBCO Rendezvous daemon can only communicate on a single network, meaning that separate daemons must be configured for each network you want the daemon to communicate on.

For simplicity, you can leave this field blank, in which case the primary network interface is used for communication with other daemons. For more information on how to configure different networks and multicast groups, see the TIBCO Rendezvous documentation.

Daemon:
The value entered here tells the API Gateway where it can find the TIBCO Rendezvous daemon, which is responsible for communicating with all other daemons on the network. This daemon can be local or remote.

For local daemons you need only specify the port number that the daemon is running on (for example `6500`). Alternatively, you can leave this field blank to connect to the daemon on the default port.

To connect to a remote daemon, you must specify both the host and port number of the daemon in this field (for example `daemon_host:6500`).

Configure Tivoli connections

Tivoli Access Manager for e-business is a commonly used product for securing web resources. Tivoli message filters allows the API Gateway to leverage existing Access Manager policies, thus avoiding the need to maintain duplicate policies in both products. At runtime, the Tivoli filters can delegate authentication and authorization decisions to Access Manager, and can also retrieve user attributes.

Tivoli connections determine how a particular API Gateway instance connects to an instance of a Tivoli server.

Note Each API Gateway instance can connect to a single Tivoli server.

Prerequisites

Before you can configure the **Tivoli Authorization** filter, you must configure API Gateway for integration with TAM. For more details, see *API Gateway Authentication and Authorization Integration Guide*

Configuration

To configure Tivoli connection, in the node tree, select **Environment Configuration > Server Settings > Security > Tivoli**. Alternatively, you can a click **Environment Configuration > External Connections > Tivoli Connection**, and select **Add a Tivoli Connection**. The newly added connection can then be assigned to a particular API Gateway instance.

In both cases, the **Tivoli Configuration** dialog is used to add the connection details required for the API Gateway to connect to the Tivoli server. The connection details are stored in the Tivoli configuration files. This dialog allows you to upload these files to the API Gateway.

Configure the following fields on the **Tivoli Configuration** dialog:

Name:

Enter a name onfor the connection, or select a previously configured Tivoli connection on which to base the new configuration.

Select configuration file specification method:

You can specify how to upload the Tivoli configurations files:

- **Upload Tivoli configuration files**: You can automatically upload all configuration files to a specified API Gateway instance. API Gateway overwrites these files each time at startup or refresh (for example, when configuration updates are deployed). This means that any changes to the main configuration file must be made using the Policy Studio and not directly to the file on disk.
- **Set file location for main Tivoli Configuration file**: You can manually copy the configuration files to the file system of a API Gateway instance and point API Gateway to pick up the configuration files from that location. API Gateway does not overwrite the files at startup or refresh time. You can edit the main configuration file directly using an editor of your choice.

Connection enabled:

Select to immediately enable the connection, deselect to disable the connection. You can edit this selection later to disable or enable the connection as needed by toggling this check box.

Tivoli configuration files:

If you selected to upload the Tivoli configuration files, use the use the **Load File** and **Next** buttons in the **Upload Tivoli configuration files** windows to upload all configuration files to the specified API Gateway instance. The files are displayed in the text area, and depending on the file, you may be able to edit the contents of the file.

Server-side Tivoli configuration file: Set the location of the main Tivoli configuration file.

Configure XKMS connections

Overview

XML Key Management Specification (XKMS) is an XML-based protocol that enables you to establish the trustworthiness of a certificate over the Internet. The API Gateway can query an XKMS responder to determine whether a given certificate can be trusted.

You can add XKMS Connections under the **Environment Configuration** > **External Connections** tree node in Policy Studio. To add a global XKMS Connection, right-click the **XKMS Connections** node, and select **Add an XKMS Connection**.

Configuration

Configure the following fields on the **Certificate Validation - XKMS** window.

Name:
Enter an appropriate name for this XKMS connection.

URL Group:
Select a group of XKMS responders from the URL Group list. The API Gateway attempts to connect to the XKMS responders in the selected group in a round-robin fashion. It attempts to connect to the responders with the highest priority first, before connecting to responders with a lower priority.

You can add, edit, or remove URL Groups by selecting the appropriate button. For more information on adding and editing URL groups, see *Configure URL groups* on page 443.

User Name:
Requests to XKMS responders can be signed by a user to whom the **Sign OCSP or XKMS Requests** privilege has been assigned. Only those users who have been assigned this privilege are displayed in the drop-down list. For more information on assigning privileges to users, see *Manage API Gateway users* on page 233.

Signing Key:
Click the **Signing Key** button to open the list of certificates in the Certificate Store. You can then select the key to use to sign requests to XKMS responders. This user must have been granted the Sign OCSP or XKMS Requests privilege.

Extend and customize API Gateway 10

This section describes some advanced features of Policy Studio that enable you to extend API Gateway to suit your environment:

For more information on extending and customizing API Gateway, for example, adding custom filters, see the *API Gateway Developer Guide*.

Advanced filter view

Overview

You can use the advanced filter view in Policy Studio to edit all filter settings as text values. This enables you to edit each field as a text value regardless of whether the field is displayed as a radio button, check box, or drop-down list in the default user-friendly view for the filter.

This also means that you can specify all filter fields using the API Gateway selector syntax. This enables settings to be evaluated and expanded at runtime using metadata (for example, from message attributes, a Key Property Store (KPS), or environment variables). This is a powerful feature for System Integrators (SIs) and Independent Software Vendors (ISVs) when integrating with other systems.

Note You should only modify filter settings using the advanced filter view under strict advice and supervision from Axway Support.

Enable advanced filter view

To enable the advanced filter view for a filter in the Policy Studio, press the **Shift** key when opening the filter. For example, you can press **Shift**, and double-click a filter on the policy canvas. Alternatively, you can press **Shift**, right-click the filter in the Policy Studio tree or policy canvas, and select **Edit**.

In the advanced filter view, settings are displayed with the following characters before the field name:

- Required: * (for example, ***name**)
- Reference: ^ (for example, **^proxyServer**)
- Radio attribute: **(:)** (for example, **(:)httpAuthType**)

Edit filter settings

You can specify all fields in this view using text values (for example, values such as `http://stockquote.com/stockquote/instance1`, `false`, `0`, `-1`, `500`, and so on). Alternatively, you can use the API Gateway selector syntax to expand values at runtime. The following example selector expands the user agent header sent by the client in the `http.headers` message attribute:

```
${http.headers["User-Agent"]}
```

For example, this selector might return a user agent header such as the following at runtime:

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.7 (KHTML, like Gecko)
Chrome/16.0.912.77 Safari/535.7
```

To confirm your updates, you must click **Save Changes** at the bottom right of the dialog. Alternatively, at any stage you can click **Restore Defaults** to return to the original factory settings.

For more details on the API Gateway selector syntax, see *Select configuration values at runtime* on page 421.

Return to default filter view

When you have finished editing filter settings in the advanced filter view, deselect the **Show Advanced Filter View** setting in **Preferences**. Then when you edit a selected filter on the policy canvas, the default user-friendly view for the filter is displayed.

Select configuration values at runtime

Overview

A selector is a special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata values (for example, from message attributes, a Key Property Store (KPS), or environment variables). The selector syntax uses the Java Unified Expression Language (JUEL) to evaluate and expand the specified values. Selectors provide a powerful feature when integrating with other systems or when customizing and extending the API Gateway.

When you press the **Shift** key and open a filter, you can edit all filter settings using text values in the advanced filter view. This means that you can specify all filter fields using the API Gateway selector syntax. For more details on the advanced view, see *Advanced filter view* on page 420.

Selector syntax

The API Gateway selector syntax uses JUEL to evaluate and expand the following types of values at runtime:

- Message attribute properties configured in message filters, for example:

```
${authentication.subject.id}
```

- Environment variables specified in `envSettings.props` and `system.properties` files, for example:

```
${env.PORT.MANAGEMENT}
```

- Values stored in a KPS table, for example:

```
${kps.CustomerProfiles[JoeBloggs].age}
```

Note Do not use hyphens (`-`) in selector expressions. Hyphens are not supported by the Java-based selector syntax. You can use underscores (`_`) instead.

Access selector fields

A message attribute selector can refer to a field of that message (for example `certificate`), and you can use `.` characters to access subfields. For example, the following selector expands to the `username` field of the object stored in the `profile` attribute in the message:

```
${profile.username}
```

You can also access fields indirectly using square brackets (`[` and `]`). For example, the following selector is equivalent to the previous example:

```
${profile[field]}
```

You can specify literal strings as follows:

```
${profile["a field name with spaces"]}
```

For example, the following selector uses the `kathy.adams@acme.com` key value to look up the `User` table in the KPS, and returns the value of the `age` property:

```
${kps.User["kathy.adams@acme.com"].age}
```

Note For backwards compatibility with the `.` spacing characters used in previous versions of the API Gateway, if a selector fails to resolve with the above rules, the flat, dotted name of a message attribute still works. For example, `${content.body}` returns the item stored with the `content.body` key in the message.

Special selector keys

The following top-level keys have a special meaning:

Key	Description
`kps`	Subfields of the `kps` key reflect the alias names of KPS tables in the API Gateway group. Further indexes represent properties of an object in a table (for example, `${kps.User["kathy.adams@acme.com"].age}`).
`env`, `system`	In previous versions, fields from the `envSettings.props` and `system.properties` files had restrictions on prefixes. The selector syntax does not require the `env` and `system` prefixes in these files. For example, `${env.` selects settings from `envSettings.props`, and the rest of the selector chooses any properties in it. However, for compatibility, if a setting in either file starts with this prefix, it is stripped away so the selectors still behave correctly with previous versions.

Resolve selectors

Each `${...}` selector string is resolved step-by-step, passing an initial context object (for example, `Message`). The top-level key is offered to the context object, and if it resolves the field (for example, the message contains the named attribute), the resolved object is indexed with the next level of key. At each step, the following rules apply:

1. At the top level, test the key for the global values (for example, `kps`, `system`, and `env`) and resolve those specially.
2. If the object being indexed is a Dictionary, KPS, or Map, use the index as a key for the item's normal indexing mechanism, and return the resulting lookup.
3. If all else fails, attempt Java reflection on the indexed object.

Note Method calls are currently only supported using Java reflection. There are currently no supported functions as specified by the Unified Expression Language (EL) standard. For more details on JUEL, see http://juel.sourceforge.net/.

Example selector expressions

This section lists some example selectors that use expressions to evaluate and expand their values at runtime.

Message attribute

The following message attribute selector returns the HTTP `User-Agent` header:

```
${http.headers["User-Agent"]}
```

For example, this might expand to the following value:

```
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.7 (KHTML, like Gecko)
Chrome/16.0.912.77 Safari/535.7
```

Environment variable

In a default configuration, the following environment variable selector returns port 8091:

```
${env.PORT.MANAGEMENT + 1}
```

Key Property Store

The following selector looks up a KPS table with an alias of `User`:

```
${kps.User[http.querystring.id].firstName}
```

This selector retrieves the object whose Policy Studio key value is specified by the `id` query parameter in the incoming HTTP request, and returns the value of the `firstName` property in that object.

The following selector explicitly provides the key value, and returns the value of the `age` property:

```
${kps.User["kathy.adams@acme.com"].age}
```

In this example, the ASCII `"` character is used to delimit the key string.

The following selector looks up a KPS table with a composite secondary key of `firstName,lastName`:

```
${kps.User[http.querystring.firstName][http.querystring.lastName].email}
```

In this example, the key values are received as query parameters in the incoming HTTP request. The selector returns the value of the `email` property from the resulting object.

For more details, see *Key Property Store* on page 258.

Examples using reflection

The following message attribute selector returns the CGI argument from an HTTP URL (for example, returns `bar` for `http://localhost/request.cgi?foo=bar`):

```
${http.client.getCgiArgument("foo")}
```

This returns the name of the top-level element in an XML document:

```
${content.body.getDocument().getDocumentElement().getNodeName()}
```

This returns `true` if the HTTP response code lies between 200 and 299:

```
${http.response.status >= 200 && http.response.status <= 299}
```

Tip You can use the **Trace** filter to determine the appropriate selector expressions to use for specific message attributes. When configured after another filter, the **Trace** filter outputs the available message attributes and their Java type (for example, `Map` or `List`). For details on `com.vordel` classes, see the API Gateway Javadoc available from Axway Support at https://support.axway.com.

For example, for the `OAuth2AccessToken` class, you can use selector expressions such as `${accesstoken.getAdditionalInformation()}`.

Extract message attributes

There are a number of API Gateway filters that extract message attribute values (for example, **Extract Certificate Attributes** and **Retrieve from HTTP Header**). Using selectors to extract message attributes offers a more flexible alternative to using such filters. For more details on using selectors instead of these filters, contact Axway Support.

Scripting language filter

Overview

The **Scripting Language** filter uses the Java Specification Request (JSR) 223 to embed a scripting environment in the API Gateway core engine. This enables you to write custom JavaScript, Groovy, or Jython code to interact with the message as it is processed by the API Gateway. You can get, set,

and evaluate specific message attributes with this filter.

Some typical uses of the **Scripting Language** filter include the following:

- Check the value of a specific message attribute
- Set the value of a message attribute
- Remove a message attribute
- DOM processing on the XML request or response

Write a custom script

To write a custom script, you must implement the `invoke()` method. This method takes a `com.vordel.circuit.Message` object as a parameter and returns a boolean result.

The API Gateway provides a **Script Library** that contains a number of prewritten `invoke()` methods to manipulate specific message attributes. For example, there are `invoke()` methods to check the value of the `SOAPAction` header, remove a specific message attribute, manipulate the message using the DOM, and assign a particular role to a user.

You can access the script examples provided in the **Script library** by clicking the **Show script library** button on the filter's main configuration window.

Note When writing the JavaScript, Groovy, or Jython code, you should note the following:

- You must implement the `invoke()` method. This method takes a `com.vordel.circuit.Message` object as a parameter, and returns a boolean.
- You can obtain the value of a message attribute using the `get()` method of the `Message` object. However, do not perform attribute substitution as follows:

```
msg.get("my.attribute.a") == msg.get("my.attribute.b")
```

  This is not thread safe and can cause performance issues.

Use local variables

The API Gateway is a multi-threaded environment, therefore, at any one time multiple threads can be executing code in a script. When writing JavaScript, Groovy, or Jython code, always declare variables locally using `var`. Otherwise, the variables are global, and global variables can be updated by multiple threads.

For example, always use the following approach:

```
var myString = new java.lang.String("hello word");
for (var i = 100; i < 100; i++) {
   java.lang.System.out.println(myString + java.lang.Integer.toString(i));
}
```

Do not use the following approach:

```
myString = new java.lang.String("hello word");
for (i = 100; i < 100; i++) {
   java.lang.System.out.println(myString + java.lang.Integer.toString(i));
}
```

Using the second example under load, you cannot guarantee which value is output because both variables (`myString` and `i`) are global.

Add your script JARs to the classpath

You must add your custom JavaScript, Groovy, or Jython JAR files to your API Gateway classpath and to the list of runtime dependencies in Policy Studio.

Add your script JARs to the API Gateway classpath

Because the scripting environment is embedded in the API Gateway engine, it has access to all Java classes on the API Gateway classpath, including all JRE classes.

If you wish to invoke a Java object, you must place its corresponding class file on the API Gateway classpath. The recommended way to add classes to the API Gateway classpath is to place them (or the JAR files that contain them) in the `INSTALL_DIR/ext/lib` folder. For more details, see the `readme.txt` in this folder.

Add your script JARs to Policy Studio

Your custom JavaScript, Groovy, or Jython script JARs must also be compiled and validated in Policy Studio. To add your JARs files to the list of runtime dependencies in Policy Studio, perform the following steps:

1. In the Policy Studio main menu, select **Window** > **Preferences** > **Runtime Dependencies**.
2. Click **Add** to select your script JAR file(s) and any other required third-party JARs.
3. Click **Apply** when finished. Copies of these JAR files are added to the `plugins` directory in your Policy Studio installation.
4. You must restart Policy Studio and the server for these changes to take effect. You should restart Policy Studio using the `policystudio -clean` command.

See also *Policy Studio preferences* on page 180.

Configure a script filter

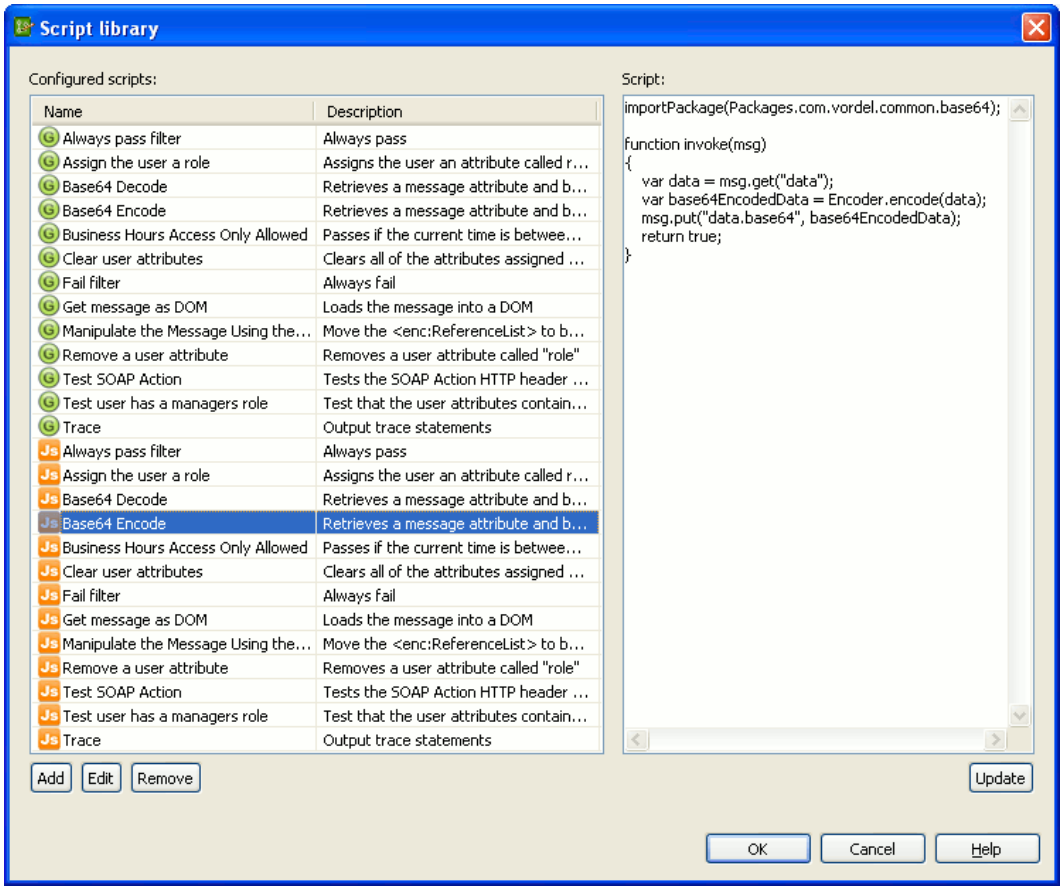
You can write or edit the JavaScript, Groovy, or Jython code in the text area on the **Script** tab. A JavaScript function skeleton is displayed by default. Use this skeleton code as the basis for your JavaScript code. You can also load an existing JavaScript or Groovy script from the **Script library** by clicking the **Show script library** button.

On the **Script library** dialog, click any of the **Configured scripts** in the table to display the script in the text area on the right. You can edit a script directly in this text area. Make sure to click the **Update** button to store the updated script to the **Script library**.

Add a script to the library

To add a new script to the library, perform the following steps:

1. Click the **Add** button, which displays the **Script Details** dialog.
2. Enter a **Name** and a **Description** for the new script in the fields provided.
3. By default, the **Language** field is set to **JavaScript**, but you can select from the following options:
    - **Groovy**: Groovy script
    - **JavaScript**: JavaScript based on Nashorn JavaScript engine (JRE8)
    - **JavaScript (Rhino engine JRE7 and earlier)**: JavaScript based on Rhino JavaScript engine (JRE7 and earlier)
    - **Jython**: Jython script
4. Write the script in the **Script** text area on the right. For example:



Note Regular expressions specified in Scripting Language filters use the regular expression engine of the relevant scripting language. For example, JavaScript-based filters use JavaScript regular expressions. This includes regular expressions inside XSDs defined by the W3C XML Schema standard.

Other API Gateway filters that use regular expressions are based on `java.util.regex.Pattern`, unless stated otherwise.

Further information

For more details on using scripts to extend API Gateway, see the *API Gateway Developer Guide*.

Common settings 11

This section describes how to configure settings common to API Gateway components:

Compressed content encoding

Overview

The API Gateway supports HTTP content encoding for the `gzip` and `deflate` compressed content encodings. This enables the API Gateway to compress files and deliver them to clients (for example, web browsers) and to back-end servers.

For example, HTML, text, and XML files can be compressed to approximately 20% to 30% of their original size, thereby saving on server traffic. Compressing files causes a slightly higher load on the server, but this is compensated by a significant decrease in client connection times to the server. A client that takes six seconds to download an uncompressed XML file might only take two seconds for a compressed version of the same file.

In Policy Studio, an **Input Encodings** list specifies the content encodings that the API Gateway can accept from peers, and an **Output Encodings** list specifies the content encodings that the API Gateway can apply to outgoing messages. You can configure these settings globally, per remote host, or per listening interface.

Encoding of HTTP responses

Content encoding of HTTP responses is negotiated using the `Accept-Encoding` HTTP request header. This enables a client to indicate its willingness to receive a particular encoding in this header. The server can then choose to encode the response with one of these client-supported encodings, and indicate this with the `Content-Encoding` header.

When the API Gateway is acting as a client communicating with a server, it uses the currently configured **Input Encodings** list to format the `Accept-Encoding` header sent to the server, thereby requesting the server to apply one of these encodings to it. If the server decides to apply one of these encodings, the API Gateway automatically inflates the compressed response when it is received.

When acting as a server, the API Gateway selects an output encoding from the intersection of what the client specified in its `Accept-Encoding` header, and the currently configured **Output Encodings**, and applies that encoding to the response.

Encoding of HTTP requests

Because a request arrives unsolicited from a client to a server, there is not normally a chance to negotiate the server's ability to process any optional features, so the automatic negotiation provided by the `Accept-Encoding` header is not available.

When acting as a client, the API Gateway selects the first configured encoding from the **Output Encodings** list to encode its request to the server. If the server fails to accept this encoding, it most likely responds with an HTTP 415 error, and the API Gateway treats this as a general HTTP error. Therefore, if the server is unable to accept content encodings, the API Gateway must be configured not to send them to that particular server.

By default, the API Gateway always accepts any supported encoding from clients, regardless of settings. For example, if a client sends gzipped content, the API Gateway inflates it regardless of configured settings.

Delimit the end of an HTTP message

HTTP sessions can encode a number of request/response pairs. The rules for delimiting the end of each message and the start of the next one are well defined, but complex due to requirements for historical compatibility, and poor support from HTTP entities.

HTTP requests

There are two ways to delimit the end of an HTTP request:

- A `Content-Length` header in the request indicates to the server the exact length of the payload entity following the HTTP headers, and can be used by the receiving server to locate the end of that entity.
- Alternatively, an HTTP/1.1 server should accept chunked transfer encoding, which precedes each chunk of the entity with a length, until a zero-length chunk indicates the end.

The benefit of using chunked transfer encoding is that the client does not need to know the length of the transmitted entity when it sends HTTP request headers (unlike when inserting a `Content-Length` header).

Because the API Gateway compresses the requests on the fly, it is prohibitively expensive to calculate the content length before compressing the body. As a result, outbound content encoding is only supported when talking to HTTP/1.1 servers that support chunked transfer encoding.

Note All HTTP/1.1 servers are required to support chunked transfer encoding, but unfortunately some do not, so you can use **Remote Host** settings to configure whether a destination is capable of supporting the chunked encoding in HTTP/1.1, regardless of its advertised HTTP protocol version. For more details, see *Configure remote host settings* on page 268.

HTTP responses

For HTTP responses, the server has three options for delimiting the end of the entity. The two mentioned above, and also the ability to close the HTTP connection after the response is transmitted. The receiving client can then infer that the entire message has been received due to the normal end of the TCP/IP session.

When content encoding responses, the API Gateway avoids using `Content-Length` headers in the response, and uses chunked encoding or TCP/IP connection closure to indicate the end of the response. This means that content encoding of responses is supported for HTTP/1.0 or HTTP/1.1 clients.

Configure content encodings

In Policy Studio, you can configure HTTP content encodings in the **Content Encodings** dialog. You can configure the following settings globally per API Gateway instance, per remote host, or per listening interface:

Input Encodings	Specifies the content encodings that the API Gateway can accept from peers.
Output Encodings	Specifies the content encodings that the API Gateway can apply to outgoing messages.

The available content encodings include `gzip` and `deflate`. By default, the content encodings configured in **Environment Configuration** > **Server Settings** > **General** are used, which apply at the API Gateway instance level. The default is no content encodings.

You can override these settings at the listening interface level (on the **Advanced** tab of the HTTP or HTTPS Interface dialog), and the remote host level (on the **Advanced** tab of the Remote Host Settings dialog).

Add content encodings

To add content encodings, click the browse button next to the **Input Encodings** or **Output Encodings** field, and perform the following steps in the **Content Encodings** dialog:

1. Deselect the **Use Default** setting.
2. Select the content encodings to add in the **Available Content Encodings** list on the left.
3. Click the **>** or **>>** button to move the content encodings to the **Content Encodings** list on the right, which displays the active settings. You can also double-click a content encoding to move it to the right or left.
4. Click **OK**. This displays the configured encodings in the **Input Encodings** or **Output Encodings** field (for example, `gzip`, `deflate`).

Configure no content encodings

To configure no content encodings, deselect the **Use Default** setting, and click **OK**. This displays `None` in the **Input Encodings** or **Output Encodings** field.

Note You can select the **Use Default** setting to switch to the **Default Settings** without losing your original content encoding selection.

Further information

For more details on the different levels at which you can configure content encodings in Policy Studio, see the following topics:

- *Configure remote host settings* on page 268
- *Configure HTTP services* on page 275
- General settings in the *API Gateway Administrator Guide*.

For more details on HTTP content encoding, see HTTP RFC 2616.

Configure connection groups

A *connection group* consists of a number of external servers that API Gateway connects to (for example, RSA Access Manager servers for authorization).

API Gateway attempts to connect to all the servers in the group in a round-robin fashion, therefore providing a high degree of failover. If one or more servers are unavailable, API Gateway can still connect to an alternative server. API Gateway attempts to connect to the listed servers according to the priorities assigned to them.

For example, assume there are two high-priority servers, one medium-priority server, and one low-priority server configured. Assuming API Gateway can successfully connect to the two high-priority servers, it alternates requests only between these two servers in a round-robin fashion. The other group servers are not used. However, if both high-priority servers become unavailable, API Gateway then tries to use the medium-priority server. Only if this fails is the low-priority server used.

Connection groups are available in the Policy Studio tree under the **Environment Configuration > External Connections > Connection Sets** node according to the filter from which they are available. For example, connection sets under the **RSA Access Manager Connection Sets** node are available in the RSA Access Manager filter. For more details, see "RSA Access Manager authorization" in the *API Gateway Policy Developer Filter Reference*.

Configure a connection group

You can configure a connection group using the **Connection Group** dialog. The external servers are listed in order of priority in the table on the **Connection Group** dialog.

API Gateway attempts to connect to the server at the top of the list first. If this server is not available, a connection attempt is made to the second server, and so on until an available server is found. If none of the listed servers are available, the client is not authorized, and a SOAP fault is returned to the client.

You can increase or decrease the priorities of the listed external servers using the **Up** and **Down** buttons. You can add, edit, and delete connections in the group using the **Add**, **Edit**, and **Remove** buttons.

Configure a connection

You can configure a connection within a connection group using the **Connection Configuration** dialog. Perform the following steps:

1. Enter the name or IP address of the machine hosting the selected Access Manager server in the **Location** field.
2. Enter the **Port** on which the specified Access Manager server is listening.
3. Select a suitable **Timeout** in seconds for connections to this server.
4. Select the appropriate **Connection Type** for API Gateway to use when connecting to the specified Access Manager server. The available connections types are clear, over Anonymous SSL, or Mutual SSL Authentication (two-way SSL).
5. If you select **SSL Authentication**, you must also select an **SSL mutual authentication certificate**. This certificate is then used to authenticate to the Access Manager server.

Configure cron expressions

Overview

Cron expressions are used in policy execution scheduling, and within several filters (for example, the **Time** utility filter).

The **Cron Dialog** enables you to create a cron expression used to trigger regularly occurring events (for example, generate a report, or block or allow messages at specified times). You can use the time tabs in this dialog to guide you through the configuration steps.

Alternatively, enter the cron expression value directly in the text boxes. When you have created the cron expression, you can click the **Test Cron** button to test the value of the cron expression and see when it is next due to fire.

For background information and details on cron expression syntax, see *Policy execution scheduling* on page 241.

Create a cron expression using the time tabs

Using the time tabs in the dialog to guide you through the configuration steps is the default option. You can create a cron expression to trigger at the specified times using the following settings:

Seconds

Select one of the following options:

- **Every Second of the Minute**

  Fires every second of the minute. This is the default setting.

- **Just on Second**

  Fires only on the specified second of the minute.

- **Range from Second**

  Fires over a range of seconds. For example, if the first value is 10 and the second value is 25, the trigger starts firing on second 10 of the minute and continues to fire for 15 seconds.

- **Start on Second**

  Fires on the specified second of the minute and repeats every specified number of seconds. For example, if the first number is 15 and the second number is 30, the trigger fires at 15 seconds and repeats every 30 seconds until stopped.

- **On Multiple Seconds**

  Fires on the specified seconds of each minute. Enter a comma separated list of seconds (values of `0-59` inclusive). For example, `10,20,30`.

Minutes

Select one of the following options:

- **Every Minute of the Hour**

  Fires every minute of the hour. This is the default setting.

- **Just on Minute**

Fires only on the specified minute of the hour.

- **Range from Minute**

  Fires over a range of minutes. For example, if the first value is 5 and the second value is 15, the trigger starts firing on minute 5 of the hour and continues to fire for 10 minutes.

- **Start on Minute**

  Fires on the specified minute of the hour and repeats every specified number of minutes. For example, if the first number is 10 and the second number is 20, the trigger fires at 10 minutes and repeats every 20 minutes until stopped.

- **On Multiple Minutes**

  Fires on the specified minute of each hour. Enter a comma-separated list of minutes (values of `0-59` inclusive). For example, `5,15,30`.

Hours

Select one of the following options:

- **Every Hour of the Day**

  Fires every hour of the day. This is the default setting.

- **Just on Hour**

  Fires only on the specified hour of the day.

- **Range from Hour**

  Fires over a range of hours. For example, if the first value is 9 and the second value is 17, the trigger starts firing on hour 9 of the day and continues to fire for 8 hours.

- **Start on Hour**

  Fires on the specified hour of the day and repeats every specified number of hours. For example, if the first number is 6 and the second number is 2, the trigger fires at hour 6 and repeats every 2 hours until stopped.

- **On Multiple Hours**

  Fires on the specified hours of each day. Enter a comma-separated list of hours (values of `0-23` inclusive). For example, `6,12,18`.

- **Multiple Ranges**

  Fires on the specified ranges of hours of each day. Enter comma separated ranges of hours (values of `0-23` inclusive). For example, `9-1,14-17`.

Day

You must first select **Day of Week** or **Day of Month** from the drop-down list (using both of these fields in not supported). **Day of Month** is selected by default.

Day of Month

Select one of the following options:

- **Every Day of the Month**

  Fires every day of the month. This is the default setting.

- **Just on Day**

  Fires only on the specified day of the month.

- **Range from Day**

  Fires over a range of days. For example, if the first value is 7 and the second value is 14, the trigger starts firing on day 7 of the month and continues to fire for 9 days.

- **Start on Day**

  Fires on the specified day of the month and repeats every specified number of days. For example, if the first day is 2 and the second number is 5, the trigger fires at day 2 and repeats every 5 days until stopped.

- **On Multiple Days**

  Fires on the specified days of each month. Enter a comma separated list of days (values of `1-32` inclusive). For example, `1,14,21,28`.

- **Last Day of the Month**

  Fires on the last day of each month (for example, 31 January, or 28 February in non-leap years).

- **Last Week Day of the Month**

  Fires on the last week day of each month (Monday-Friday inclusive only).

Day of Week

Select one of the following options:

- **Every Day of the Week**

  Fires every day of the week. This is the default setting.

- **Just on *Day***

  Fires only on the specified day of the week. Defaults to `SUN`.

- **Range from *Day***

  Fires over a range of days. For example, if the first value is `MON` and the second value is `FRI`, the trigger starts firing on `MON` and continues to fire until `FRI`.

- **Start on *Day***

  Fires on the specified day of the week and repeats every specified number of days. For example, if the first day is `TUES` and the number is 3, the trigger fires on `TUES` and repeats every 3 days until stopped.

- **On Multiple *Days***

Fires on the specified days of each week. Enter a comma separated list of days. For example, `MON,WED,FRI`.

- **Last Day of the Week**

  Fires on the last day of each week (`SAT`).

- **On the *Nth Day***

  Fires on the Nth day of the week of each month (for example, the second `FRI` of each month).

Month

Select one of the following options:

- **Every Month of the Year**

  Fires every month of the year. This is the default setting.

- **Just on *Month***

  Fires only on the specified month of the year. Defaults to `JAN`.

- **Range from *Month***

  Fires over a range of months. For example, if the first value is `MAY` and the second value is `JUL`, the trigger starts firing on `MAY` and continues to fire until `JUL`.

- **Start on *Month***

  Fires on the specified month of the year and repeats every specified number of months. For example, if the first month is `FEB` and the number is 2, the trigger fires in `FEB` and repeats every 2 months until stopped.

- **On Multiple *Months***

  Fires on the specified months of each year. Enter a comma-separated list of months (values of `JAN-DEC` or `1-12` inclusive). For example, `MAR,JUN,SEPT`.

Year

Select one of the following options:

- **Every Year**

  Fires every year. This is the default setting.

- **No Specific Year**

  Fires no specific year.

- **Just on *Year***

  Fires only on the specified year. Defaults to current year.

- **Range from *Year***

  Fires over a range of years. For example, if the first value is `2012` and the second value is `2015`, the trigger starts firing on `2012` and continues to fire until `2015`.

- **Start on *Year***

  Fires on the specified year and repeats every specified number of years. For example, if the first year is `2012` and the number is 2, the trigger fires in `2012` and repeats every 2 years until stopped.

- **On Multiple *Years***

  Fires on the specified Year. Enter a comma-separated list of years (for example, `2012,2013,2015`).

Enter a cron expression

To enter the cron expression value directly in the dialog, click **Create cron expression using edit boxes**, and enter the values in the appropriate boxes. For example, the following cron expression fires on April 27 and 28, at any time except between 10:00:01 and 10:59:59:

```
* * 0-9,11-23 27-28 APR ?
```

For details on cron expression syntax and special characters, see *Policy execution scheduling* on page 241.

Test the cron expression

When you have configured the cron expression using either approach, click the **Test Cron** button to test the syntax of the cron expression value and view when it is next due to fire. If the configured cron expression is invalid, a warning dialog is displayed.

Results

The test results include the following output:

Expression	Displays the configured cron expression. For example: `* * 9-17 * * ? *`
Next Fire Times	Displays when cron expression is next due to fire. For example: `Next fire event:Fri Jul 22 10:26:09 EST 2012`

Further information

For details on using the **Cron Dialog** to create cron expressions that trigger regularly occurring events (for example, generate reports, or block or allow messages at specified times), see the following topics:

- The **Time** filter in the *API Gateway Policy Developer Filter Reference*
- Configure scheduled reports in the *API Gateway Administrator Guide*

Signature location

Overview

A given XML message can contain several XML signatures. Consider an XML document (for example, a company policy approval form) that must be digitally signed by a number of users (for example, department managers) before being submitted to the destination web service (for example, a company policy approval web service). Such a message contains several XML signatures by the time it is ready to be submitted to the web service.

In such cases, where multiple signatures are present within a given XML message, it is necessary to specify which signature the API Gateway should use in the validation process. You can specify the location of the signature in the XML message in the **XML Signature Verification** filter. For more information on validating XML signatures, see "XML signature verification" in the *API Gateway Policy Developer Filter Reference*.

Signature location options

The API Gateway can extract the signature from an XML message using several different methods:

- WS-Security block
- SOAP message header
- Advanced (XPath)

Select the most appropriate method from the **Signature Location** field. Your selection depends on the types of SOAP messages that you expect to receive. For example, if incoming SOAP messages contain an XML signature within a WS-Security block, you should choose this option from the list.

Use WS-Security actors

If the signature is present in a WS-Security block:

1. Select `WS-Security block` from the **Signature Location** field.
2. Select a SOAP actor from the **Select Actor/Role(s)** field. Each actor uniquely identifies a separate WS-Security block. By selecting `Current actor/role only` from the list, the WS-Security block with no actor is taken.
3. In cases where there might be multiple signatures within the WS-Security block, it is necessary to extract one using the **Signature Position** field.

The following is a skeleton version of a message where the XML signature is contained in the `sample` WS-Security block, (`soap-env:actor="sample"`):

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
s:actor="sample">
      <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="s1">
        ....
      </dsig:Signature>
    </wsse:Security>
  </s:Header>
  <s:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
    </ns1:getTime>
  </s:Body>
</s:Envelope>
```

Use SOAP header

If the signature is present in the SOAP header:

1. Select `SOAP message header` from the **Signature Location** field.
2. If there is more than one signature in the SOAP header, then it is necessary to specify which signature the API Gateway should use. Specify the appropriate signature by setting the **Signature Position** field.

The following is an example of an XML message where the XML signature is contained within the SOAP header:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="s1">
      ....
    </dsig:Signature>
  </s:Header>
  <s:Body>
    <ns1:getTime xmlns:ns1="urn:timeservice">
    </ns1:getTime>
  </s:Body>
</s:Envelope>
```

Use XPath expression

To use an XPath expression to locate the signature:

1. Select `Advanced (XPath)` from the **Signature Location** field.
2. Select an existing XPath expression from the list, or add a new one by clicking on the **Add** button. XPath expressions can also be edited or removed with the **Edit** and **Remove** buttons.

The `First signature in SOAP Header` XPath expression takes the first signature from the SOAP header. The expression is as follows:

```
//s:Envelope/s:Header/dsig:Signature[1]
```

To edit this expression, click the **Edit** button to display the **Enter XPath Expression** dialog.

An example of a SOAP message containing an XML signature in the SOAP header is provided below.

```
<?xml version="1.0" encoding="UTF-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="s1">
      ....
    </dsig:Signature>
  </s:Header>
  <s:Body>
    <product xmlns="http://www.axway.com">
      <name>SOA Product*</name>
      <company>Company</company>
      <description>Web Services Security</description>
    </product>
  </s:Body>
</s:Envelope>
```

Because the elements referenced in the expression (`Envelope` and `Signature`) are prefixed elements, you must define the namespace mappings for each of these elements as follows:

Prefix	URI
s	http://schemas.xmlsoap.org/soap/envelope/
dsig	http://www.w3.org/2000/09/xmldsig#

When adding your own XPath expressions, you must be careful to define any namespace mappings in a manner similar to that outlined above. This avoids any potential clashes that might occur where elements of the same name, but belonging to different namespaces, are present in an XML message.

Configure URL groups

Overview

The API Gateway can make connections on a round-robin basis to the URLs listed in a URL group, thus enabling a high degree of failover to external servers (for example, Entrust GetAccess, SAML PDP, or XKMS). The API Gateway attempts to connect to the listed servers according to the priorities assigned to them.

For example, assume there are two High priority URLs, one Medium URL, and one Low URL configured. Assuming the API Gateway can successfully connect to the two High priority URLs, it alternates requests between these two URLs only in a round-robin fashion. The other group URLs are not used. However, if both of the High priority URLs become unavailable, the API Gateway then tries to use the Medium priority URL, and only if this fails is the Low priority URL used.

URL groups are available in the Policy Studio tree under the **Environment Configuration** > **External Connections** > **URL Connection Sets** node. For example, URL sets under the **XKMS URL Sets** node are available in XKMS connections. For more details, see *Configure XKMS connections* on page 418.

Configure a URL group

Configure the following fields:

URL Group Name:
Enter a name for the URL group.

To add URLs to the group, click the **Add** button and complete the following fields:

- **URL**:
  Enter the full URL of the external server.
- **Timeout (secs)**:
  Specify the timeout in seconds for connections to the specified server.
- **Retry After (secs)**:
  Whenever the server becomes unavailable for whatever reason (for example, maintenance), no attempt is made to connect to that server until the time specified here has elapsed. In other words, when a connection failure is detected, the next connection to that URL is after this amount of time.
- **SSL mutual authentication certificate**:
  If the specified server requires clients to authenticate to it over two-way SSL, click the **Signing Key** button to select an SSL certificate from the Certificate Store for authentication.
- **Host/IP**:
  If the specified server sits behind a proxy server, you must enter the host name or IP address of the proxy server.

- **Port**:
  Enter the port on which the proxy is listening.

To edit or delete a URL, select the URL from the table, and click the **Edit** or **Delete** buttons.

In general, the API Gateway attempts to round-robin requests over URLs of the same priority, but uses higher priority URLs before lower priority ones. When a new URL is added to the group, it is automatically given the highest priority. You can change priorities by selecting the URL in the table, and clicking the **Up** or **Down** buttons.

Configure XPath expressions

Overview

The API Gateway uses XPath expressions in a number of ways. These include to locate an XML signature in a SOAP message, to determine what elements of an XML message to validate against an XML schema, to check the content of a particular element within an XML message, amongst many more uses. For example, see the **Locate XML Nodes**, **Content Validation**, or **XML Signature Generation** filters in the *API Gateway Policy Developer Filter Reference*.

You can configure XPath expressions as follows:

- *Manual XPath configuration* on page 444
- *XPath wizard* on page 446

Manual XPath configuration

If you are already familiar with XPath and wish to configure the expression manually, complete the following fields, using the examples below if necessary:

1. Enter or select a name for the XPath expression in the **Name** drop-down list.
2. Enter the XPath expression to use in the **XPath Expression** field.
3. In order to resolve any prefixes within the XPath expression, the namespace mappings (Prefix , URI) should be entered in the table.

For example, consider the following SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sig1">
      ..............
    </dsig:Signature>
  </soap:Header>
  <soap:Body>
    <prod:product xmlns:prod="http://www.company.com">
```

```
      <prod:name>SOA Product</prod:name>
      <prod:company>Company</prod:company>
      <prod:description>WebServices Security</prod:description>
    </prod:product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the `<company>` element contains the value `Company`:

```
//prod:company[text()='Company']
```

In this case, you must define a mapping for the *prod* namespace as follows:

Prefix	URI
prod	http://www.company.com

In another example, the element to be examined by the XPath expression belongs to a default namespace. Consider the following SOAP message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" id="sig1">
      ..............
    </dsig:Signature>
  </soap:Header>
  <soap:Body>
    <product xmlns="http://www.company.com">
      <name>SOA Product</name>
      <company>Company</company>
      <description>Web Services Security</description>
    </product>
  </soap:Body>
</soap:Envelope>
```

The following XPath expression evaluates to true if the `<company>` element contains the value `Company`:

```
//ns:company[text()='Company']
```

Because the `<company>` element belongs to the default (`xmlns`) namespace (`http://www.company.com`), you must make up an arbitrary prefix (`ns`) for use in the XPath expression, and assign it to `http://www.company.com`. This is necessary to distinguish between potentially several default namespaces which might exist throughout the XML message. The following mapping illustrates this:

Prefix	URI
ns	http://www.company.com

Return a nodeset

Both of the examples above dealt with cases where the XPath expression evaluated to a Boolean value. For example, the expression in the above example asks does the <company> element in the http://www.company.com namespace contain a text node with the value Company.

It is sometimes necessary to use the XPath expression to return a subset of the XML message. For example, when using an XPath expression to determine what nodes should be signed in a signed XML message, or when retrieving the nodeset to validate against an XML Schema.

The API Gateway ships with an XPath expression that returns All Elements inside SOAP Body. To view this expression, select it from the **Name** field. It appears as follows:

```
/soap:Envelope/soap:Body//*
```

This XPath expression simply returns all child elements of the SOAP <Body> element. To construct and test more complicated expressions, use the XPath wizard.

XPath wizard

The XPath wizard assists you in creating correct and accurate XPath expressions. The wizard enables you to load an XML message and then run an XPath expression on it to determine what nodes are returned. To launch the XPath wizard, click the **XPath Wizard** button on the XPath Expression dialog.

To use the XPath wizard, enter (or browse to) the location of an XML file in the **File** field. The contents of the XML file are displayed in the main window of the wizard. Enter an XPath expression in the **XPath** field, and click the **Evaluate** button to run the XPath against the contents of the file. If the XPath expression returns any elements (or returns true), those elements are highlighted in the main window.

If you are not sure how to write the XPath expression, you can select an element in the main window. An XPath expression to isolate this element is automatically generated and displayed in the **Selected** field. To use this expression, select the **Use this path** button, and click **OK**.

WS-Policy reference

Overview

This topic provides reference information on the following WS-Policies:

For details on configuring WS-Policies, see *Register and secure web services* on page 75.

AsymmetricBinding WS-Policies

WS-Policy Name	Description
AsymmetricBinding with Encrypted UsernameToken	The service exposes an `AsymmetricBinding` where the client and server use their respective X.509 v3 tokens to sign and encrypt the message. An encrypted `UsernameToken` with hash password must be included in all messages from the client to the server.
AsymmetricBinding with SAML 1.1 (Sender Vouches) Assertion and Signed Supporting Token	The service is secured with an `AsymmetricBinding` where the client and server use their respective X.509 v3 certificates to secure the message. The client must include a SAML 1.1 `Assertion` (sender vouches) in all messages it sends to the service.
AsymmetricBinding with Signed and Encrypted UsernameToken	The service exposes an `AsymmetricBinding` where the client and server use their respective X.509 v3 tokens to sign and encrypt the message. A signed and encrypted `UsernameToken` with plaintext password must be included in all messages from the client to the service.

WS-Policy Name	Description
AsymmetricBinding with WSS 1.0 Mutual Authentication with X509 Certificates, Sign, Encrypt	The service exposes an `AsymmetricBinding` interface where the client and server use their respective X.509 v3 certificates for mutual authentication, signing, and encrypting.
AsymmetricBinding with X509v3 Tokens	The service exposes an `AsymmetricBinding` where the client and server use their respective X.509 v3 tokens to sign and encrypt the message.

Message-level WS-Policies

WS-Policy Name	Description
Encrypt SOAP Body	The SOAP body must be encrypted.
Sign and Encrypt SOAP Body	The SOAP body must be signed and encrypted.
Sign SOAP Body	The SOAP body must be encrypted.

Oracle Web Services Manager WS-Policies

WS-Policy Name	Description
WS-Security 1.0 Mutual Auth with Certificates	`AsymmetricBinding` where the client and server use their respective X.509 v3 certificates to secure the message.
WS-Security 1.0 SAML with Certificates	`AsymmetricBinding` with SAML assertion as `SignedSupportingToken`.
WS-Security 1.0 Username with Certificate	`AsymmetricBinding` with WS-Security `UsernameToken` as `SignedSupportingToken`.

WS-Policy Name	Description
WS-Security 1.1 Mutual Auth with Certificates	`SymmetricBinding` where the same X.509 v3 certificate is used to secure all messages between the client and the service.
WS-Security 1.1 Username with Certificates	`SymmetricBinding` with a WS-Security `UsernameToken` as a `SignedSupportingToken`. The message is endorsed with an asymmetric `Signature`.
WS-Security SAML Token Over SSL	`TransportBinding` with a SAML Token as a `SupportingToken`.
WS-Security UsernameToken Over SSL	`TransportBinding` with a WS-Security `UsernameToken` as a `SupportingToken`.

Simple WS-Policies

WS-Policy Name	Description
SAML 1.1 Bearer	The client must include a SAML 1.1 `Assertion` (bearer) representing the requestor in all messages from the client to the service.
Username SupportingToken Hash Password	The client must authenticate with a WS-Security SAML `UsernameToken` with hash password.
Username SupportingToken No Password	The client must authenticate with a WS-Security `UsernameToken` without a password.
Username SupportingToken Plaintext Password	The client must authenticate with a WS-Security `UsernameToken` with a plaintext password.

SymmetricBinding WS-Policies

WS-Policy Name	Description
SymmetricBinding with SAML 2.0 (Sender Vouches) Assertion and Endorsing Supporting Token	The service exposes a `SymmetricBinding` that requires the client to send a SAML 2.0 `Assertion` to the service. An X.509 v3 token is also included in all messages from the client to the service as an `EndorsingSupportingToken`.
SymmetricBinding with Signed and Encrypted UsernameToken	The service uses a `SymmetricBinding` where the client and service use the same X.509 v3 token to sign and encrypt the message. A signed and encrypted `UsernameToken` with plaintext password must be included in all messages from the client to the service. The policy uses WSS SOAP Message Security 1.1 options.
SymmetricBinding with WSS 1.1 Anonymous Authentication with X.509v3, Sign, Encrypt	The service is secured by a `SymmetricBinding` where the same X.509 v3 certificate is used to secure all messages between the client and the service. Derived keys are used for signing and encrypting and Signature Confirmation is required by the policy.
SymmetricBinding with WSS 1.1 Mutual Authentication with X.509v3, Sign, Encrypt	The service exposes a `SymmetricBinding` where the same X.509 v3 certificate is used to secure all messages between the client and the service. The client also endorses the primary message signature using another X.509v3 certificate.

TransportBinding WS-Policies

WS-Policy Name	Description
SAML 1.1 Holder-of-Key over SSL	The client includes a SAML 1.1 `Assertion` (sender vouches) in all messages from the client to the service. The client provides an endorsing signature to prove that it is the holder-of-key. A `TransportBinding` is used to sign and encrypt the message.

WS-Policy Name	Description
SAML 1.1 Sender-Vouches over SSL	The client includes a SAML 1.1 `Assertion` (sender vouches) on behalf of the requestor to all messages from the client to the service. The service uses a `TransportBinding` to ensure that all messages are signed and encrypted.
SAML 2.0 Holder-of-Key over SSL	The client includes a SAML 2.0 `Assertion` (sender vouches) in all messages from the client to the service. The client provides an endorsing signature to prove that it is the holder-of-key. A `TransportBinding` is used to sign and encrypt the message.
SAML 2.0 Sender-Vouches over SSL	The client includes a SAML 2.0 `Assertion` (sender vouches) on behalf of the requestor to all messages from the client to the service. The service uses a `TransportBinding` to ensure that all messages are signed and encrypted.
SSL Transport Binding	The service is secured by SSL (HTTPS).
Username Token over SSL with no Timestamp	The service is secured over SSL (HTTPS), the client is authenticated with a `UsernameToken`, and no timestamp should be included in the `Security` header.
Username Token over SSL with Timestamp	The service is secured over SSL (HTTPS), the client is authenticated with a `UsernameToken`. The `Security` header contains a timestamp.

License acknowledgments

Axway API Gateway uses several third-party toolkits to perform specific types of processing. In accordance with the Licensing Agreements for these toolkits, the relevant acknowledgments are listed below.

Acknowledgments

Apache Software Foundation:
This product includes software developed by the Apache Software Foundation.

OpenSSL Project:
This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit.

Eric Young:
This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

James Cooper:
This product includes software developed by James Cooper.

iconmonstr:
This product includes graphic icons developed by iconmonstr.